

CSci 127: Introduction to Computer Science



hunter.cuny.edu/csci

Announcements



- Thanksgiving Break starts in 9 days.

Announcements



- Thanksgiving Break starts in 9 days.
- No CUNY classes:
Thursday-Saturday, 28-31 November.

Announcements



- Thanksgiving Break starts in 9 days.
- No CUNY classes:
Thursday-Saturday, 28-31 November.
- In response to wrap-up requests, additional challenges today with while loops and binary & hexadecimal numbers.

Today's Topics



- Design Patterns: Searching
- Python Recap
- Machine Language
- Machine Language: Jumps & Loops
- Binary & Hex Arithmetic
- Final Exam: Format

Today's Topics



- **Design Patterns: Searching**
- Python Recap
- Machine Language
- Machine Language: Jumps & Loops
- Binary & Hex Arithmetic
- Final Exam: Format

In Pairs or Triples:

Predict what the code will do:

```
def search(nums, locate):
    found = False
    i = 0
    while not found and i < len(nums):
        print(nums[i])
        if locate == nums[i]:
            found = True
        else:
            i = i+1
    return(found)

nums= [1,4,10,6,5,42,9,8,12]
if search(nums,6):
    print('Found it! 6 is in the list!')
else:
    print('Did not find 6 in the list.')
```

Python Tutor

```
def search(nums, locate):
    found = False
    i = 0
    while not found and i < len(nums):
        print(nums[i])
        if locate == nums[i]:
            found = True
        else:
            i = i+1
    return(found)

nums= [1,4,10,6,5,42,9,8,12]
if search(nums,6):
    print('Found it! 6 is in the list!')
else:
    print('Did not find 6 in the list.')
```

(Demo with pythonTutor)

Design Pattern: Linear Search

- Example of **linear search**.

```
def search(nums, locate):
    found = False
    i = 0
    while not found and i < len(nums):
        print(nums[i])
        if locate == nums[i]:
            found = True
        else:
            i = i+1
    return(found)

nums= [1,4,10,6,5,42,9,8,12]
if search(nums,6):
    print('Found it! 6 is in the list!')
else:
    print('Did not find 6 in the list.')
```

Design Pattern: Linear Search

```
def search(nums, locate):
    found = False
    i = 0
    while not found and i < len(nums):
        print(nums[i])
        if locate == nums[i]:
            found = True
        else:
            i = i+1
    return(found)

nums= [1,4,10,6,5,42,9,8,12]
if search(nums,6):
    print('Found it! 6 is in the list!')
else:
    print('Did not find 6 in the list.')
```

- Example of **linear search**.
- Start at the beginning of the list.

Design Pattern: Linear Search

```
def search(nums, locate):
    found = False
    i = 0
    while not found and i < len(nums):
        print(nums[i])
        if locate == nums[i]:
            found = True
        else:
            i = i+1
    return(found)

nums= [1,4,10,6,5,42,9,8,12]
if search(nums,6):
    print('Found it! 6 is in the list!')
else:
    print('Did not find 6 in the list.')
```

- Example of **linear search**.
- Start at the beginning of the list.
- Look at each item, one-by-one.

Design Pattern: Linear Search

```
def search(nums, locate):
    found = False
    i = 0
    while not found and i < len(nums):
        print(nums[i])
        if locate == nums[i]:
            found = True
        else:
            i = i+1
    return(found)

nums= [1,4,10,6,5,42,9,8,12]
if search(nums,6):
    print('Found it! 6 is in the list!')
else:
    print('Did not find 6 in the list.')
```

- Example of **linear search**.
- Start at the beginning of the list.
- Look at each item, one-by-one.
- Stopping, when found, or the end of list is reached.

Today's Topics



- Design Patterns: Searching
- **Python Recap**
- Machine Language
- Machine Language: Jumps & Loops
- Binary & Hex Arithmetic
- Final Exam: Format

Python & Circuits Review: 10 Weeks in 10 Minutes



A whirlwind tour of the semester, so far...

Week 1: print(), loops, comments, & turtles

Week 1: print(), loops, comments, & turtles

- Introduced comments & print():

```
#Name:  Thomas Hunter
```

← *These lines are comments*

```
#Date:  September 1, 2017
```

← *(for us, not computer to read)*

```
#This program prints:  Hello, World!
```

← *(this one also)*

```
print("Hello, World!")
```

← *Prints the string "Hello, World!" to the screen*

Week 1: print(), loops, comments, & turtles

- Introduced comments & print():

```
#Name: Thomas Hunter
```

← These lines are comments

```
#Date: September 1, 2017
```

← (for us, not computer to read)

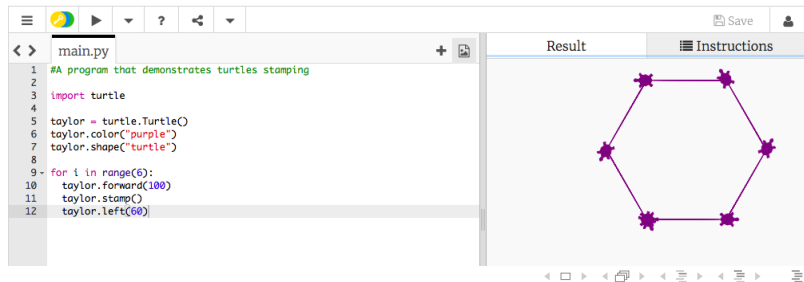
```
#This program prints: Hello, World!
```

← (this one also)

```
print("Hello, World!")
```

← Prints the string "Hello, World!" to the screen

- As well as definite loops & the turtle package:



Week 2: variables, data types, more on loops & range()

Week 2: variables, data types, more on loops & range()

- A **variable** is a reserved memory location for storing a value.

Week 2: variables, data types, more on loops & range()

- A **variable** is a reserved memory location for storing a value.
- Different kinds, or **types**, of values need different amounts of space:
 - ▶ **int**: integer or whole numbers

Week 2: variables, data types, more on loops & range()

- A **variable** is a reserved memory location for storing a value.
- Different kinds, or **types**, of values need different amounts of space:
 - ▶ **int**: integer or whole numbers
 - ▶ **float**: floating point or real numbers

Week 2: variables, data types, more on loops & range()

- A **variable** is a reserved memory location for storing a value.
- Different kinds, or **types**, of values need different amounts of space:
 - ▶ **int**: integer or whole numbers
 - ▶ **float**: floating point or real numbers
 - ▶ **string**: sequence of characters

Week 2: variables, data types, more on loops & range()

- A **variable** is a reserved memory location for storing a value.
- Different kinds, or **types**, of values need different amounts of space:
 - ▶ **int**: integer or whole numbers
 - ▶ **float**: floating point or real numbers
 - ▶ **string**: sequence of characters
 - ▶ **list**: a sequence of items

Week 2: variables, data types, more on loops & range()

- A **variable** is a reserved memory location for storing a value.
- Different kinds, or **types**, of values need different amounts of space:
 - ▶ **int**: integer or whole numbers
 - ▶ **float**: floating point or real numbers
 - ▶ **string**: sequence of characters
 - ▶ **list**: a sequence of items
e.g. [3, 1, 4, 5, 9] or ['violet', 'purple', 'indigo']

Week 2: variables, data types, more on loops & range()






- A **variable** is a reserved memory location for storing a value.
- Different kinds, or **types**, of values need different amounts of space:
 - ▶ **int**: integer or whole numbers
 - ▶ **float**: floating point or real numbers
 - ▶ **string**: sequence of characters
 - ▶ **list**: a sequence of items
e.g. [3, 1, 4, 5, 9] or ['violet', 'purple', 'indigo']
 - ▶ **class variables**: for complex objects, like turtles.

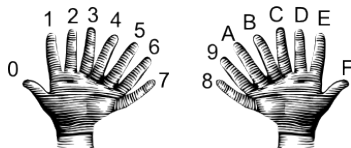
Week 2: variables, data types, more on loops & range()

- A **variable** is a reserved memory location for storing a value.
- Different kinds, or **types**, of values need different amounts of space:
 - ▶ **int**: integer or whole numbers
 - ▶ **float**: floating point or real numbers
 - ▶ **string**: sequence of characters
 - ▶ **list**: a sequence of items
e.g. [3, 1, 4, 5, 9] or ['violet', 'purple', 'indigo']
 - ▶ **class variables**: for complex objects, like turtles.
- More on loops & ranges:






```
1 #Predict what will be printed:
2
3 for num in [2,4,6,8,10]:
4     print(num)
5
6 sum = 0
7 for x in range(0,12,2):
8     print(x)
9     sum = sum + x
10
11 print(x)
12
13 for c in "ABCD":
14     print(c)
```

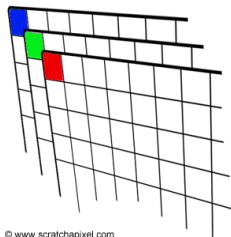
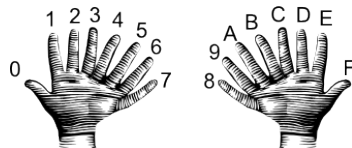
Week 3: colors, hex, slices, numpy & images

Color Name	HEX	Color
Black	#000000	
Navy	#000080	
DarkBlue	#00008B	
MediumBlue	#0000CD	
Blue	#0000FF	



Week 3: colors, hex, slices, numpy & images

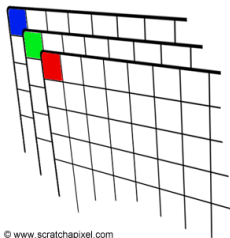
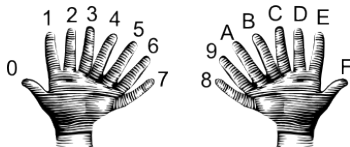
Color Name	HEX	Color
Black	#000000	
Navy	#000080	
DarkBlue	#00008B	
MediumBlue	#0000CD	
Blue	#0000FF	



© www.scratchapixel.com

Week 3: colors, hex, slices, numpy & images

Color Name	HEX	Color
Black	#000000	
Navy	#000080	
DarkBlue	#00008B	
MediumBlue	#0000CD	
Blue	#0000FF	



© www.scratchapixel.com

```
>>> a[0,3:5]  
array([3,4])
```

```
>>> a[4:,4:]  
array([[44, 45],  
       [54, 55]])
```

```
>>> a[:,2]  
array([2,12,22,32,42,52])
```

```
>>> a[2::2,::2]  
array([[20,22,24],  
       [40,42,44]])
```

0	1	2	3	4	5
10	11	12	13	14	15
20	21	22	23	24	25
30	31	32	33	34	35
40	41	42	43	44	45
50	51	52	53	54	55

Week 4: design problem (cropping images) & decisions



Week 4: design problem (cropping images) & decisions



- First: specify inputs/outputs. *Input file name, output file name, upper, lower, left, right ("bounding box")*

Week 4: design problem (cropping images) & decisions



- First: specify inputs/outputs. *Input file name, output file name, upper, lower, left, right ("bounding box")*
- Next: write pseudocode.
 - 1 Import numpy and pyplot.
 - 2 Ask user for file names and dimensions for cropping.
 - 3 Save input file to an array.
 - 4 Copy the cropped portion to a new array.
 - 5 Save the new array to the output file.

Week 4: design problem (cropping images) & decisions



- First: specify inputs/outputs. *Input file name, output file name, upper, lower, left, right ("bounding box")*
- Next: write pseudocode.
 - ① Import numpy and pyplot.
 - ② Ask user for file names and dimensions for cropping.
 - ③ Save input file to an array.
 - ④ Copy the cropped portion to a new array.
 - ⑤ Save the new array to the output file.
- Next: translate to Python.

Week 4: design problem (cropping images) & decisions

```
yearBorn = int(input('Enter year born: '))
if yearBorn < 1946:
    print("Greatest Generation")
elif yearBorn <= 1964:
    print("Baby Boomer")
elif yearBorn <= 1984:
    print("Generation X")
elif yearBorn <= 2004:
    print("Millennial")
else:
    print("TBD")

x = int(input('Enter number: '))
if x % 2 == 0:
    print('Even number')
else:
    print('Odd number')
```

Week 5: logical operators, truth tables & logical circuits

```
origin = "Indian Ocean"
winds = 100
if (winds > 74):
    print("Major storm, called a ", end="")
    if origin == "Indian Ocean" or origin == "South Pacific":
        print("cyclone.")
    elif origin == "North Pacific":
        print("typhoon.")
    else:
        print("hurricane.")

visibility = 0.2
winds = 40
conditions = "blowing snow"
if (winds > 35) and (visibility < 0.25) and \
    (conditions == "blowing snow" or conditions == "heavy snow"):
    print("Blizzard!")
```

Week 5: logical operators, truth tables & logical circuits

```
origin = "Indian Ocean"
winds = 100
if (winds > 74):
    print("Major storm, called a ", end="")
    if origin == "Indian Ocean" or origin == "South Pacific":
        print("cyclone.")
    elif origin == "North Pacific":
        print("typhoon.")
    else:
        print("hurricane.")

visibility = 0.2
winds = 40
conditions = "blowing snow"
if (winds > 35) and (visibility < 0.25) and \
    (conditions == "blowing snow" or conditions == "heavy snow"):
    print("Blizzard!")
```

in1		in2	returns:
False	and	False	False
False	and	True	False
True	and	False	False
True	and	True	True



Week 6: structured data, pandas, & more design

```
Source: https://en.wikipedia.org/wiki/Demographics_of_New_York_City,,,,,
All population figures are consistent with present-day boundaries,,,,,,
First census after the consolidation of the five boroughs,,,,,,
,,,,,
,,,,,
Year,Manhattan,Brooklyn,Queens,Bronx,Staten Island>Total
1698,4937,2017,,,727,7681
1771,21863,3623,,,2847,28423
1790,33131,45049,6159,1781,3827,49447
1800,40515,5740,6642,1755,4563,79215
1810,96373,40203,7444,2267,5347,119734
1820,123706,11187,8246,2782,6135,152056
1830,202589,20535,9049,3023,7082,242278
1840,312710,47613,14480,3344,10965,391114
1850,515547,138882,18593,8032,15061,696115
1860,813649,279122,32963,23593,25492,1174779
1870,942292,419921,45468,37393,33829,1470193
1880,1164673,599495,56559,51980,38991,1911690
1890,1441216,838547,87050,88908,51692,2507414
1900,1650093,1146582,152899,200507,67021,24372702
1910,2331542,1634351,284041,430980,85969,4766883
1920,2284103,2018296,469042,732018,116511,5620048
1930,1867312,2560461,1079129,1265258,159346,6590446
1940,1889924,2698295,1297634,1394711,174441,7454995
1950,1940101,2738275,1550849,1452177,191555,78991957
1960,1698281,2627319,1809578,1424815,221993,7781984
1970,1539233,2602012,1986473,1471701,295443,7094862
1980,1428285,2230936,1801325,1168872,352121,7071439
1990,1487536,2300644,1951598,1203789,378977,7322564
2000,1537195,2465326,2229379,1332650,443728,8006278
2010,1494873,2504790,2230722,1385108,448730,81751123
2015,1644518,2636735,2339150,1455444,476558,8550405
```

nycHistPop.csv

In Lab 6

Week 6: structured data, pandas, & more design

```
import matplotlib.pyplot as plt
import pandas as pd
```

```
Source: https://en.wikipedia.org/wiki/Demographics_of_New_York_City,,,,,
All population figures are consistent with present-day boundaries,,,,,,
First census after the consolidation of the five boroughs,,,,,,
,,,,,
,,,,,
Year,Manhattan,Brooklyn,Queens,Bronx,Staten Island>Total
1698,4937,2017,,,727,7681
1771,21883,3623,,,2847,28423
1790,33131,45049,6159,1781,3827,49447
1800,60515,5740,6642,1755,4563,79215
1810,96373,40203,7444,2267,5347,119734
1820,123706,11187,8246,2782,6135,152056
1830,202589,20535,9049,3023,7082,242278
1840,312710,47613,14480,3344,10965,391114
1850,515547,138882,18593,8032,15061,696115
1860,813649,279122,32963,23593,25492,1174779
1870,942292,419801,45468,37393,33829,1470183
1880,1164673,599495,56559,51980,38991,1911698
1890,1441216,838547,87050,88908,51692,2507414
1900,1650093,1146582,152899,200507,67021,2437202
1910,2331542,1634351,284041,430980,85969,4766883
1920,2284103,2018256,469042,732016,116531,5620048
1930,1867312,2560461,1079129,1265258,159346,4590446
1940,1889924,2698295,1297634,1394711,174441,7454995
1950,1940101,2738075,1550849,1452177,191555,78991957
1960,1698281,2627319,1809578,1624815,221993,7781984
1970,1539233,2602012,1986473,1471701,295443,7894862
1980,1428285,2230936,1801325,1168872,352121,7071639
1990,1487536,2300644,1951598,1203789,378977,7322564
2000,1537195,2465326,2229379,1332650,443728,8008278
2010,1484873,2504700,2230722,1385108,448730,81751123
2015,1644518,2636735,2339150,1455444,476558,8550405
```

nycHistPop.csv

In Lab 6

Week 6: structured data, pandas, & more design

```
import matplotlib.pyplot as plt
import pandas as pd
```

```
pop = pd.read_csv('nycHistPop.csv',skiprows=5)
```

```
Source: https://en.wikipedia.org/wiki/Demographics_of_New_York_City,,,,,
All population figures are consistent with present-day boundaries,,,,,,
First census after the consolidation of the five boroughs,,,,,,
,,,,,
,,,,,
Year,Manhattan,Brooklyn,Queens,Bronx,Staten Island>Total
1698,4937,2017,,,727,7681
1771,21863,3623,,,2847,28423
1790,33131,45049,6159,1781,3827,49447
1800,40515,5740,6642,1755,4563,79215
1810,96373,40203,7444,2267,5347,119734
1820,123706,11187,8246,2782,6135,152056
1830,202589,20535,9049,3023,7082,242278
1840,312710,47613,14480,3344,10965,391114
1850,515547,138882,18593,8032,15061,696115
1860,813649,279122,32963,23593,25492,1174779
1870,942292,419921,45468,37393,33829,1470193
1880,1164673,599495,56559,51980,38991,1911690
1890,1441216,838547,87050,88908,51692,2507414
1900,1650093,1146582,152899,200507,67021,2437202
1910,2331542,1634351,284041,430980,85949,4766883
1920,2284123,2018256,469042,732016,116511,5620048
1930,1867312,2560461,1079129,1265258,159346,4590446
1940,1889924,2698285,1297634,1394711,174441,7454995
1950,1940101,2738075,1500849,1451277,191555,78991957
1960,1698281,2627319,1809578,1624815,221993,7781984
1970,1539233,2602012,1986473,1471701,295443,7894862
1980,1428285,2230936,1801325,1168872,352121,7071439
1990,1487536,2300644,1951598,1203789,378977,7322564
2000,1537195,2465326,2229379,1332650,443728,8008278
2010,1494873,2504700,2230722,1385108,448730,81751123
2015,1644518,2636735,2339150,1455444,476558,8550405
```

nycHistPop.csv

In Lab 6

Week 6: structured data, pandas, & more design

```
import matplotlib.pyplot as plt
import pandas as pd
```

```
pop = pd.read_csv('nycHistPop.csv', skiprows=5)
```

```
Source: https://en.wikipedia.org/wiki/Demographics_of_New_York_City,,,,,
All population figures are consistent with present-day boundaries,,,,,,
First census after the consolidation of the five boroughs,,,,,,
,,,,,
,,,,,
```

```
Year,Manhattan,Brooklyn,Queens,Bronx,Staten Island>Total
1698,4937,2017,,,727,7681
1771,21863,3623,,,2847,28423
1790,33131,4548,6159,1781,3827,49447
1800,40515,5740,6642,1755,4563,79215
1810,96373,8003,7444,2267,5347,119734
1820,123706,11187,8246,2782,6135,152056
1830,202589,20535,9049,3023,7082,242278
1840,312710,47613,14480,3344,10965,391114
1850,515547,138882,18593,8032,15061,696115
1860,813649,279122,32963,23593,25492,1174779
1870,942292,419921,45468,37393,33829,1470193
1880,1164673,599495,56559,51980,38991,1911698
1890,1441216,838547,87050,88908,51692,2507414
1900,1650093,1146582,152899,200507,67021,24372702
1910,2331542,1634351,284041,430980,85949,4766883
1920,2284103,2018256,469042,732016,116511,8420048
1930,1867312,2580461,1079129,1265258,158346,6506446
1940,1889924,2698285,1297634,1394711,174441,7454995
1950,1940101,2738275,1500849,1451277,291555,78991957
1960,1698281,2627319,1809578,1624815,221993,7781984
1970,1539233,2602012,1986473,1471701,295443,7894862
1980,1428285,2210936,1801325,1168872,352121,7071439
1990,1487536,2300644,1951598,1203789,378977,7322564
2000,1537195,2465326,2229379,1332650,443728,8008278
2010,1484873,2504790,2230722,1385108,448730,81751123
2015,1644518,2636735,2339150,1455444,476558,8550405
```

nycHistPop.csv

In Lab 6

Week 6: structured data, pandas, & more design

```
import matplotlib.pyplot as plt
import pandas as pd
```

```
pop = pd.read_csv('nycHistPop.csv', skiprows=5)
```

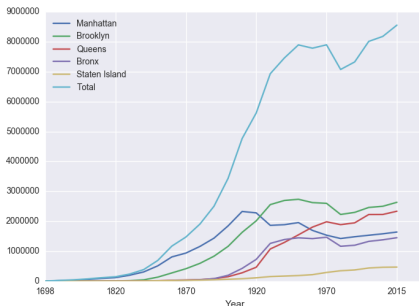
Source: https://en.wikipedia.org/wiki/Demographics_of_New_York_City,
All population figures are consistent with present-day boundaries.
First census after the consolidation of the five boroughs.

```
Year,Manhattan,Brooklyn,Queens,Bronx,Staten Island,Total
1698,4937,2017,,727,7681
1771,21863,3623,,2847,28423
1790,33131,45049,6159,1781,3827,49447
1800,40515,5740,6642,1755,4563,79215
1810,96373,8003,7444,2267,5347,119734
1820,123706,11187,8246,2782,6135,152056
1830,202589,20535,9049,3023,7082,242278
1840,312710,47613,14480,3344,10965,391114
1850,515547,138882,18593,8032,15061,696115
1860,813649,279122,32963,23593,25492,1174779
1870,942292,419801,45468,37393,33829,1470183
1880,1164673,599495,56559,51980,38991,1911698
1890,1441216,838547,87050,88908,51692,2507414
1900,1650093,1146582,152899,200507,67021,3437202
1910,2331542,1634351,284041,430989,85969,4766883
1920,2284103,2018256,469042,732016,116531,5620048
1930,1867312,2560451,1079129,1265598,159346,4690446
1940,1889924,2698285,1297634,1394711,174441,7454995
1950,1940101,2738275,1550469,1452177,291559,7892957
1960,1698281,2627319,1809578,1624815,221993,7781984
1970,1539233,2602012,1986473,1471701,295443,7094862
1980,1428285,2210936,1801325,1164872,352121,7071439
1990,1487536,2300644,1951598,1203789,378977,7322564
2000,1537195,2465326,2229379,1326550,443728,8006278
2010,1484873,2504760,2230722,1385108,468730,8175133
2015,1644518,2636735,2339155,1455444,476558,8550405
```

nycHistPop.csv

In Lab 6

```
pop.plot(x="Year")
plt.show()
```



Week 7: functions

- Functions are a way to break code into pieces, that can be easily reused.

```
#Name: your name here
#Date: October 2017
#This program, uses functions,
#    says hello to the world!

def main():
    print("Hello, World!")

if __name__ == "__main__":
    main()
```

Week 7: functions

```
#Name:  your name here  
#Date:  October 2017  
#This program, uses functions,  
#      says hello to the world!
```

```
def main():  
    print("Hello, World!")  
  
if __name__ == "__main__":  
    main()
```

- Functions are a way to break code into pieces, that can be easily reused.
- Many languages require that all code must be organized with functions.

Week 7: functions

```
#Name:  your name here
#Date:  October 2017
#This program, uses functions,
#      says hello to the world!

def main():
    print("Hello, World!")

if __name__ == "__main__":
    main()
```

- Functions are a way to break code into pieces, that can be easily reused.
- Many languages require that all code must be organized with functions.
- The opening function is often called `main()`

Week 7: functions

```
#Name: your name here
#Date: October 2017
#This program, uses functions,
#    says hello to the world!

def main():
    print("Hello, World!")

if __name__ == "__main__":
    main()
```

- Functions are a way to break code into pieces, that can be easily reused.
- Many languages require that all code must be organized with functions.
- The opening function is often called `main()`
- You **call** or **invoke** a function by typing its name, followed by any inputs, surrounded by parenthesis:

Week 7: functions

```
#Name: your name here
#Date: October 2017
#This program, uses functions,
#    says hello to the world!

def main():
    print("Hello, World!")

if __name__ == "__main__":
    main()
```

- Functions are a way to break code into pieces, that can be easily reused.
- Many languages require that all code must be organized with functions.
- The opening function is often called `main()`
- You **call** or **invoke** a function by typing its name, followed by any inputs, surrounded by parenthesis:
Example: `print("Hello", "World")`

Week 7: functions

```
#Name: your name here
#Date: October 2017
#This program, uses functions,
#    says hello to the world!

def main():
    print("Hello, World!")

if __name__ == "__main__":
    main()
```

- Functions are a way to break code into pieces, that can be easily reused.
- Many languages require that all code must be organized with functions.
- The opening function is often called `main()`
- You **call** or **invoke** a function by typing its name, followed by any inputs, surrounded by parenthesis: Example: `print("Hello", "World")`
- Can write, or **define** your own functions,

Week 7: functions

```
#Name: your name here
#Date: October 2017
#This program, uses functions,
#    says hello to the world!

def main():
    print("Hello, World!")

if __name__ == "__main__":
    main()
```

- Functions are a way to break code into pieces, that can be easily reused.
- Many languages require that all code must be organized with functions.
- The opening function is often called `main()`
- You **call** or **invoke** a function by typing its name, followed by any inputs, surrounded by parenthesis: Example: `print("Hello", "World")`
- Can write, or **define** your own functions, which are stored, until invoked or called.

Week 8: function parameters, github

- Functions can have **input parameters**.

```
def totalWithTax(food,tip):  
    total = 0  
    tax = 0.0875  
    total = food + food * tax  
    total = total + tip  
    return(total)  
  
lunch = float(input('Enter lunch total: '))  
lTip = float(input('Enter lunch tip: ' ))  
lTotal = totalWithTax(lunch, lTip)  
print('Lunch total is', lTotal)  
  
dinner= float(input('Enter dinner total: '))  
dTip = float(input('Enter dinner tip: ' ))  
dTotal = totalWithTax(dinner, dTip)  
print('Dinner total is', dTotal)
```

Week 8: function parameters, github

- Functions can have **input parameters**.
- Surrounded by parenthesis, both in the function definition, and in the function call (invocation).

```
def totalWithTax(food,tip):  
    total = 0  
    tax = 0.0875  
    total = food + food * tax  
    total = total + tip  
    return(total)  
  
lunch = float(input('Enter lunch total: '))  
lTip = float(input('Enter lunch tip: ' ))  
lTotal = totalWithTax(lunch, lTip)  
print('Lunch total is', lTotal)  
  
dinner= float(input('Enter dinner total: '))  
dTip = float(input('Enter dinner tip: ' ))  
dTotal = totalWithTax(dinner, dTip)  
print('Dinner total is', dTotal)
```

Week 8: function parameters, github

- Functions can have **input parameters**.
- Surrounded by parenthesis, both in the function definition, and in the function call (invocation).
- The “placeholders” in the function definition: **formal parameters**.

```
def totalWithTax(food,tip):  
    total = 0  
    tax = 0.0875  
    total = food + food * tax  
    total = total + tip  
    return(total)  
  
lunch = float(input('Enter lunch total: '))  
lTip = float(input('Enter lunch tip: ' ))  
lTotal = totalWithTax(lunch, lTip)  
print('Lunch total is', lTotal)  
  
dinner= float(input('Enter dinner total: '))  
dTip = float(input('Enter dinner tip: ' ))  
dTotal = totalWithTax(dinner, dTip)  
print('Dinner total is', dTotal)
```

Week 8: function parameters, github

```
def totalWithTax(food,tip):
    total = 0
    tax = 0.0875
    total = food + food * tax
    total = total + tip
    return(total)

lunch = float(input('Enter lunch total: '))
lTip = float(input('Enter lunch tip: '))
lTotal = totalWithTax(lunch, lTip)
print('Lunch total is', lTotal)

dinner= float(input('Enter dinner total: '))
dTip = float(input('Enter dinner tip: '))
dTotal = totalWithTax(dinner, dTip)
print('Dinner total is', dTotal)
```

- Functions can have **input parameters**.
- Surrounded by parenthesis, both in the function definition, and in the function call (invocation).
- The “placeholders” in the function definition: **formal parameters**.
- The ones in the function call: **actual parameters**

Week 8: function parameters, github

```
def totalWithTax(food,tip):
    total = 0
    tax = 0.0875
    total = food + food * tax
    total = total + tip
    return(total)

lunch = float(input('Enter lunch total: '))
lTip = float(input('Enter lunch tip: '))
lTotal = totalWithTax(lunch, lTip)
print('Lunch total is', lTotal)

dinner= float(input('Enter dinner total: '))
dTip = float(input('Enter dinner tip: '))
dTotal = totalWithTax(dinner, dTip)
print('Dinner total is', dTotal)
```

- Functions can have **input parameters**.
- Surrounded by parenthesis, both in the function definition, and in the function call (invocation).
- The “placeholders” in the function definition: **formal parameters**.
- The ones in the function call: **actual parameters**
- Functions can also **return values** to where it was called.

Week 8: function parameters, github

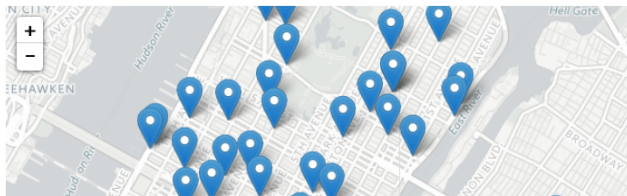
```
def totalWithTax(food,tip):  
    total = 0  
    tax = 0.0875  
    total = food + food * tax  
    total = total + tip  
    return(total)  
  
lunch = float(input('Enter lunch total: '))  
lTip = float(input('Enter lunch tip: '))  
lTotal = totalWithTax(lunch, lTip)  
print('Lunch total is', lTotal)  
  
dinner= float(input('Enter dinner total: '))  
dTip = float(input('Enter dinner tip: '))  
dTotal = totalWithTax(dinner, dTip)  
print('Dinner total is', dTotal)
```

Formal Parameters

Actual Parameters

- Functions can have **input parameters**.
- Surrounded by parenthesis, both in the function definition, and in the function call (invocation).
- The “placeholders” in the function definition: **formal parameters**.
- The ones in the function call: **actual parameters**.
- Functions can also **return values** to where it was called.

Week 9: top-down design, folium, loops, and random()



```
def main():  
    dataF = getData()  
    latColName, lonColName = getColumnNames()  
    lat, lon = getLocale()  
    cityMap = folium.Map(location = [lat,lon], tiles = 'cartodbpositron', zoom_start=11)  
    dotAllPoints(cityMap,dataF,latColName,lonColName)  
    markAndFindClosest(cityMap,dataF,latColName,lonColName,lat,lon)  
    writeMap(cityMap)
```

Week 10: more on loops, max design pattern, random()

```
dist = int(input('Enter distance: '))
while dist < 0:
    print('Distances cannot be negative.')
    dist = int(input('Enter distance: '))
print('The distance entered is', dist)
```

- Indefinite (while) loops allow you to repeat a block of code as long as a condition holds.

```
import turtle
import random

trey = turtle.Turtle()
trey.speed(10)

for i in range(100):
    trey.forward(10)
    a = random.randrange(0,360,90)
    trey.right(a)
```


Week 10: more on loops, max design pattern, random()

```
dist = int(input('Enter distance: '))
while dist < 0:
    print('Distances cannot be negative.')
    dist = int(input('Enter distance: '))
print('The distance entered is', dist)
```

- Indefinite (while) loops allow you to repeat a block of code as long as a condition holds.
- Very useful for checking user input for correctness.

```
import turtle
import random

trex = turtle.Turtle()
trex.speed(10)

for i in range(100):
    trex.forward(10)
    a = random.randrange(0,360,90)
    trex.right(a)
```

Week 10: more on loops, max design pattern, random()

```
dist = int(input('Enter distance: '))
while dist < 0:
    print('Distances cannot be negative.')
    dist = int(input('Enter distance: '))
print('The distance entered is', dist)
```

```
import turtle
import random

trex = turtle.Turtle()
trex.speed(10)

for i in range(100):
    trex.forward(10)
    a = random.randrange(0,360,90)
    trex.right(a)
```

- Indefinite (while) loops allow you to repeat a block of code as long as a condition holds.
- Very useful for checking user input for correctness.
- Python's built-in random package has useful methods for generating random whole numbers and real numbers.

Week 10: more on loops, max design pattern, random()

```
dist = int(input('Enter distance: '))
while dist < 0:
    print('Distances cannot be negative.')
    dist = int(input('Enter distance: '))
print('The distance entered is', dist)
```

```
import turtle
import random
```

```
trey = turtle.Turtle()
trey.speed(10)
```

```
for i in range(100):
    trey.forward(10)
    a = random.randrange(0,360,90)
    trey.right(a)
```

- Indefinite (while) loops allow you to repeat a block of code as long as a condition holds.
- Very useful for checking user input for correctness.
- Python's built-in random package has useful methods for generating random whole numbers and real numbers.
- To use, must include:
`import random.`

Week 10: more on loops, max design pattern, random()

```
dist = int(input('Enter distance: '))
while dist < 0:
    print('Distances cannot be negative.')
    dist = int(input('Enter distance: '))
print('The distance entered is', dist)
```

```
import turtle
import random
```

```
trey = turtle.Turtle()
trey.speed(10)
```

```
for i in range(100):
    trey.forward(10)
    a = random.randrange(0,360,90)
    trey.right(a)
```

- Indefinite (while) loops allow you to repeat a block of code as long as a condition holds.
- Very useful for checking user input for correctness.
- Python's built-in random package has useful methods for generating random whole numbers and real numbers.
- To use, must include:
`import random.`
- The max design pattern provides a template for finding maximum value from a list.

Python & Circuits Review: 10 Weeks in 10 Minutes



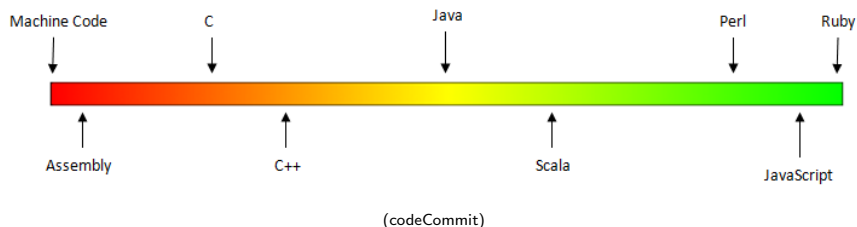
- Input/Output (I/O): `input()` and `print()`;
pandas for CSV files
- Types:
 - ▶ Primitive: `int`, `float`, `bool`, `string`;
 - ▶ Container: lists (but not dictionaries/hashes or tuples)
- Objects: turtles (used but did not design our own)
- Loops: definite & indefinite
- Conditionals: `if-elif-else`
- Logical Expressions & Circuits
- Functions: parameters & returns
- Packages:
 - ▶ Built-in: `turtle`, `math`, `random`
 - ▶ Popular: `numpy`, `matplotlib`, `pandas`, `folium`

Today's Topics



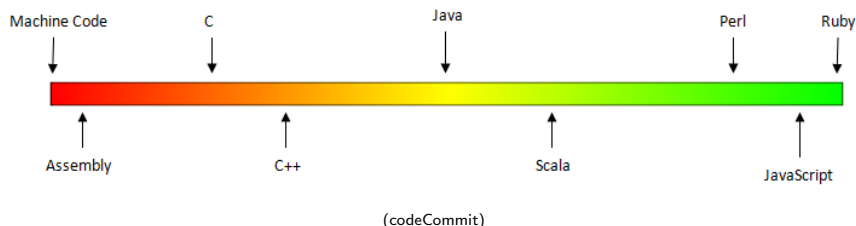
- Design Patterns: Searching
- Python Recap
- **Machine Language**
- Machine Language: Jumps & Loops
- Binary & Hex Arithmetic
- Final Exam: Format

Low-Level vs. High-Level Languages



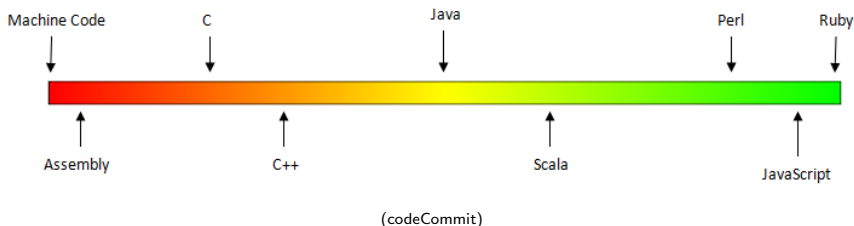
- Can view programming languages on a continuum.

Low-Level vs. High-Level Languages



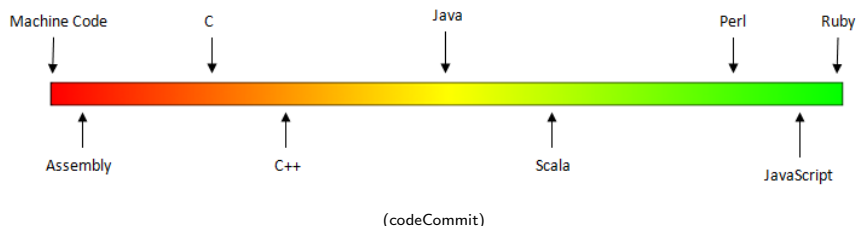
- Can view programming languages on a continuum.
- Those that directly access machine instructions & memory and have little abstraction are **low-level languages**

Low-Level vs. High-Level Languages



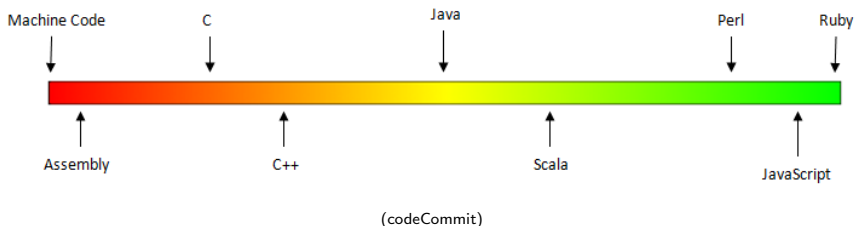
- Can view programming languages on a continuum.
- Those that directly access machine instructions & memory and have little abstraction are **low-level languages** (e.g. machine language, assembly language).

Low-Level vs. High-Level Languages



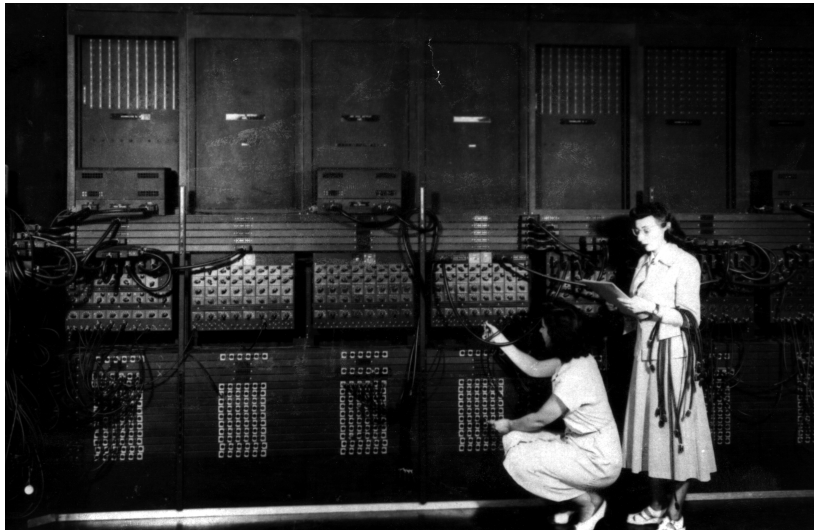
- Can view programming languages on a continuum.
- Those that directly access machine instructions & memory and have little abstraction are **low-level languages** (e.g. machine language, assembly language).
- Those that have strong abstraction (allow programming paradigms independent of the machine details, such as complex variables, functions and looping that do not translate directly into machine code) are called **high-level languages**.

Low-Level vs. High-Level Languages



- Can view programming languages on a continuum.
- Those that directly access machine instructions & memory and have little abstraction are **low-level languages** (e.g. machine language, assembly language).
- Those that have strong abstraction (allow programming paradigms independent of the machine details, such as complex variables, functions and looping that do not translate directly into machine code) are called **high-level languages**.
- Some languages, like C, are in between– allowing both low level access and high level data structures.

Machine Language



(Ruth Gordon & Ester Gerston programming the ENIAC, UPenn)

Machine Language

```
1 FOX 12:01a 23- 1
A 002000 C2 30 REP #$30
A 002002 18 CLC
A 002003 F8 SED
A 002004 A9 34 12 LDA #$1234
A 002007 69 21 43 ADC #$4321
A 00200A 8F 03 7F 01 STA $017F03
A 00200E D8 CLD
A 00200F E2 30 SEP #$30
A 002011 00 BRK
A 2012

r
PB PC NUmxDI2C .A .X .Y SP DP DB
; 00 E012 00110000 0000 0000 0002 CFFF 0000 00
g 2000

BREAK

PB PC NUmxDI2C .A .X .Y SP DP DB
; 00 2013 00110000 5555 0000 0002 CFFF 0000 00
m 7f03 7f03
>007F03 55 55 00 00 00 00 00 00 00 00 00 00 00 00 00:UU.....
█
```

(wiki)

Machine Language

- We will be writing programs in a simplified machine language, WeMIPS.

[illegible]

(wiki)

Machine Language

```
002000 c2 30 REP #30
002002 10 CLC
002003 F0 SED
002004 40 34 12 LSH #1224
002007 60 21 43 RSC #4321
00200A 0F 03 7F 01 STA #017F03
00200C 00 CLJ
00200F E2 30 SEP #30
002011 00 BRX
002012

P PC Mem32C A X Y SP BP BB
: 00 2012 00110000 0000 0000 0002 C7FF 0000 00
$ 2000
BREAK
P PC Mem32C A X Y SP BP BB
: 00 2013 00110000 5555 0000 0002 C7FF 0000 00
n 1103 7403
007F03 55 55 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

(wiki)

- We will be writing programs in a simplified machine language, WeMIPS.
- It is based on a reduced instruction set computer (RISC) design, originally developed by the MIPS Computer Systems.

Machine Language



(wiki)

- We will be writing programs in a simplified machine language, WeMIPS.
- It is based on a reduced instruction set computer (RISC) design, originally developed by the MIPS Computer Systems.
- Due to its small set of commands, processors can be designed to run those commands very efficiently.

Machine Language



(wiki)

- We will be writing programs in a simplified machine language, WeMIPS.
- It is based on a reduced instruction set computer (RISC) design, originally developed by the MIPS Computer Systems.
- Due to its small set of commands, processors can be designed to run those commands very efficiently.
- More in future architecture classes....

"Hello World!" in Simplified Machine Language

Line: 3 Go!

Show/Hide Demos

[User Guide](#) | [Unit Tests](#) | [Docs](#)

Addition Doubler

Stav

Looper

Stack Test

Hello World

Code Gen Save String

Interactive

Binary2 Decimal

Decimal2 Binary

Debug

```
1 # Store 'Hello world!' at the top of the stack
2 ADDI $sp, $sp, -13
3 ADDI $t0, $zero, 72 # H
4 SB $t0, 0($sp)
5 ADDI $t0, $zero, 101 # e
6 SB $t0, 1($sp)
7 ADDI $t0, $zero, 108 # l
8 SB $t0, 2($sp)
9 ADDI $t0, $zero, 108 # l
10 SB $t0, 3($sp)
11 ADDI $t0, $zero, 111 # o
12 SB $t0, 4($sp)
13 ADDI $t0, $zero, 32 # (space)
14 SB $t0, 5($sp)
15 ADDI $t0, $zero, 119 # w
16 SB $t0, 6($sp)
17 ADDI $t0, $zero, 111 # o
18 SB $t0, 7($sp)
19 ADDI $t0, $zero, 114 # r
20 SB $t0, 8($sp)
21 ADDI $t0, $zero, 108 # l
22 SB $t0, 9($sp)
23 ADDI $t0, $zero, 100 # d
24 SB $t0, 10($sp)
25 ADDI $t0, $zero, 33 # !
26 SB $t0, 11($sp)
27 ADDI $t0, $zero, 0 # (null)
28 SB $t0, 12($sp)
29
30 ADDI $v0, $zero, 4 # 4 is for print string
31 ADDI $a0, $sp, 0
32 syscall # print to the log
```

Step Run ☒ Enable auto switching

S T A V Stack Log

s0:	10
s1:	9
s2:	9
s3:	22
s4:	696
s5:	976
s6:	927
s7:	418

(WeMIPS)

WeMIPS

Line: 3 dis

Show/Hide Demos

User Guide | Unit Tests | Docs

Addition Doubler Stop Looper Stack Test Hello World

Code Gen Save String Interactive Binary2 Decimal Decimal2 Binary

Debug

```
1 # Store "hello world!" at the top of the stack
2 ADDUI $a0, $zero, 32 # 8
3 ROR $t0, $t0, 1
4 ADDUI $t0, $zero, 101 # e
5 DD $t0, 10($sp)
6 ADDUI $t0, $zero, 108 # l
7 DD $t0, 6($sp)
8 ADDUI $t0, $zero, 108 # l
9 DD $t0, 6($sp)
10 ADDUI $t0, $zero, 111 # o
11 DD $t0, 6($sp)
12 ADDUI $t0, $zero, 32 # (space)
13 DD $t0, 6($sp)
14 ADDUI $t0, $zero, 119 # w
15 DD $t0, 6($sp)
16 ADDUI $t0, $zero, 114 # u
17 DD $t0, 6($sp)
18 ADDUI $t0, $zero, 108 # d
19 DD $t0, 6($sp)
20 ADDUI $t0, $zero, 108 # l
21 DD $t0, 6($sp)
22 DD $t0, 6($sp)
23 ADDUI $t0, $zero, 103 # d
24 DD $t0, 10($sp)
25 ADDUI $t0, $zero, 33 # !
26 DD $t0, 10($sp)
27 ADDUI $t0, $zero, 0 # (null)
28 DD $t0, 10($sp)
29
30 ADDUI $v0, $zero, 6 # 4 in for print string
31 ADDUI $a0, $a0, 0
32 syscall # print to the log
```

Step	Run	Enable auto-switching			
S	T	A	V	Stack	Log
a0:				10	
t0:				9	
a0:				9	
a0:				22	
a0:				695	
a0:				970	
a0:				927	
a0:				418	

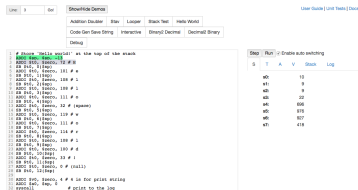
(Demo with WeMIPS)

MIPS Commands

```
1 # Store "hello world!" at the top of the stack
2 ADDI $0, $zero, 12 # 12
3 SW $0, 0($0)
4 ADDI $0, $zero, 191 # 1
5 SW $0, 100($0)
6 ADDI $0, $zero, 100 # 1
7 SW $0, 200($0)
8 ADDI $0, $zero, 131 # 1
9 SW $0, 300($0)
10 ADDI $0, $zero, 12 # (space)
11 SW $0, 400($0)
12 ADDI $0, $zero, 131 # 1
13 SW $0, 500($0)
14 ADDI $0, $zero, 114 # 1
15 SW $0, 600($0)
16 ADDI $0, $zero, 100 # 1
17 SW $0, 700($0)
18 ADDI $0, $zero, 100 # 1
19 SW $0, 800($0)
20 ADDI $0, $zero, 10 # (newline)
21 SW $0, 900($0)
22 ADDI $0, $zero, 4 # 4 is for print string
23 ADDI $0, $zero, 0 # print to the log
24 syscall
```

- **Registers:** locations for storing information that can be quickly accessed.

MIPS Commands



```
1 # Store "hello world" at the top of the stack
2 ADDI $0, $zero, 128
3 SB $0, 0($0)
4 ADDI $0, $zero, 191 # n
5 SB $0, 1($0)
6 ADDI $0, $zero, 109 # i
7 SB $0, 2($0)
8 ADDI $0, $zero, 131 # n
9 SB $0, 3($0)
10 ADDI $0, $zero, 32 # (space)
11 SB $0, 4($0)
12 ADDI $0, $zero, 111 # n
13 SB $0, 5($0)
14 ADDI $0, $zero, 114 # n
15 SB $0, 6($0)
16 ADDI $0, $zero, 109 # i
17 SB $0, 7($0)
18 ADDI $0, $zero, 100 # d
19 SB $0, 8($0)
20 ADDI $0, $zero, 33 # !
21 SB $0, 9($0)
22 ADDI $0, $zero, 0 # (null)
23 SB $0, 10($0)
24 ADDI $0, $zero, 4 # n is for print ending
25 ADDI $0, $zero, 0 # print to the log
26 syscall
```

\$	T	A	V	Stack	Log
\$0				10	
\$1				9	
\$2				8	
\$3				22	
\$4				856	
\$5				876	
\$6				887	
\$7				418	

- **Registers:** locations for storing information that can be quickly accessed. Names start with '\$': \$s0, \$s1, \$t0, \$t1,...

MIPS Commands

[illegible]

- **Registers:** locations for storing information that can be quickly accessed. Names start with '\$': \$s0, \$s1, \$t0, \$t1,...
- **R Instructions:** Commands that use data in the registers:

MIPS Commands

[illegible]

- **Registers:** locations for storing information that can be quickly accessed. Names start with '\$': \$s0, \$s1, \$t0, \$t1,...
- **R Instructions:** Commands that use data in the registers:
add \$s1, \$s2, \$s3

MIPS Commands

[illegible]

- **Registers:** locations for storing information that can be quickly accessed. Names start with '\$': \$s0, \$s1, \$t0, \$t1,...
- **R Instructions:** Commands that use data in the registers:
add \$s1, \$s2, \$s3 (Basic form: OP rd, rs, rt)
- **I Instructions:** instructions that also use intermediate values.

MIPS Commands

[illegible]

- **Registers:** locations for storing information that can be quickly accessed. Names start with '\$': \$s0, \$s1, \$t0, \$t1,...
- **R Instructions:** Commands that use data in the registers:
add \$s1, \$s2, \$s3 (Basic form: OP rd, rs, rt)
- **I Instructions:** instructions that also use intermediate values.
addi \$s1, \$s2, 100

MIPS Commands

```

# Store "Hello world" on the top of the stack
A000 100, 1000, 12 #R#
R0 100, 1000
A000 100, 1000, 131 #R#
R0 100, 1000
A000 100, 1000, 108 # 1
R0 100, 1000
A000 100, 1000, 109 # 1
R0 100, 1000
A000 100, 1000, 110 # 1
R0 100, 1000
A000 100, 1000, 12 # R (again)
R0 100, 1000
A000 100, 1000, 129 # #
R0 100, 1000
A000 100, 1000, 111 # #
R0 100, 1000
A000 100, 1000, 114 # #
R0 100, 1000
A000 100, 1000, 109 # 1
R0 100, 1000
A000 100, 1000, 109 # 1
R0 100, 1000
A000 100, 1000, 12 # 1
R0 100, 1000
A000 100, 1000, 0 # (null)
R0 100, 121[reg]
A000 100, 1000, 0 # 0 is for print string
A000 100, 1000, 0
STOP
# print to the log

```

- **Registers:** locations for storing information that can be quickly accessed. Names start with '\$': \$s0, \$s1, \$t0, \$t1,...
- **R Instructions:** Commands that use data in the registers:
add \$s1, \$s2, \$s3 (Basic form: OP rd, rs, rt)
- **I Instructions:** instructions that also use intermediate values.
addi \$s1, \$s2, 100 (Basic form: OP rd, rs, imm)
- **J Instructions:** instructions that jump to another memory location.

MIPS Commands

[illegible]

- **Registers:** locations for storing information that can be quickly accessed. Names start with '\$': \$s0, \$s1, \$t0, \$t1,...
- **R Instructions:** Commands that use data in the registers:
add \$s1, \$s2, \$s3 (Basic form: OP rd, rs, rt)
- **I Instructions:** instructions that also use intermediate values.
addi \$s1, \$s2, 100 (Basic form: OP rd, rs, imm)
- **J Instructions:** instructions that jump to another memory location.
j done

MIPS Commands

```

# Store "Hello world" on the top of the stack
A000 100, 1000, 12 #R#
R0 100, 1000
A000 100, 1000, 131 #R#
R0 100, 1000
A000 100, 1000, 108 # 1
R0 100, 1000
A000 100, 1000, 109 # 1
R0 100, 1000
A000 100, 1000, 110 # 1
R0 100, 1000
A000 100, 1000, 12 # R (again)
R0 100, 1000
A000 100, 1000, 129 # #
R0 100, 1000
A000 100, 1000, 111 # #
R0 100, 1000
A000 100, 1000, 114 # #
R0 100, 1000
A000 100, 1000, 109 # 1
R0 100, 1000
A000 100, 1000, 109 # 1
R0 100, 1000
A000 100, 1000, 12 # 1
R0 100, 1000
A000 100, 1000, 0 # (null)
R0 100, 121[reg]
A000 100, 1000, 0 # 0 is for print string
A000 100, 1000, 0
STOP
# print to the log

```

- **Registers:** locations for storing information that can be quickly accessed. Names start with '\$': \$s0, \$s1, \$t0, \$t1,...
- **R Instructions:** Commands that use data in the registers:
add \$s1, \$s2, \$s3 (Basic form: OP rd, rs, rt)
- **I Instructions:** instructions that also use intermediate values.
addi \$s1, \$s2, 100 (Basic form: OP rd, rs, imm)
- **J Instructions:** instructions that jump to another memory location.
j done (Basic form: OP label)

In Pairs or Triples:

Line: 3 Go!

Show/Hide Demos

[User Guide](#) | [Unit Tests](#) | [Docs](#)

Addition Doubler

Stav

Looper

Stack Test

Hello World

Code Gen Save String

Interactive

Binary2 Decimal

Decimal2 Binary

Debug

```
1 # Store 'Hello world!' at the top of the stack
2 ADDI $sp, $sp, -13
3 ADDI $t0, $zero, 72 # H
4 SB $t0, 0($sp)
5 ADDI $t0, $zero, 101 # e
6 SB $t0, 1($sp)
7 ADDI $t0, $zero, 108 # l
8 SB $t0, 2($sp)
9 ADDI $t0, $zero, 108 # l
10 SB $t0, 3($sp)
11 ADDI $t0, $zero, 111 # o
12 SB $t0, 4($sp)
13 ADDI $t0, $zero, 32 # (space)
14 SB $t0, 5($sp)
15 ADDI $t0, $zero, 119 # w
16 SB $t0, 6($sp)
17 ADDI $t0, $zero, 111 # o
18 SB $t0, 7($sp)
19 ADDI $t0, $zero, 114 # r
20 SB $t0, 8($sp)
21 ADDI $t0, $zero, 108 # l
22 SB $t0, 9($sp)
23 ADDI $t0, $zero, 100 # d
24 SB $t0, 10($sp)
25 ADDI $t0, $zero, 33 # !
26 SB $t0, 11($sp)
27 ADDI $t0, $zero, 0 # (null)
28 SB $t0, 12($sp)
29
30 ADDI $v0, $zero, 4 # 4 is for print string
31 ADDI $a0, $sp, 0
32 syscall # print to the log
```

Step Run ☒ Enable auto switching

S	T	A	V	Stack	Log
				s0:	10
				s1:	9
				s2:	9
				s3:	22
				s4:	696
				s5:	976
				s6:	927
				s7:	418

Write a program that prints out the alphabet: a b c d ... x y z

WeMIPS

Line: 3 dis

Show/Hide Demos

User Guide | Unit Tests | Docs

Addition Doubler Stop Looper Stack Test Hello World

Code Gen Save String Interactive Binary2 Decimal Decimal2 Binary

Debug

```
1 # Store "hello world!" at the top of the stack
2 ADDUI $a0, $zero, 32 # 0
3 LR $t0, 0($a0)
4 ADDUI $t0, $zero, 101 # e
5 SD $t0, 0($a0)
6 SD $t0, 4($a0)
7 ADDUI $t0, $zero, 108 # l
8 SD $t0, 8($a0)
9 ADDUI $t0, $zero, 108 # l
10 SD $t0, 12($a0)
11 ADDUI $t0, $zero, 111 # o
12 SD $t0, 16($a0)
13 ADDUI $t0, $zero, 32 # (space)
14 SD $t0, 20($a0)
15 ADDUI $t0, $zero, 119 # w
16 SD $t0, 24($a0)
17 ADDUI $t0, $zero, 114 # u
18 SD $t0, 28($a0)
19 ADDUI $t0, $zero, 114 # u
20 SD $t0, 32($a0)
21 ADDUI $t0, $zero, 108 # l
22 SD $t0, 36($a0)
23 ADDUI $t0, $zero, 108 # l
24 SD $t0, 40($a0)
25 ADDUI $t0, $zero, 33 # !
26 SD $t0, 44($a0)
27 ADDUI $t0, $zero, 0 # (null)
28 SD $t0, 48($a0)
29
30 ADDUI $v0, $zero, 6 # 4 in for print string
31 ADDUI $a0, $zero, 0
32 syscall # print to the log
```

Step Run v Enable auto-switching

S	T	A	V	Stack	Log
				a0:	10
				t0:	9
				a0:	9
				a0:	22
				a0:	695
				a0:	970
				a0:	927
				a0:	418

(Demo with WeMIPS)

Today's Topics



- Design Patterns: Searching
- Python Recap
- Machine Language
- **Machine Language: Jumps & Loops**
- Binary & Hex Arithmetic
- Final Exam: Format

Loops & Jumps in Machine Language

- Instead of built-in looping structures like `for` and `while`, you create your own loops by “jumping” to the location in the program.



The screenshot shows a debugger window with two panes. The left pane displays assembly code with instructions like `movl $0, %eax`, `movl $1, %ecx`, and `jmp $0x00000000`. The right pane shows the state of registers, including `%eax`, `%ecx`, and `%edx`.

Loops & Jumps in Machine Language

- Instead of built-in looping structures like `for` and `while`, you create your own loops by “jumping” to the location in the program.
- Can indicate locations by writing **labels** at the beginning of a line.
- Then give a command to jump to that location.



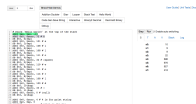
Loops & Jumps in Machine Language

- Instead of built-in looping structures like `for` and `while`, you create your own loops by “jumping” to the location in the program.
- Can indicate locations by writing **labels** at the beginning of a line.
- Then give a command to jump to that location.
- Different kinds of jumps:



Loops & Jumps in Machine Language

- Instead of built-in looping structures like `for` and `while`, you create your own loops by “jumping” to the location in the program.
- Can indicate locations by writing **labels** at the beginning of a line.
- Then give a command to jump to that location.
- Different kinds of jumps:
 - ▶ **Unconditional:** `j Done` will jump to the address with label `Done`.



Loops & Jumps in Machine Language

- Instead of built-in looping structures like `for` and `while`, you create your own loops by “jumping” to the location in the program.
- Can indicate locations by writing **labels** at the beginning of a line.
- Then give a command to jump to that location.
- Different kinds of jumps:
 - ▶ **Unconditional:** `j Done` will jump to the address with label `Done`.
 - ▶ **Branch if Equal:** `beq $s0 $s1 DoAgain` will jump to the address with label `DoAgain` if the registers `$s0` and `$s1` contain the same value.



Loops & Jumps in Machine Language

- Instead of built-in looping structures like `for` and `while`, you create your own loops by “jumping” to the location in the program.
- Can indicate locations by writing **labels** at the beginning of a line.
- Then give a command to jump to that location.
- Different kinds of jumps:
 - ▶ **Unconditional:** `j Done` will jump to the address with label `Done`.
 - ▶ **Branch if Equal:** `beq $s0 $s1 DoAgain` will jump to the address with label `DoAgain` if the registers `$s0` and `$s1` contain the same value.
 - ▶ See reading for more variations.



Jump Demo

Line: 3 dis

Show/Hide Demos

User Guide | Unit Tests | Docs

Addition Doubler Stop Looper Stack Test Hello World

Code Gen Save String Interactive Binary2 Decimal Decimal2 Binary

Debug

```
1 # Store "hello world!" at the top of the stack
2 ADDI $a0, $zero, 32 # 0
3 R0 $a0, 0($a0)
4 ADDI $t0, $zero, 101 # e
5 R0 $t0, 0($a0)
6 R0 $t0, 0($a0)
7 ADDI $t0, $zero, 108 # l
8 R0 $t0, 0($a0)
9 ADDI $t0, $zero, 109 # l
10 R0 $t0, 0($a0)
11 ADDI $t0, $zero, 111 # o
12 R0 $t0, 0($a0)
13 ADDI $t0, $zero, 32 # (space)
14 R0 $t0, 0($a0)
15 ADDI $t0, $zero, 119 # w
16 R0 $t0, 0($a0)
17 ADDI $t0, $zero, 114 # u
18 R0 $t0, 0($a0)
19 ADDI $t0, $zero, 114 # u
20 R0 $t0, 0($a0)
21 ADDI $t0, $zero, 108 # l
22 R0 $t0, 0($a0)
23 ADDI $t0, $zero, 103 # d
24 R0 $t0, 0($a0)
25 ADDI $t0, $zero, 33 # !
26 R0 $t0, 0($a0)
27 ADDI $t0, $zero, 0 # (null)
28 R0 $t0, 0($a0)
29
30 ADDI $v0, $zero, 6 # 4 in for print string
31 ADDI $a0, $a0, 0
32 syscall # print to the log
```

Step Run ☐ Enable auto-switching

S	T	A	V	Stack	Log
				a0:	10
				t0:	9
				a0:	9
				a0:	22
				a0:	695
				a0:	970
				a0:	927
				a0:	418

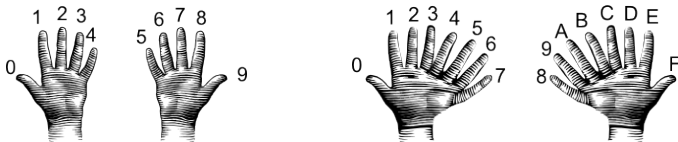
(Demo with WeMIPS)

Today's Topics



- Design Patterns: Searching
- Python Recap
- Machine Language
- Machine Language: Jumps & Loops
- **Binary & Hex Arithmetic**
- Final Exam: Format

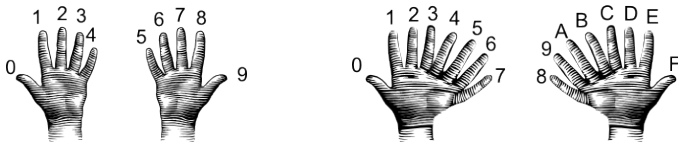
Hexadecimal to Decimal: Converting Between Bases



(from i-programmer.info)

- From hexadecimal to decimal:
 - ▶ Convert first digit to decimal and multiple by 16.

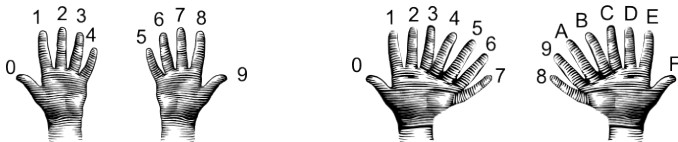
Hexadecimal to Decimal: Converting Between Bases



(from i-programmer.info)

- From hexadecimal to decimal:
 - ▶ Convert first digit to decimal and multiple by 16.
 - ▶ Convert second digit to decimal and add to total.

Hexadecimal to Decimal: Converting Between Bases

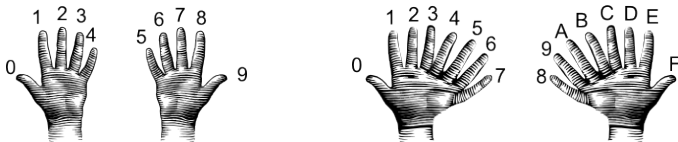


(from i-programmer.info)

- From hexadecimal to decimal:

- ▶ Convert first digit to decimal and multiple by 16.
- ▶ Convert second digit to decimal and add to total.
- ▶ Example: what is 2A as a decimal number?

Hexadecimal to Decimal: Converting Between Bases

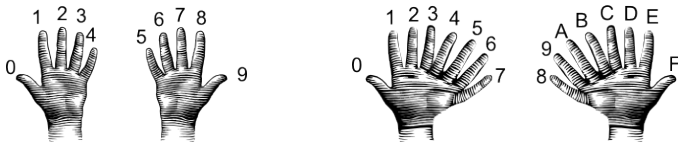


(from i-programmer.info)

- From hexadecimal to decimal:

- ▶ Convert first digit to decimal and multiple by 16.
- ▶ Convert second digit to decimal and add to total.
- ▶ Example: what is 2A as a decimal number?
2 in decimal is 2.

Hexadecimal to Decimal: Converting Between Bases

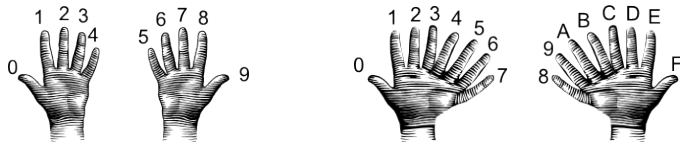


(from i-programmer.info)

- From hexadecimal to decimal:

- ▶ Convert first digit to decimal and multiple by 16.
- ▶ Convert second digit to decimal and add to total.
- ▶ Example: what is 2A as a decimal number?
2 in decimal is 2. 2×16 is 32.

Hexadecimal to Decimal: Converting Between Bases

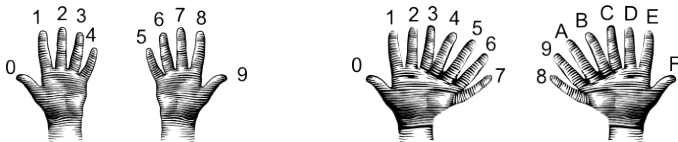


(from i-programmer.info)

- From hexadecimal to decimal:

- ▶ Convert first digit to decimal and multiple by 16.
- ▶ Convert second digit to decimal and add to total.
- ▶ Example: what is 2A as a decimal number?
 - 2 in decimal is 2. 2×16 is 32.
 - A in decimal digits is 10.

Hexadecimal to Decimal: Converting Between Bases

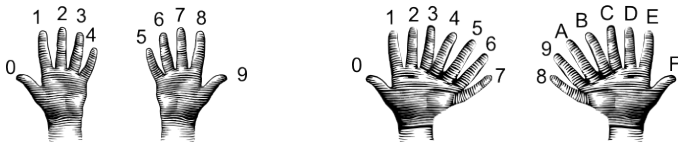


(from i-programmer.info)

- From hexadecimal to decimal:

- ▶ Convert first digit to decimal and multiple by 16.
- ▶ Convert second digit to decimal and add to total.
- ▶ Example: what is 2A as a decimal number?
 - 2 in decimal is 2. 2×16 is 32.
 - A in decimal digits is 10.
 - $32 + 10$ is 42.

Hexadecimal to Decimal: Converting Between Bases

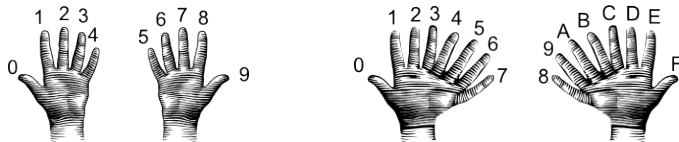


(from i-programmer.info)

- From hexadecimal to decimal:

- ▶ Convert first digit to decimal and multiple by 16.
- ▶ Convert second digit to decimal and add to total.
- ▶ Example: what is 2A as a decimal number?
2 in decimal is 2. 2×16 is 32.
A in decimal digits is 10.
 $32 + 10$ is 42.
Answer is 42.
- ▶ Example: what is 99 as a decimal number?

Hexadecimal to Decimal: Converting Between Bases

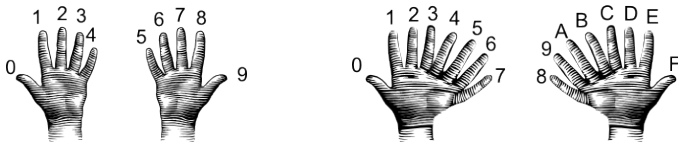


(from i-programmer.info)

- From hexadecimal to decimal:

- ▶ Convert first digit to decimal and multiple by 16.
- ▶ Convert second digit to decimal and add to total.
- ▶ Example: what is 2A as a decimal number?
2 in decimal is 2. 2×16 is 32.
A in decimal digits is 10.
 $32 + 10$ is 42.
Answer is 42.
- ▶ Example: what is 99 as a decimal number?
9 in decimal is 9.

Hexadecimal to Decimal: Converting Between Bases

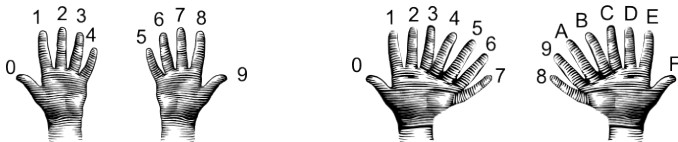


(from i-programmer.info)

- From hexadecimal to decimal:

- ▶ Convert first digit to decimal and multiple by 16.
- ▶ Convert second digit to decimal and add to total.
- ▶ Example: what is 2A as a decimal number?
 - 2 in decimal is 2. 2×16 is 32.
 - A in decimal digits is 10.
 - $32 + 10$ is 42.
 - Answer is 42.
- ▶ Example: what is 99 as a decimal number?
 - 9 in decimal is 9. 9×16 is 144.

Hexadecimal to Decimal: Converting Between Bases

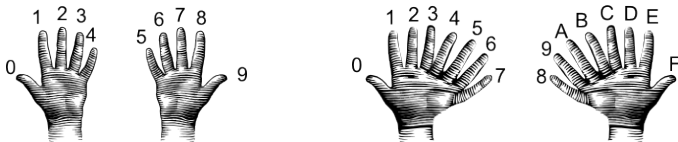


(from i-programmer.info)

- From hexadecimal to decimal:

- ▶ Convert first digit to decimal and multiple by 16.
- ▶ Convert second digit to decimal and add to total.
- ▶ Example: what is 2A as a decimal number?
 - 2 in decimal is 2. 2×16 is 32.
 - A in decimal digits is 10.
 - $32 + 10$ is 42.
 - Answer is 42.
- ▶ Example: what is 99 as a decimal number?
 - 9 in decimal is 9. 9×16 is 144.
 - 9 in decimal digits is 9

Hexadecimal to Decimal: Converting Between Bases

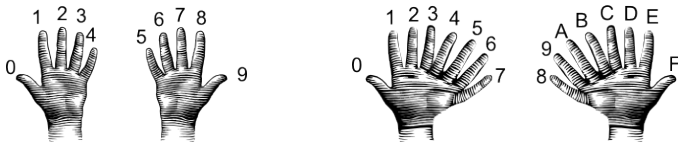


(from i-programmer.info)

- From hexadecimal to decimal:

- ▶ Convert first digit to decimal and multiple by 16.
- ▶ Convert second digit to decimal and add to total.
- ▶ Example: what is 2A as a decimal number?
2 in decimal is 2. 2×16 is 32.
A in decimal digits is 10.
 $32 + 10$ is 42.
Answer is 42.
- ▶ Example: what is 99 as a decimal number?
9 in decimal is 9. 9×16 is 144.
9 in decimal digits is 9
 $144 + 9$ is 153.

Hexadecimal to Decimal: Converting Between Bases



(from i-programmer.info)

- From hexadecimal to decimal:

- ▶ Convert first digit to decimal and multiple by 16.
- ▶ Convert second digit to decimal and add to total.
- ▶ Example: what is 2A as a decimal number?

2 in decimal is 2. 2×16 is 32.

A in decimal digits is 10.

$32 + 10$ is 42.

Answer is 42.

- ▶ Example: what is 99 as a decimal number?

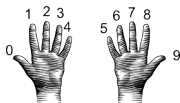
9 in decimal is 9. 9×16 is 144.

9 in decimal digits is 9

$144 + 9$ is 153.

Answer is 153.

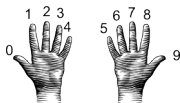
Decimal to Binary: Converting Between Bases



- From decimal to binary:

- ▶ Divide by 128 ($= 2^7$). Quotient is the first digit.

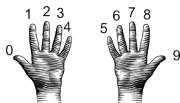
Decimal to Binary: Converting Between Bases



- From decimal to binary:

- ▶ Divide by 128 ($= 2^7$). Quotient is the first digit.
- ▶ Divide remainder by 64 ($= 2^6$). Quotient is the next digit.

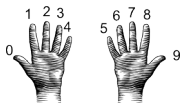
Decimal to Binary: Converting Between Bases



- From decimal to binary:

- ▶ Divide by 128 ($= 2^7$). Quotient is the first digit.
- ▶ Divide remainder by 64 ($= 2^6$). Quotient is the next digit.
- ▶ Divide remainder by 32 ($= 2^5$). Quotient is the next digit.

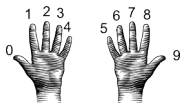
Decimal to Binary: Converting Between Bases



- From decimal to binary:

- ▶ Divide by 128 ($= 2^7$). Quotient is the first digit.
- ▶ Divide remainder by 64 ($= 2^6$). Quotient is the next digit.
- ▶ Divide remainder by 32 ($= 2^5$). Quotient is the next digit.
- ▶ Divide remainder by 16 ($= 2^4$). Quotient is the next digit.

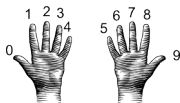
Decimal to Binary: Converting Between Bases



- From decimal to binary:

- ▶ Divide by 128 ($= 2^7$). Quotient is the first digit.
- ▶ Divide remainder by 64 ($= 2^6$). Quotient is the next digit.
- ▶ Divide remainder by 32 ($= 2^5$). Quotient is the next digit.
- ▶ Divide remainder by 16 ($= 2^4$). Quotient is the next digit.
- ▶ Divide remainder by 8 ($= 2^3$). Quotient is the next digit.

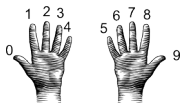
Decimal to Binary: Converting Between Bases



● From decimal to binary:

- ▶ Divide by 128 ($= 2^7$). Quotient is the first digit.
- ▶ Divide remainder by 64 ($= 2^6$). Quotient is the next digit.
- ▶ Divide remainder by 32 ($= 2^5$). Quotient is the next digit.
- ▶ Divide remainder by 16 ($= 2^4$). Quotient is the next digit.
- ▶ Divide remainder by 8 ($= 2^3$). Quotient is the next digit.
- ▶ Divide remainder by 4 ($= 2^2$). Quotient is the next digit.

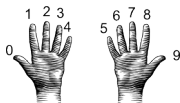
Decimal to Binary: Converting Between Bases



● From decimal to binary:

- ▶ Divide by 128 ($= 2^7$). Quotient is the first digit.
- ▶ Divide remainder by 64 ($= 2^6$). Quotient is the next digit.
- ▶ Divide remainder by 32 ($= 2^5$). Quotient is the next digit.
- ▶ Divide remainder by 16 ($= 2^4$). Quotient is the next digit.
- ▶ Divide remainder by 8 ($= 2^3$). Quotient is the next digit.
- ▶ Divide remainder by 4 ($= 2^2$). Quotient is the next digit.
- ▶ Divide remainder by 2 ($= 2^1$). Quotient is the next digit.

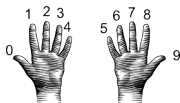
Decimal to Binary: Converting Between Bases



● From decimal to binary:

- ▶ Divide by 128 ($= 2^7$). Quotient is the first digit.
- ▶ Divide remainder by 64 ($= 2^6$). Quotient is the next digit.
- ▶ Divide remainder by 32 ($= 2^5$). Quotient is the next digit.
- ▶ Divide remainder by 16 ($= 2^4$). Quotient is the next digit.
- ▶ Divide remainder by 8 ($= 2^3$). Quotient is the next digit.
- ▶ Divide remainder by 4 ($= 2^2$). Quotient is the next digit.
- ▶ Divide remainder by 2 ($= 2^1$). Quotient is the next digit.
- ▶ The last remainder is the last digit.

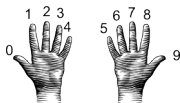
Decimal to Binary: Converting Between Bases



● From decimal to binary:

- ▶ Divide by 128 ($= 2^7$). Quotient is the first digit.
- ▶ Divide remainder by 64 ($= 2^6$). Quotient is the next digit.
- ▶ Divide remainder by 32 ($= 2^5$). Quotient is the next digit.
- ▶ Divide remainder by 16 ($= 2^4$). Quotient is the next digit.
- ▶ Divide remainder by 8 ($= 2^3$). Quotient is the next digit.
- ▶ Divide remainder by 4 ($= 2^2$). Quotient is the next digit.
- ▶ Divide remainder by 2 ($= 2^1$). Quotient is the next digit.
- ▶ The last remainder is the last digit.
- ▶ Example: what is 130 in binary notation?

Decimal to Binary: Converting Between Bases

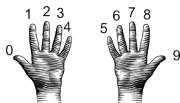


● From decimal to binary:

- ▶ Divide by 128 ($= 2^7$). Quotient is the first digit.
- ▶ Divide remainder by 64 ($= 2^6$). Quotient is the next digit.
- ▶ Divide remainder by 32 ($= 2^5$). Quotient is the next digit.
- ▶ Divide remainder by 16 ($= 2^4$). Quotient is the next digit.
- ▶ Divide remainder by 8 ($= 2^3$). Quotient is the next digit.
- ▶ Divide remainder by 4 ($= 2^2$). Quotient is the next digit.
- ▶ Divide remainder by 2 ($= 2^1$). Quotient is the next digit.
- ▶ The last remainder is the last digit.
- ▶ Example: what is 130 in binary notation?

130/128 is 1 rem 2.

Decimal to Binary: Converting Between Bases

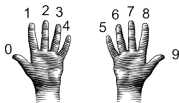


● From decimal to binary:

- ▶ Divide by 128 ($= 2^7$). Quotient is the first digit.
- ▶ Divide remainder by 64 ($= 2^6$). Quotient is the next digit.
- ▶ Divide remainder by 32 ($= 2^5$). Quotient is the next digit.
- ▶ Divide remainder by 16 ($= 2^4$). Quotient is the next digit.
- ▶ Divide remainder by 8 ($= 2^3$). Quotient is the next digit.
- ▶ Divide remainder by 4 ($= 2^2$). Quotient is the next digit.
- ▶ Divide remainder by 2 ($= 2^1$). Quotient is the next digit.
- ▶ The last remainder is the last digit.
- ▶ Example: what is 130 in binary notation?

130/128 is 1 rem 2. First digit is 1:

Decimal to Binary: Converting Between Bases



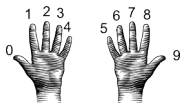
● From decimal to binary:

- ▶ Divide by 128 ($= 2^7$). Quotient is the first digit.
- ▶ Divide remainder by 64 ($= 2^6$). Quotient is the next digit.
- ▶ Divide remainder by 32 ($= 2^5$). Quotient is the next digit.
- ▶ Divide remainder by 16 ($= 2^4$). Quotient is the next digit.
- ▶ Divide remainder by 8 ($= 2^3$). Quotient is the next digit.
- ▶ Divide remainder by 4 ($= 2^2$). Quotient is the next digit.
- ▶ Divide remainder by 2 ($= 2^1$). Quotient is the next digit.
- ▶ The last remainder is the last digit.
- ▶ Example: what is 130 in binary notation?

130/128 is 1 rem 2. First digit is 1: 1...

2/64 is 0 rem 2.

Decimal to Binary: Converting Between Bases



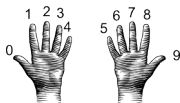
● From decimal to binary:

- ▶ Divide by 128 ($= 2^7$). Quotient is the first digit.
- ▶ Divide remainder by 64 ($= 2^6$). Quotient is the next digit.
- ▶ Divide remainder by 32 ($= 2^5$). Quotient is the next digit.
- ▶ Divide remainder by 16 ($= 2^4$). Quotient is the next digit.
- ▶ Divide remainder by 8 ($= 2^3$). Quotient is the next digit.
- ▶ Divide remainder by 4 ($= 2^2$). Quotient is the next digit.
- ▶ Divide remainder by 2 ($= 2^1$). Quotient is the next digit.
- ▶ The last remainder is the last digit.
- ▶ Example: what is 130 in binary notation?

130/128 is 1 rem 2. First digit is 1: 1...

2/64 is 0 rem 2. Next digit is 0:

Decimal to Binary: Converting Between Bases



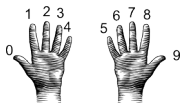
● From decimal to binary:

- ▶ Divide by 128 ($= 2^7$). Quotient is the first digit.
- ▶ Divide remainder by 64 ($= 2^6$). Quotient is the next digit.
- ▶ Divide remainder by 32 ($= 2^5$). Quotient is the next digit.
- ▶ Divide remainder by 16 ($= 2^4$). Quotient is the next digit.
- ▶ Divide remainder by 8 ($= 2^3$). Quotient is the next digit.
- ▶ Divide remainder by 4 ($= 2^2$). Quotient is the next digit.
- ▶ Divide remainder by 2 ($= 2^1$). Quotient is the next digit.
- ▶ The last remainder is the last digit.
- ▶ Example: what is 130 in binary notation?

130/128 is 1 rem 2. First digit is 1: 1...

2/64 is 0 rem 2. Next digit is 0: 10...

Decimal to Binary: Converting Between Bases



● From decimal to binary:

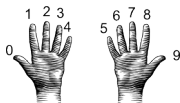
- ▶ Divide by 128 ($= 2^7$). Quotient is the first digit.
- ▶ Divide remainder by 64 ($= 2^6$). Quotient is the next digit.
- ▶ Divide remainder by 32 ($= 2^5$). Quotient is the next digit.
- ▶ Divide remainder by 16 ($= 2^4$). Quotient is the next digit.
- ▶ Divide remainder by 8 ($= 2^3$). Quotient is the next digit.
- ▶ Divide remainder by 4 ($= 2^2$). Quotient is the next digit.
- ▶ Divide remainder by 2 ($= 2^1$). Quotient is the next digit.
- ▶ The last remainder is the last digit.
- ▶ Example: what is 130 in binary notation?

130/128 is 1 rem 2. First digit is 1: 1...

2/64 is 0 rem 2. Next digit is 0: 10...

2/32 is 0 rem 2.

Decimal to Binary: Converting Between Bases



● From decimal to binary:

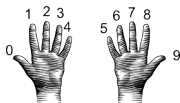
- ▶ Divide by 128 ($= 2^7$). Quotient is the first digit.
- ▶ Divide remainder by 64 ($= 2^6$). Quotient is the next digit.
- ▶ Divide remainder by 32 ($= 2^5$). Quotient is the next digit.
- ▶ Divide remainder by 16 ($= 2^4$). Quotient is the next digit.
- ▶ Divide remainder by 8 ($= 2^3$). Quotient is the next digit.
- ▶ Divide remainder by 4 ($= 2^2$). Quotient is the next digit.
- ▶ Divide remainder by 2 ($= 2^1$). Quotient is the next digit.
- ▶ The last remainder is the last digit.
- ▶ Example: what is 130 in binary notation?

130/128 is 1 rem 2. First digit is 1: 1...

2/64 is 0 rem 2. Next digit is 0: 10...

2/32 is 0 rem 2. Next digit is 0:

Decimal to Binary: Converting Between Bases



● From decimal to binary:

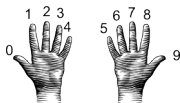
- ▶ Divide by 128 ($= 2^7$). Quotient is the first digit.
- ▶ Divide remainder by 64 ($= 2^6$). Quotient is the next digit.
- ▶ Divide remainder by 32 ($= 2^5$). Quotient is the next digit.
- ▶ Divide remainder by 16 ($= 2^4$). Quotient is the next digit.
- ▶ Divide remainder by 8 ($= 2^3$). Quotient is the next digit.
- ▶ Divide remainder by 4 ($= 2^2$). Quotient is the next digit.
- ▶ Divide remainder by 2 ($= 2^1$). Quotient is the next digit.
- ▶ The last remainder is the last digit.
- ▶ Example: what is 130 in binary notation?

130/128 is 1 rem 2. First digit is 1: 1...

2/64 is 0 rem 2. Next digit is 0: 10...

2/32 is 0 rem 2. Next digit is 0: 100...

Decimal to Binary: Converting Between Bases



● From decimal to binary:

- ▶ Divide by 128 ($= 2^7$). Quotient is the first digit.
- ▶ Divide remainder by 64 ($= 2^6$). Quotient is the next digit.
- ▶ Divide remainder by 32 ($= 2^5$). Quotient is the next digit.
- ▶ Divide remainder by 16 ($= 2^4$). Quotient is the next digit.
- ▶ Divide remainder by 8 ($= 2^3$). Quotient is the next digit.
- ▶ Divide remainder by 4 ($= 2^2$). Quotient is the next digit.
- ▶ Divide remainder by 2 ($= 2^1$). Quotient is the next digit.
- ▶ The last remainder is the last digit.
- ▶ Example: what is 130 in binary notation?

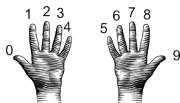
130/128 is 1 rem 2. First digit is 1: 1...

2/64 is 0 rem 2. Next digit is 0: 10...

2/32 is 0 rem 2. Next digit is 0: 100...

2/16 is 0 rem 2.

Decimal to Binary: Converting Between Bases



- From decimal to binary:

- ▶ Divide by 128 ($= 2^7$). Quotient is the first digit.
- ▶ Divide remainder by 64 ($= 2^6$). Quotient is the next digit.
- ▶ Divide remainder by 32 ($= 2^5$). Quotient is the next digit.
- ▶ Divide remainder by 16 ($= 2^4$). Quotient is the next digit.
- ▶ Divide remainder by 8 ($= 2^3$). Quotient is the next digit.
- ▶ Divide remainder by 4 ($= 2^2$). Quotient is the next digit.
- ▶ Divide remainder by 2 ($= 2^1$). Quotient is the next digit.
- ▶ The last remainder is the last digit.
- ▶ Example: what is 130 in binary notation?

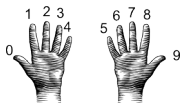
130/128 is 1 rem 2. First digit is 1: 1...

2/64 is 0 rem 2. Next digit is 0: 10...

2/32 is 0 rem 2. Next digit is 0: 100...

2/16 is 0 rem 2. Next digit is 0:

Decimal to Binary: Converting Between Bases

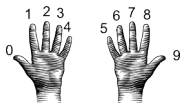


● From decimal to binary:

- ▶ Divide by 128 ($= 2^7$). Quotient is the first digit.
- ▶ Divide remainder by 64 ($= 2^6$). Quotient is the next digit.
- ▶ Divide remainder by 32 ($= 2^5$). Quotient is the next digit.
- ▶ Divide remainder by 16 ($= 2^4$). Quotient is the next digit.
- ▶ Divide remainder by 8 ($= 2^3$). Quotient is the next digit.
- ▶ Divide remainder by 4 ($= 2^2$). Quotient is the next digit.
- ▶ Divide remainder by 2 ($= 2^1$). Quotient is the next digit.
- ▶ The last remainder is the last digit.
- ▶ Example: what is 130 in binary notation?

130/128 is 1 rem 2. First digit is 1:	1...
2/64 is 0 rem 2. Next digit is 0:	10...
2/32 is 0 rem 2. Next digit is 0:	100...
2/16 is 0 rem 2. Next digit is 0:	1000...

Decimal to Binary: Converting Between Bases

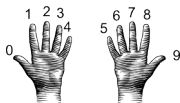


- From decimal to binary:

- ▶ Divide by 128 ($= 2^7$). Quotient is the first digit.
- ▶ Divide remainder by 64 ($= 2^6$). Quotient is the next digit.
- ▶ Divide remainder by 32 ($= 2^5$). Quotient is the next digit.
- ▶ Divide remainder by 16 ($= 2^4$). Quotient is the next digit.
- ▶ Divide remainder by 8 ($= 2^3$). Quotient is the next digit.
- ▶ Divide remainder by 4 ($= 2^2$). Quotient is the next digit.
- ▶ Divide remainder by 2 ($= 2^1$). Quotient is the next digit.
- ▶ The last remainder is the last digit.
- ▶ Example: what is 130 in binary notation?

130/128 is 1 rem 2. First digit is 1: 1...
2/64 is 0 rem 2. Next digit is 0: 10...
2/32 is 0 rem 2. Next digit is 0: 100...
2/16 is 0 rem 2. Next digit is 0: 1000...
2/8 is 0 rem 2.

Decimal to Binary: Converting Between Bases

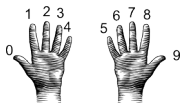


● From decimal to binary:

- ▶ Divide by 128 ($= 2^7$). Quotient is the first digit.
- ▶ Divide remainder by 64 ($= 2^6$). Quotient is the next digit.
- ▶ Divide remainder by 32 ($= 2^5$). Quotient is the next digit.
- ▶ Divide remainder by 16 ($= 2^4$). Quotient is the next digit.
- ▶ Divide remainder by 8 ($= 2^3$). Quotient is the next digit.
- ▶ Divide remainder by 4 ($= 2^2$). Quotient is the next digit.
- ▶ Divide remainder by 2 ($= 2^1$). Quotient is the next digit.
- ▶ The last remainder is the last digit.
- ▶ Example: what is 130 in binary notation?

130/128 is 1 rem 2. First digit is 1: 1...
2/64 is 0 rem 2. Next digit is 0: 10...
2/32 is 0 rem 2. Next digit is 0: 100...
2/16 is 0 rem 2. Next digit is 0: 1000...
2/8 is 0 rem 2. Next digit is 0:

Decimal to Binary: Converting Between Bases

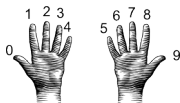


- From decimal to binary:

- ▶ Divide by 128 ($= 2^7$). Quotient is the first digit.
- ▶ Divide remainder by 64 ($= 2^6$). Quotient is the next digit.
- ▶ Divide remainder by 32 ($= 2^5$). Quotient is the next digit.
- ▶ Divide remainder by 16 ($= 2^4$). Quotient is the next digit.
- ▶ Divide remainder by 8 ($= 2^3$). Quotient is the next digit.
- ▶ Divide remainder by 4 ($= 2^2$). Quotient is the next digit.
- ▶ Divide remainder by 2 ($= 2^1$). Quotient is the next digit.
- ▶ The last remainder is the last digit.
- ▶ Example: what is 130 in binary notation?

130/128 is 1 rem 2. First digit is 1: 1...
2/64 is 0 rem 2. Next digit is 0: 10...
2/32 is 0 rem 2. Next digit is 0: 100...
2/16 is 0 rem 2. Next digit is 0: 1000...
2/8 is 0 rem 2. Next digit is 0: 10000...

Decimal to Binary: Converting Between Bases

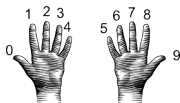


- From decimal to binary:

- ▶ Divide by 128 ($= 2^7$). Quotient is the first digit.
- ▶ Divide remainder by 64 ($= 2^6$). Quotient is the next digit.
- ▶ Divide remainder by 32 ($= 2^5$). Quotient is the next digit.
- ▶ Divide remainder by 16 ($= 2^4$). Quotient is the next digit.
- ▶ Divide remainder by 8 ($= 2^3$). Quotient is the next digit.
- ▶ Divide remainder by 4 ($= 2^2$). Quotient is the next digit.
- ▶ Divide remainder by 2 ($= 2^1$). Quotient is the next digit.
- ▶ The last remainder is the last digit.
- ▶ Example: what is 130 in binary notation?

130/128 is 1 rem 2. First digit is 1: 1...
2/64 is 0 rem 2. Next digit is 0: 10...
2/32 is 0 rem 2. Next digit is 0: 100...
2/16 is 0 rem 2. Next digit is 0: 1000...
2/8 is 0 rem 2. Next digit is 0: 10000...
2/4 is 0 remainder 2.

Decimal to Binary: Converting Between Bases

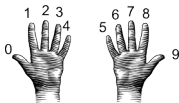


- From decimal to binary:

- ▶ Divide by 128 ($= 2^7$). Quotient is the first digit.
- ▶ Divide remainder by 64 ($= 2^6$). Quotient is the next digit.
- ▶ Divide remainder by 32 ($= 2^5$). Quotient is the next digit.
- ▶ Divide remainder by 16 ($= 2^4$). Quotient is the next digit.
- ▶ Divide remainder by 8 ($= 2^3$). Quotient is the next digit.
- ▶ Divide remainder by 4 ($= 2^2$). Quotient is the next digit.
- ▶ Divide remainder by 2 ($= 2^1$). Quotient is the next digit.
- ▶ The last remainder is the last digit.
- ▶ Example: what is 130 in binary notation?

130/128 is 1 rem 2. First digit is 1: 1...
2/64 is 0 rem 2. Next digit is 0: 10...
2/32 is 0 rem 2. Next digit is 0: 100...
2/16 is 0 rem 2. Next digit is 0: 1000...
2/8 is 0 rem 2. Next digit is 0: 10000...
2/4 is 0 remainder 2. Next digit is 0:

Decimal to Binary: Converting Between Bases

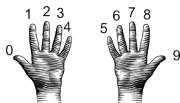


- From decimal to binary:

- ▶ Divide by 128 ($= 2^7$). Quotient is the first digit.
- ▶ Divide remainder by 64 ($= 2^6$). Quotient is the next digit.
- ▶ Divide remainder by 32 ($= 2^5$). Quotient is the next digit.
- ▶ Divide remainder by 16 ($= 2^4$). Quotient is the next digit.
- ▶ Divide remainder by 8 ($= 2^3$). Quotient is the next digit.
- ▶ Divide remainder by 4 ($= 2^2$). Quotient is the next digit.
- ▶ Divide remainder by 2 ($= 2^1$). Quotient is the next digit.
- ▶ The last remainder is the last digit.
- ▶ Example: what is 130 in binary notation?

130/128 is 1 rem 2. First digit is 1: 1...
2/64 is 0 rem 2. Next digit is 0: 10...
2/32 is 0 rem 2. Next digit is 0: 100...
2/16 is 0 rem 2. Next digit is 0: 1000...
2/8 is 0 rem 2. Next digit is 0: 10000...
2/4 is 0 remainder 2. Next digit is 0: 100000...

Decimal to Binary: Converting Between Bases

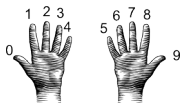


- From decimal to binary:

- ▶ Divide by 128 ($= 2^7$). Quotient is the first digit.
- ▶ Divide remainder by 64 ($= 2^6$). Quotient is the next digit.
- ▶ Divide remainder by 32 ($= 2^5$). Quotient is the next digit.
- ▶ Divide remainder by 16 ($= 2^4$). Quotient is the next digit.
- ▶ Divide remainder by 8 ($= 2^3$). Quotient is the next digit.
- ▶ Divide remainder by 4 ($= 2^2$). Quotient is the next digit.
- ▶ Divide remainder by 2 ($= 2^1$). Quotient is the next digit.
- ▶ The last remainder is the last digit.
- ▶ Example: what is 130 in binary notation?

```
130/128 is 1 rem 2. First digit is 1: 1...
2/64 is 0 rem 2. Next digit is 0:    10...
2/32 is 0 rem 2. Next digit is 0:    100...
2/16 is 0 rem 2. Next digit is 0:    1000...
2/8 is 0 rem 2. Next digit is 0:     10000...
2/4 is 0 remainder 2. Next digit is 0: 100000...
2/2 is 1 rem 0.
```


Decimal to Binary: Converting Between Bases

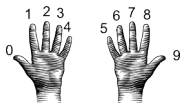


● From decimal to binary:

- ▶ Divide by 128 ($= 2^7$). Quotient is the first digit.
- ▶ Divide remainder by 64 ($= 2^6$). Quotient is the next digit.
- ▶ Divide remainder by 32 ($= 2^5$). Quotient is the next digit.
- ▶ Divide remainder by 16 ($= 2^4$). Quotient is the next digit.
- ▶ Divide remainder by 8 ($= 2^3$). Quotient is the next digit.
- ▶ Divide remainder by 4 ($= 2^2$). Quotient is the next digit.
- ▶ Divide remainder by 2 ($= 2^1$). Quotient is the next digit.
- ▶ The last remainder is the last digit.
- ▶ Example: what is 130 in binary notation?

```
130/128 is 1 rem 2. First digit is 1: 1...
2/64 is 0 rem 2. Next digit is 0:    10...
2/32 is 0 rem 2. Next digit is 0:    100...
2/16 is 0 rem 2. Next digit is 0:    1000...
2/8 is 0 rem 2. Next digit is 0:     10000...
2/4 is 0 remainder 2. Next digit is 0: 100000...
2/2 is 1 rem 0. Next digit is 1:
```

Decimal to Binary: Converting Between Bases

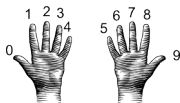


- From decimal to binary:

- ▶ Divide by 128 ($= 2^7$). Quotient is the first digit.
- ▶ Divide remainder by 64 ($= 2^6$). Quotient is the next digit.
- ▶ Divide remainder by 32 ($= 2^5$). Quotient is the next digit.
- ▶ Divide remainder by 16 ($= 2^4$). Quotient is the next digit.
- ▶ Divide remainder by 8 ($= 2^3$). Quotient is the next digit.
- ▶ Divide remainder by 4 ($= 2^2$). Quotient is the next digit.
- ▶ Divide remainder by 2 ($= 2^1$). Quotient is the next digit.
- ▶ The last remainder is the last digit.
- ▶ Example: what is 130 in binary notation?

```
130/128 is 1 rem 2. First digit is 1: 1...
2/64 is 0 rem 2. Next digit is 0: 10...
2/32 is 0 rem 2. Next digit is 0: 100...
2/16 is 0 rem 2. Next digit is 0: 1000...
2/8 is 0 rem 2. Next digit is 0: 10000...
2/4 is 0 remainder 2. Next digit is 0: 100000...
2/2 is 1 rem 0. Next digit is 1: 1000001...
```

Decimal to Binary: Converting Between Bases



● From decimal to binary:

- ▶ Divide by 128 ($= 2^7$). Quotient is the first digit.
- ▶ Divide remainder by 64 ($= 2^6$). Quotient is the next digit.
- ▶ Divide remainder by 32 ($= 2^5$). Quotient is the next digit.
- ▶ Divide remainder by 16 ($= 2^4$). Quotient is the next digit.
- ▶ Divide remainder by 8 ($= 2^3$). Quotient is the next digit.
- ▶ Divide remainder by 4 ($= 2^2$). Quotient is the next digit.
- ▶ Divide remainder by 2 ($= 2^1$). Quotient is the next digit.
- ▶ The last remainder is the last digit.
- ▶ Example: what is 130 in binary notation?

130/128 is 1 rem 2. First digit is 1: 1...

2/64 is 0 rem 2. Next digit is 0: 10...

2/32 is 0 rem 2. Next digit is 0: 100...

2/16 is 0 rem 2. Next digit is 0: 1000...

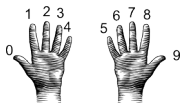
2/8 is 0 rem 2. Next digit is 0: 10000...

2/4 is 0 remainder 2. Next digit is 0: 100000...

2/2 is 1 rem 0. Next digit is 1: 1000001...

Adding the last remainder: 10000010

Decimal to Binary: Converting Between Bases



- From decimal to binary:

- ▶ Divide by 128 ($= 2^7$). Quotient is the first digit.
- ▶ Divide remainder by 64 ($= 2^6$). Quotient is the next digit.
- ▶ Divide remainder by 32 ($= 2^5$). Quotient is the next digit.
- ▶ Divide remainder by 16 ($= 2^4$). Quotient is the next digit.
- ▶ Divide remainder by 8 ($= 2^3$). Quotient is the next digit.
- ▶ Divide remainder by 4 ($= 2^2$). Quotient is the next digit.
- ▶ Divide remainder by 2 ($= 2^1$). Quotient is the next digit.
- ▶ The last remainder is the last digit.
- ▶ Example: what is 130 in binary notation?

130/128 is 1 rem 2. First digit is 1: 1...

2/64 is 0 rem 2. Next digit is 0: 10...

2/32 is 0 rem 2. Next digit is 0: 100...

2/16 is 0 rem 2. Next digit is 0: 1000...

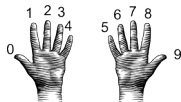
2/8 is 0 rem 2. Next digit is 0: 10000...

2/4 is 0 remainder 2. Next digit is 0: 100000...

2/2 is 1 rem 0. Next digit is 1: 1000001...

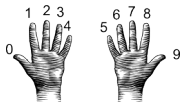
Adding the last remainder: 10000010

Decimal to Binary: Converting Between Bases



- Example: what is 99 in binary notation?

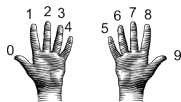
Decimal to Binary: Converting Between Bases



- Example: what is 99 in binary notation?

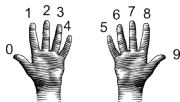
99/128 is 0 rem 99.

Decimal to Binary: Converting Between Bases



- Example: what is 99 in binary notation?
99/128 is 0 rem 99. First digit is 0:

Decimal to Binary: Converting Between Bases

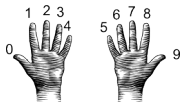


- Example: what is 99 in binary notation?

99/128 is 0 rem 99. First digit is 0: 0...

99/64 is 1 rem 35.

Decimal to Binary: Converting Between Bases

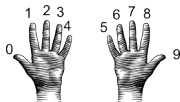


- Example: what is 99 in binary notation?

99/128 is 0 rem 99. First digit is 0: 0...

99/64 is 1 rem 35. Next digit is 1:

Decimal to Binary: Converting Between Bases

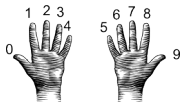


- Example: what is 99 in binary notation?

99/128 is 0 rem 99. First digit is 0: 0...

99/64 is 1 rem 35. Next digit is 1: 01...

Decimal to Binary: Converting Between Bases



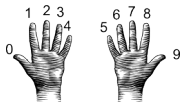
- Example: what is 99 in binary notation?

99/128 is 0 rem 99. First digit is 0: 0...

99/64 is 1 rem 35. Next digit is 1: 01...

35/32 is 1 rem 3.

Decimal to Binary: Converting Between Bases



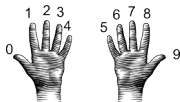
- Example: what is 99 in binary notation?

99/128 is 0 rem 99. First digit is 0: 0...

99/64 is 1 rem 35. Next digit is 1: 01...

35/32 is 1 rem 3. Next digit is 1:

Decimal to Binary: Converting Between Bases



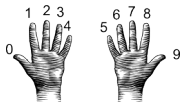
- Example: what is 99 in binary notation?

99/128 is 0 rem 99. First digit is 0: 0...

99/64 is 1 rem 35. Next digit is 1: 01...

35/32 is 1 rem 3. Next digit is 1: 011...

Decimal to Binary: Converting Between Bases



- Example: what is 99 in binary notation?

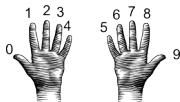
99/128 is 0 rem 99. First digit is 0: 0...

99/64 is 1 rem 35. Next digit is 1: 01...

35/32 is 1 rem 3. Next digit is 1: 011...

3/16 is 0 rem 3.

Decimal to Binary: Converting Between Bases



- Example: what is 99 in binary notation?

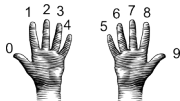
99/128 is 0 rem 99. First digit is 0: 0...

99/64 is 1 rem 35. Next digit is 1: 01...

35/32 is 1 rem 3. Next digit is 1: 011...

3/16 is 0 rem 3. Next digit is 0:

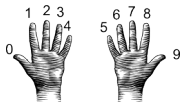
Decimal to Binary: Converting Between Bases



- Example: what is 99 in binary notation?

99/128 is 0 rem 99. First digit is 0: 0...
99/64 is 1 rem 35. Next digit is 1: 01...
35/32 is 1 rem 3. Next digit is 1: 011...
3/16 is 0 rem 3. Next digit is 0: 0110...

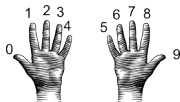
Decimal to Binary: Converting Between Bases



- Example: what is 99 in binary notation?

99/128 is 0 rem 99. First digit is 0: 0...
99/64 is 1 rem 35. Next digit is 1: 01...
35/32 is 1 rem 3. Next digit is 1: 011...
3/16 is 0 rem 3. Next digit is 0: 0110...
3/8 is 0 rem 3.

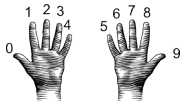
Decimal to Binary: Converting Between Bases



- Example: what is 99 in binary notation?

99/128 is 0 rem 99. First digit is 0: 0...
99/64 is 1 rem 35. Next digit is 1: 01...
35/32 is 1 rem 3. Next digit is 1: 011...
3/16 is 0 rem 3. Next digit is 0: 0110...
3/8 is 0 rem 3. Next digit is 0:

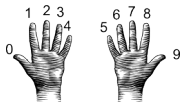
Decimal to Binary: Converting Between Bases



- Example: what is 99 in binary notation?

99/128 is 0 rem 99. First digit is 0: 0...
99/64 is 1 rem 35. Next digit is 1: 01...
35/32 is 1 rem 3. Next digit is 1: 011...
3/16 is 0 rem 3. Next digit is 0: 0110...
3/8 is 0 rem 3. Next digit is 0: 01100...

Decimal to Binary: Converting Between Bases



- Example: what is 99 in binary notation?

99/128 is 0 rem 99. First digit is 0: 0...

99/64 is 1 rem 35. Next digit is 1: 01...

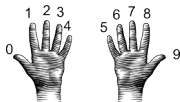
35/32 is 1 rem 3. Next digit is 1: 011...

3/16 is 0 rem 3. Next digit is 0: 0110...

3/8 is 0 rem 3. Next digit is 0: 01100...

3/4 is 0 remainder 3.

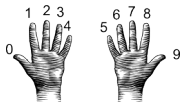
Decimal to Binary: Converting Between Bases



- Example: what is 99 in binary notation?

99/128 is 0 rem 99. First digit is 0: 0...
99/64 is 1 rem 35. Next digit is 1: 01...
35/32 is 1 rem 3. Next digit is 1: 011...
3/16 is 0 rem 3. Next digit is 0: 0110...
3/8 is 0 rem 3. Next digit is 0: 01100...
3/4 is 0 remainder 3. Next digit is 0:

Decimal to Binary: Converting Between Bases



- Example: what is 99 in binary notation?

99/128 is 0 rem 99. First digit is 0: 0...

99/64 is 1 rem 35. Next digit is 1: 01...

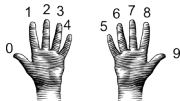
35/32 is 1 rem 3. Next digit is 1: 011...

3/16 is 0 rem 3. Next digit is 0: 0110...

3/8 is 0 rem 3. Next digit is 0: 01100...

3/4 is 0 remainder 3. Next digit is 0: 011000...

Decimal to Binary: Converting Between Bases



- Example: what is 99 in binary notation?

99/128 is 0 rem 99. First digit is 0: 0...

99/64 is 1 rem 35. Next digit is 1: 01...

35/32 is 1 rem 3. Next digit is 1: 011...

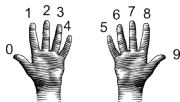
3/16 is 0 rem 3. Next digit is 0: 0110...

3/8 is 0 rem 3. Next digit is 0: 01100...

3/4 is 0 remainder 3. Next digit is 0: 011000...

3/2 is 1 rem 1.

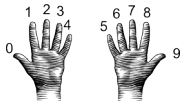
Decimal to Binary: Converting Between Bases



- Example: what is 99 in binary notation?

99/128 is 0 rem 99. First digit is 0: 0...
99/64 is 1 rem 35. Next digit is 1: 01...
35/32 is 1 rem 3. Next digit is 1: 011...
3/16 is 0 rem 3. Next digit is 0: 0110...
3/8 is 0 rem 3. Next digit is 0: 01100...
3/4 is 0 remainder 3. Next digit is 0: 011000...
3/2 is 1 rem 1. Next digit is 1:

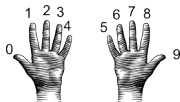
Decimal to Binary: Converting Between Bases



- Example: what is 99 in binary notation?

99/128 is 0 rem 99. First digit is 0:	0...
99/64 is 1 rem 35. Next digit is 1:	01...
35/32 is 1 rem 3. Next digit is 1:	011...
3/16 is 0 rem 3. Next digit is 0:	0110...
3/8 is 0 rem 3. Next digit is 0:	01100...
3/4 is 0 remainder 3. Next digit is 0:	011000...
3/2 is 1 rem 1. Next digit is 1:	0110001...

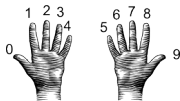
Decimal to Binary: Converting Between Bases



- Example: what is 99 in binary notation?

```
99/128 is 0 rem 99. First digit is 0:    0...
99/64 is 1 rem 35. Next digit is 1:      01...
35/32 is 1 rem 3. Next digit is 1:       011...
3/16 is 0 rem 3. Next digit is 0:        0110...
3/8 is 0 rem 3. Next digit is 0:         01100...
3/4 is 0 remainder 3. Next digit is 0:    011000...
3/2 is 1 rem 1. Next digit is 1:         0110001...
Adding the last remainder:               01100011
```

Decimal to Binary: Converting Between Bases

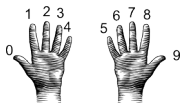


- Example: what is 99 in binary notation?

99/128 is 0 rem 99. First digit is 0:	0...
99/64 is 1 rem 35. Next digit is 1:	01...
35/32 is 1 rem 3. Next digit is 1:	011...
3/16 is 0 rem 3. Next digit is 0:	0110...
3/8 is 0 rem 3. Next digit is 0:	01100...
3/4 is 0 remainder 3. Next digit is 0:	011000...
3/2 is 1 rem 1. Next digit is 1:	0110001...
Adding the last remainder:	01100011

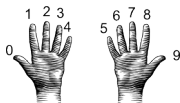
Answer is 1100011.

Binary to Decimal: Converting Between Bases



- From binary to decimal:
 - ▶ Set sum = last digit.

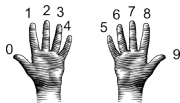
Binary to Decimal: Converting Between Bases



- From binary to decimal:

- ▶ Set sum = last digit.
- ▶ Multiply next digit by $2 = 2^1$. Add to sum.

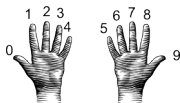
Binary to Decimal: Converting Between Bases



- From binary to decimal:

- ▶ Set sum = last digit.
- ▶ Multiply next digit by $2 = 2^1$. Add to sum.
- ▶ Multiply next digit by $4 = 2^2$. Add to sum.

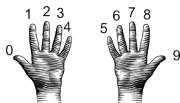
Binary to Decimal: Converting Between Bases



● From binary to decimal:

- ▶ Set sum = last digit.
- ▶ Multiply next digit by $2 = 2^1$. Add to sum.
- ▶ Multiply next digit by $4 = 2^2$. Add to sum.
- ▶ Multiply next digit by $8 = 2^3$. Add to sum.

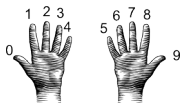
Binary to Decimal: Converting Between Bases



● From binary to decimal:

- ▶ Set sum = last digit.
- ▶ Multiply next digit by $2 = 2^1$. Add to sum.
- ▶ Multiply next digit by $4 = 2^2$. Add to sum.
- ▶ Multiply next digit by $8 = 2^3$. Add to sum.
- ▶ Multiply next digit by $16 = 2^4$. Add to sum.

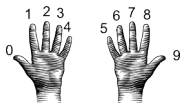
Binary to Decimal: Converting Between Bases



● From binary to decimal:

- ▶ Set sum = last digit.
- ▶ Multiply next digit by $2 = 2^1$. Add to sum.
- ▶ Multiply next digit by $4 = 2^2$. Add to sum.
- ▶ Multiply next digit by $8 = 2^3$. Add to sum.
- ▶ Multiply next digit by $16 = 2^4$. Add to sum.
- ▶ Multiply next digit by $32 = 2^5$. Add to sum.

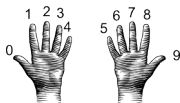
Binary to Decimal: Converting Between Bases



● From binary to decimal:

- ▶ Set sum = last digit.
- ▶ Multiply next digit by $2 = 2^1$. Add to sum.
- ▶ Multiply next digit by $4 = 2^2$. Add to sum.
- ▶ Multiply next digit by $8 = 2^3$. Add to sum.
- ▶ Multiply next digit by $16 = 2^4$. Add to sum.
- ▶ Multiply next digit by $32 = 2^5$. Add to sum.
- ▶ Multiply next digit by $64 = 2^6$. Add to sum.

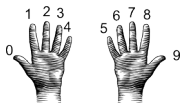
Binary to Decimal: Converting Between Bases



● From binary to decimal:

- ▶ Set sum = last digit.
- ▶ Multiply next digit by $2 = 2^1$. Add to sum.
- ▶ Multiply next digit by $4 = 2^2$. Add to sum.
- ▶ Multiply next digit by $8 = 2^3$. Add to sum.
- ▶ Multiply next digit by $16 = 2^4$. Add to sum.
- ▶ Multiply next digit by $32 = 2^5$. Add to sum.
- ▶ Multiply next digit by $64 = 2^6$. Add to sum.
- ▶ Multiply next digit by $128 = 2^7$. Add to sum.

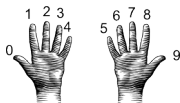
Binary to Decimal: Converting Between Bases



● From binary to decimal:

- ▶ Set sum = last digit.
- ▶ Multiply next digit by $2 = 2^1$. Add to sum.
- ▶ Multiply next digit by $4 = 2^2$. Add to sum.
- ▶ Multiply next digit by $8 = 2^3$. Add to sum.
- ▶ Multiply next digit by $16 = 2^4$. Add to sum.
- ▶ Multiply next digit by $32 = 2^5$. Add to sum.
- ▶ Multiply next digit by $64 = 2^6$. Add to sum.
- ▶ Multiply next digit by $128 = 2^7$. Add to sum.
- ▶ Sum is the decimal number.

Binary to Decimal: Converting Between Bases

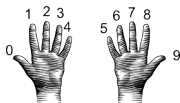


● From binary to decimal:

- ▶ Set sum = last digit.
- ▶ Multiply next digit by $2 = 2^1$. Add to sum.
- ▶ Multiply next digit by $4 = 2^2$. Add to sum.
- ▶ Multiply next digit by $8 = 2^3$. Add to sum.
- ▶ Multiply next digit by $16 = 2^4$. Add to sum.
- ▶ Multiply next digit by $32 = 2^5$. Add to sum.
- ▶ Multiply next digit by $64 = 2^6$. Add to sum.
- ▶ Multiply next digit by $128 = 2^7$. Add to sum.
- ▶ Sum is the decimal number.
- ▶ Example: What is 111101 in decimal?

Sum starts with:

Binary to Decimal: Converting Between Bases



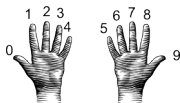
● From binary to decimal:

- ▶ Set sum = last digit.
- ▶ Multiply next digit by $2 = 2^1$. Add to sum.
- ▶ Multiply next digit by $4 = 2^2$. Add to sum.
- ▶ Multiply next digit by $8 = 2^3$. Add to sum.
- ▶ Multiply next digit by $16 = 2^4$. Add to sum.
- ▶ Multiply next digit by $32 = 2^5$. Add to sum.
- ▶ Multiply next digit by $64 = 2^6$. Add to sum.
- ▶ Multiply next digit by $128 = 2^7$. Add to sum.
- ▶ Sum is the decimal number.
- ▶ Example: What is 111101 in decimal?

Sum starts with: 1

$0 \times 2 = 0$. Add 0 to sum:

Binary to Decimal: Converting Between Bases



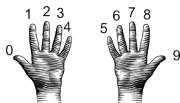
● From binary to decimal:

- ▶ Set sum = last digit.
- ▶ Multiply next digit by $2 = 2^1$. Add to sum.
- ▶ Multiply next digit by $4 = 2^2$. Add to sum.
- ▶ Multiply next digit by $8 = 2^3$. Add to sum.
- ▶ Multiply next digit by $16 = 2^4$. Add to sum.
- ▶ Multiply next digit by $32 = 2^5$. Add to sum.
- ▶ Multiply next digit by $64 = 2^6$. Add to sum.
- ▶ Multiply next digit by $128 = 2^7$. Add to sum.
- ▶ Sum is the decimal number.
- ▶ Example: What is 111101 in decimal?

Sum starts with: 1

$0 \times 2 = 0$. Add 0 to sum: 1

Binary to Decimal: Converting Between Bases

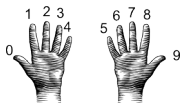


● From binary to decimal:

- ▶ Set sum = last digit.
- ▶ Multiply next digit by $2 = 2^1$. Add to sum.
- ▶ Multiply next digit by $4 = 2^2$. Add to sum.
- ▶ Multiply next digit by $8 = 2^3$. Add to sum.
- ▶ Multiply next digit by $16 = 2^4$. Add to sum.
- ▶ Multiply next digit by $32 = 2^5$. Add to sum.
- ▶ Multiply next digit by $64 = 2^6$. Add to sum.
- ▶ Multiply next digit by $128 = 2^7$. Add to sum.
- ▶ Sum is the decimal number.
- ▶ Example: What is 111101 in decimal?

Sum starts with: 1
 $0 \times 2 = 0$. Add 0 to sum: 1
 $1 \times 4 = 4$. Add 4 to sum:

Binary to Decimal: Converting Between Bases

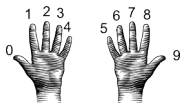


● From binary to decimal:

- ▶ Set sum = last digit.
- ▶ Multiply next digit by $2 = 2^1$. Add to sum.
- ▶ Multiply next digit by $4 = 2^2$. Add to sum.
- ▶ Multiply next digit by $8 = 2^3$. Add to sum.
- ▶ Multiply next digit by $16 = 2^4$. Add to sum.
- ▶ Multiply next digit by $32 = 2^5$. Add to sum.
- ▶ Multiply next digit by $64 = 2^6$. Add to sum.
- ▶ Multiply next digit by $128 = 2^7$. Add to sum.
- ▶ Sum is the decimal number.
- ▶ Example: What is 111101 in decimal?

Sum starts with: 1
 $0 \times 2 = 0$. Add 0 to sum: 1
 $1 \times 4 = 4$. Add 4 to sum: 5

Binary to Decimal: Converting Between Bases

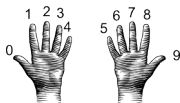


● From binary to decimal:

- ▶ Set sum = last digit.
- ▶ Multiply next digit by $2 = 2^1$. Add to sum.
- ▶ Multiply next digit by $4 = 2^2$. Add to sum.
- ▶ Multiply next digit by $8 = 2^3$. Add to sum.
- ▶ Multiply next digit by $16 = 2^4$. Add to sum.
- ▶ Multiply next digit by $32 = 2^5$. Add to sum.
- ▶ Multiply next digit by $64 = 2^6$. Add to sum.
- ▶ Multiply next digit by $128 = 2^7$. Add to sum.
- ▶ Sum is the decimal number.
- ▶ Example: What is 111101 in decimal?

Sum starts with:	1
$0 \times 2 = 0$. Add 0 to sum:	1
$1 \times 4 = 4$. Add 4 to sum:	5
$1 \times 8 = 8$. Add 8 to sum:	

Binary to Decimal: Converting Between Bases

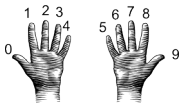


● From binary to decimal:

- ▶ Set sum = last digit.
- ▶ Multiply next digit by $2 = 2^1$. Add to sum.
- ▶ Multiply next digit by $4 = 2^2$. Add to sum.
- ▶ Multiply next digit by $8 = 2^3$. Add to sum.
- ▶ Multiply next digit by $16 = 2^4$. Add to sum.
- ▶ Multiply next digit by $32 = 2^5$. Add to sum.
- ▶ Multiply next digit by $64 = 2^6$. Add to sum.
- ▶ Multiply next digit by $128 = 2^7$. Add to sum.
- ▶ Sum is the decimal number.
- ▶ Example: What is 111101 in decimal?

Sum starts with:	1
$0 \times 2 = 0$. Add 0 to sum:	1
$1 \times 4 = 4$. Add 4 to sum:	5
$1 \times 8 = 8$. Add 8 to sum:	13

Binary to Decimal: Converting Between Bases

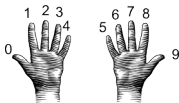


● From binary to decimal:

- ▶ Set sum = last digit.
- ▶ Multiply next digit by $2 = 2^1$. Add to sum.
- ▶ Multiply next digit by $4 = 2^2$. Add to sum.
- ▶ Multiply next digit by $8 = 2^3$. Add to sum.
- ▶ Multiply next digit by $16 = 2^4$. Add to sum.
- ▶ Multiply next digit by $32 = 2^5$. Add to sum.
- ▶ Multiply next digit by $64 = 2^6$. Add to sum.
- ▶ Multiply next digit by $128 = 2^7$. Add to sum.
- ▶ Sum is the decimal number.
- ▶ Example: What is 111101 in decimal?

Sum starts with:	1
$0 \times 2 = 0$. Add 0 to sum:	1
$1 \times 4 = 4$. Add 4 to sum:	5
$1 \times 8 = 8$. Add 8 to sum:	13
$1 \times 16 = 16$. Add 16 to sum:	

Binary to Decimal: Converting Between Bases

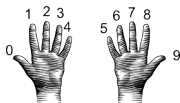


● From binary to decimal:

- ▶ Set sum = last digit.
- ▶ Multiply next digit by $2 = 2^1$. Add to sum.
- ▶ Multiply next digit by $4 = 2^2$. Add to sum.
- ▶ Multiply next digit by $8 = 2^3$. Add to sum.
- ▶ Multiply next digit by $16 = 2^4$. Add to sum.
- ▶ Multiply next digit by $32 = 2^5$. Add to sum.
- ▶ Multiply next digit by $64 = 2^6$. Add to sum.
- ▶ Multiply next digit by $128 = 2^7$. Add to sum.
- ▶ Sum is the decimal number.
- ▶ Example: What is 111101 in decimal?

Sum starts with:	1
$0 \times 2 = 0$. Add 0 to sum:	1
$1 \times 4 = 4$. Add 4 to sum:	5
$1 \times 8 = 8$. Add 8 to sum:	13
$1 \times 16 = 16$. Add 16 to sum:	29

Binary to Decimal: Converting Between Bases

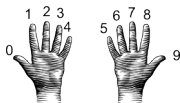


● From binary to decimal:

- ▶ Set sum = last digit.
- ▶ Multiply next digit by $2 = 2^1$. Add to sum.
- ▶ Multiply next digit by $4 = 2^2$. Add to sum.
- ▶ Multiply next digit by $8 = 2^3$. Add to sum.
- ▶ Multiply next digit by $16 = 2^4$. Add to sum.
- ▶ Multiply next digit by $32 = 2^5$. Add to sum.
- ▶ Multiply next digit by $64 = 2^6$. Add to sum.
- ▶ Multiply next digit by $128 = 2^7$. Add to sum.
- ▶ Sum is the decimal number.
- ▶ Example: What is 111101 in decimal?

Sum starts with:	1
$0 \times 2 = 0$. Add 0 to sum:	1
$1 \times 4 = 4$. Add 4 to sum:	5
$1 \times 8 = 8$. Add 8 to sum:	13
$1 \times 16 = 16$. Add 16 to sum:	29
$1 \times 32 = 32$. Add 32 to sum:	

Binary to Decimal: Converting Between Bases

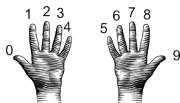


● From binary to decimal:

- ▶ Set sum = last digit.
- ▶ Multiply next digit by $2 = 2^1$. Add to sum.
- ▶ Multiply next digit by $4 = 2^2$. Add to sum.
- ▶ Multiply next digit by $8 = 2^3$. Add to sum.
- ▶ Multiply next digit by $16 = 2^4$. Add to sum.
- ▶ Multiply next digit by $32 = 2^5$. Add to sum.
- ▶ Multiply next digit by $64 = 2^6$. Add to sum.
- ▶ Multiply next digit by $128 = 2^7$. Add to sum.
- ▶ Sum is the decimal number.
- ▶ Example: What is 111101 in decimal?

Sum starts with:	1
$0 \times 2 = 0$. Add 0 to sum:	1
$1 \times 4 = 4$. Add 4 to sum:	5
$1 \times 8 = 8$. Add 8 to sum:	13
$1 \times 16 = 16$. Add 16 to sum:	29
$1 \times 32 = 32$. Add 32 to sum:	61

Binary to Decimal: Converting Between Bases

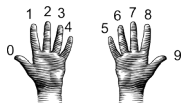


● From binary to decimal:

- ▶ Set sum = last digit.
- ▶ Multiply next digit by $2 = 2^1$. Add to sum.
- ▶ Multiply next digit by $4 = 2^2$. Add to sum.
- ▶ Multiply next digit by $8 = 2^3$. Add to sum.
- ▶ Multiply next digit by $16 = 2^4$. Add to sum.
- ▶ Multiply next digit by $32 = 2^5$. Add to sum.
- ▶ Multiply next digit by $64 = 2^6$. Add to sum.
- ▶ Multiply next digit by $128 = 2^7$. Add to sum.
- ▶ Sum is the decimal number.
- ▶ Example: What is 111101 in decimal?

Sum starts with:	1
$0 \times 2 = 0$. Add 0 to sum:	1
$1 \times 4 = 4$. Add 4 to sum:	5
$1 \times 8 = 8$. Add 8 to sum:	13
$1 \times 16 = 16$. Add 16 to sum:	29
$1 \times 32 = 32$. Add 32 to sum:	61

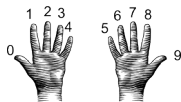
Binary to Decimal: Converting Between Bases



- Example: What is 10100100 in decimal?

Sum starts with:

Binary to Decimal: Converting Between Bases

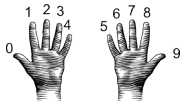


- Example: What is 10100100 in decimal?

Sum starts with: 0

$0 * 2 = 0$. Add 0 to sum:

Binary to Decimal: Converting Between Bases

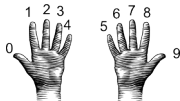


- Example: What is 10100100 in decimal?

Sum starts with: 0

$0 * 2 = 0$. Add 0 to sum: 0

Binary to Decimal: Converting Between Bases



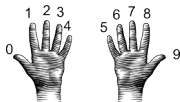
- Example: What is 10100100 in decimal?

Sum starts with: 0

$0 \times 2 = 0$. Add 0 to sum: 0

$1 \times 4 = 4$. Add 4 to sum:

Binary to Decimal: Converting Between Bases



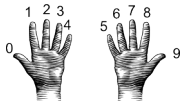
- Example: What is 10100100 in decimal?

Sum starts with: 0

$0 \times 2 = 0$. Add 0 to sum: 0

$1 \times 4 = 4$. Add 4 to sum: 4

Binary to Decimal: Converting Between Bases



- Example: What is 10100100 in decimal?

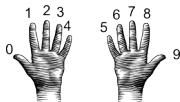
Sum starts with: 0

$0 \times 2 = 0$. Add 0 to sum: 0

$1 \times 4 = 4$. Add 4 to sum: 4

$0 \times 8 = 0$. Add 0 to sum:

Binary to Decimal: Converting Between Bases



- Example: What is 10100100 in decimal?

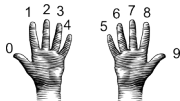
Sum starts with: 0

$0 \times 2 = 0$. Add 0 to sum: 0

$1 \times 4 = 4$. Add 4 to sum: 4

$0 \times 8 = 0$. Add 0 to sum: 4

Binary to Decimal: Converting Between Bases



- Example: What is 10100100 in decimal?

Sum starts with: 0

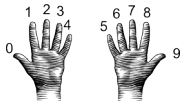
$0 \times 2 = 0$. Add 0 to sum: 0

$1 \times 4 = 4$. Add 4 to sum: 4

$0 \times 8 = 0$. Add 0 to sum: 4

$0 \times 16 = 0$. Add 0 to sum:

Binary to Decimal: Converting Between Bases



- Example: What is 10100100 in decimal?

Sum starts with: 0

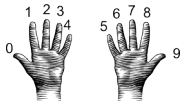
$0 \times 2 = 0$. Add 0 to sum: 0

$1 \times 4 = 4$. Add 4 to sum: 4

$0 \times 8 = 0$. Add 0 to sum: 4

$0 \times 16 = 0$. Add 0 to sum: 4

Binary to Decimal: Converting Between Bases



- Example: What is 10100100 in decimal?

Sum starts with: 0

$0 \times 2 = 0$. Add 0 to sum: 0

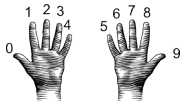
$1 \times 4 = 4$. Add 4 to sum: 4

$0 \times 8 = 0$. Add 0 to sum: 4

$0 \times 16 = 0$. Add 0 to sum: 4

$1 \times 32 = 32$. Add 32 to sum:

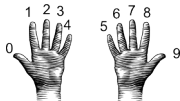
Binary to Decimal: Converting Between Bases



- Example: What is 10100100 in decimal?

Sum starts with:	0
$0 \times 2 = 0$. Add 0 to sum:	0
$1 \times 4 = 4$. Add 4 to sum:	4
$0 \times 8 = 0$. Add 0 to sum:	4
$0 \times 16 = 0$. Add 0 to sum:	4
$1 \times 32 = 32$. Add 32 to sum:	36

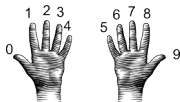
Binary to Decimal: Converting Between Bases



- Example: What is 10100100 in decimal?

Sum starts with:	0
$0 \times 2 = 0$. Add 0 to sum:	0
$1 \times 4 = 4$. Add 4 to sum:	4
$0 \times 8 = 0$. Add 0 to sum:	4
$0 \times 16 = 0$. Add 0 to sum:	4
$1 \times 32 = 32$. Add 32 to sum:	36
$0 \times 64 = 0$. Add 0 to sum:	

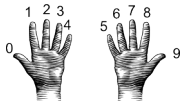
Binary to Decimal: Converting Between Bases



- Example: What is 10100100 in decimal?

Sum starts with:	0
$0 \times 2 = 0$. Add 0 to sum:	0
$1 \times 4 = 4$. Add 4 to sum:	4
$0 \times 8 = 0$. Add 0 to sum:	4
$0 \times 16 = 0$. Add 0 to sum:	4
$1 \times 32 = 32$. Add 32 to sum:	36
$0 \times 64 = 0$. Add 0 to sum:	36

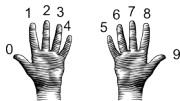
Binary to Decimal: Converting Between Bases



- Example: What is 10100100 in decimal?

Sum starts with:	0
$0 \times 2 = 0$. Add 0 to sum:	0
$1 \times 4 = 4$. Add 4 to sum:	4
$0 \times 8 = 0$. Add 0 to sum:	4
$0 \times 16 = 0$. Add 0 to sum:	4
$1 \times 32 = 32$. Add 32 to sum:	36
$0 \times 64 = 0$. Add 0 to sum:	36
$1 \times 128 = 128$. Add 128 to sum:	

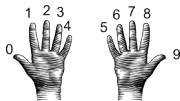
Binary to Decimal: Converting Between Bases



- Example: What is 10100100 in decimal?

Sum starts with:	0
$0 \times 2 = 0$. Add 0 to sum:	0
$1 \times 4 = 4$. Add 4 to sum:	4
$0 \times 8 = 0$. Add 0 to sum:	4
$0 \times 16 = 0$. Add 0 to sum:	4
$1 \times 32 = 32$. Add 32 to sum:	36
$0 \times 64 = 0$. Add 0 to sum:	36
$1 \times 128 = 128$. Add 128 to sum:	164

Binary to Decimal: Converting Between Bases

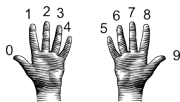


- Example: What is 10100100 in decimal?

Sum starts with:	0
$0 \times 2 = 0$. Add 0 to sum:	0
$1 \times 4 = 4$. Add 4 to sum:	4
$0 \times 8 = 0$. Add 0 to sum:	4
$0 \times 16 = 0$. Add 0 to sum:	4
$1 \times 32 = 32$. Add 32 to sum:	36
$0 \times 64 = 0$. Add 0 to sum:	36
$1 \times 128 = 128$. Add 128 to sum:	164

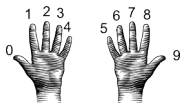
The answer is 164.

Design Challenge: Incrementers



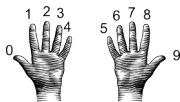
- Simplest arithmetic: add one (“increment”) a variable.

Design Challenge: Incrementers



- Simplest arithmetic: add one (“increment”) a variable.
- Example: Increment a decimal number:

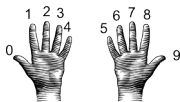
Design Challenge: Incrementers



- Simplest arithmetic: add one (“increment”) a variable.
- Example: Increment a decimal number:

```
def addOne(n):  
    m = n+1  
    return(m)
```

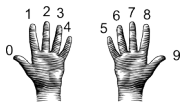
Design Challenge: Incrementers



- Simplest arithmetic: add one (“increment”) a variable.
- Example: Increment a decimal number:

```
def addOne(n):  
    m = n+1  
    return(m)
```
- Challenge: Write an algorithm for incrementing numbers expressed as words.

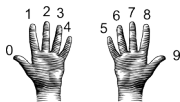
Design Challenge: Incrementers



- Simplest arithmetic: add one (“increment”) a variable.
- Example: Increment a decimal number:

```
def addOne(n):  
    m = n+1  
    return(m)
```
- Challenge: Write an algorithm for incrementing numbers expressed as words.
Example: "forty one" → "forty two"

Design Challenge: Incrementers



- Simplest arithmetic: add one ("increment") a variable.

- Example: Increment a decimal number:

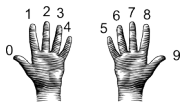
```
def addOne(n):  
    m = n+1  
    return(m)
```

- Challenge: Write an algorithm for incrementing numbers expressed as words.

Example: "forty one" → "forty two"

Hint: Convert to numbers, increment, and convert back to strings.

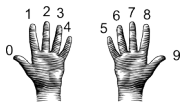
Design Challenge: Incrementers



- Simplest arithmetic: add one (“increment”) a variable.
- Example: Increment a decimal number:

```
def addOne(n):  
    m = n+1  
    return(m)
```
- Challenge: Write an algorithm for incrementing numbers expressed as words.
Example: "forty one" → "forty two"
Hint: Convert to numbers, increment, and convert back to strings.
- Challenge: Write an algorithm for incrementing binary numbers.

Design Challenge: Incrementers



- Simplest arithmetic: add one (“increment”) a variable.
- Example: Increment a decimal number:

```
def addOne(n):  
    m = n+1  
    return(m)
```
- Challenge: Write an algorithm for incrementing numbers expressed as words.
Example: "forty one" → "forty two"
Hint: Convert to numbers, increment, and convert back to strings.
- Challenge: Write an algorithm for incrementing binary numbers.
Example: "1001" → "1010"

Recap

- On lecture slip, write down a topic you wish we had spent more time (and why).



Recap



- On lecture slip, write down a topic you wish we had spent more time (and why).
- Searching through data is a common task– built-in functions and standard design patterns for this.

Recap



- On lecture slip, write down a topic you wish we had spent more time (and why).
- Searching through data is a common task– built-in functions and standard design patterns for this.
- Programming languages can be classified by the level of abstraction and direct access to data.

Recap



- On lecture slip, write down a topic you wish we had spent more time (and why).
- Searching through data is a common task– built-in functions and standard design patterns for this.
- Programming languages can be classified by the level of abstraction and direct access to data.
- Pass your lecture slips to the aisles for the UTAs to collect.

Today's Topics



- Design Patterns: Searching
- Python Recap
- Machine Language
- Machine Language: Jumps & Loops
- Binary & Hex Arithmetic
- **Final Exam: Format**

Final Overview: Format

- The exam is 2 hours long.

Final Overview: Format

- The exam is 2 hours long.
- There are 4 different versions to discourage copying.

Final Overview: Format

- The exam is 2 hours long.
- There are 4 different versions to discourage copying.
- It is on paper. No use of computers, phones, etc. allowed.

Final Overview: Format

- The exam is 2 hours long.
- There are 4 different versions to discourage copying.
- It is on paper. No use of computers, phones, etc. allowed.
- You may have 1 piece of **8.5" x 11"** piece of paper.

Final Overview: Format

- The exam is 2 hours long.
- There are 4 different versions to discourage copying.
- It is on paper. No use of computers, phones, etc. allowed.
- You may have 1 piece of **8.5" x 11"** piece of paper.
 - ▶ With notes, examples, programs: what will help you on the exam.

Final Overview: Format

- The exam is 2 hours long.
- There are 4 different versions to discourage copying.
- It is on paper. No use of computers, phones, etc. allowed.
- You may have 1 piece of **8.5" x 11"** piece of paper.
 - ▶ With notes, examples, programs: what will help you on the exam.
 - ▶ No origami– it's distracting to others taking the exam.

Final Overview: Format

- The exam is 2 hours long.
- There are 4 different versions to discourage copying.
- It is on paper. No use of computers, phones, etc. allowed.
- You may have 1 piece of **8.5" x 11"** piece of paper.
 - ▶ With notes, examples, programs: what will help you on the exam.
 - ▶ No origami– it's distracting to others taking the exam.
 - ▶ Best if you design/write yours since excellent way to study.

Final Overview: Format

- The exam is 2 hours long.
- There are 4 different versions to discourage copying.
- It is on paper. No use of computers, phones, etc. allowed.
- You may have 1 piece of **8.5" x 11"** piece of paper.
 - ▶ With notes, examples, programs: what will help you on the exam.
 - ▶ No origami– it's distracting to others taking the exam.
 - ▶ Best if you design/write yours since excellent way to study.
- The exam format:

Final Overview: Format

- The exam is 2 hours long.
- There are 4 different versions to discourage copying.
- It is on paper. No use of computers, phones, etc. allowed.
- You may have 1 piece of **8.5" x 11"** piece of paper.
 - ▶ With notes, examples, programs: what will help you on the exam.
 - ▶ No origami– it's distracting to others taking the exam.
 - ▶ Best if you design/write yours since excellent way to study.
- The exam format:
 - ▶ 10 questions, each worth 10 points.

Final Overview: Format

- The exam is 2 hours long.
- There are 4 different versions to discourage copying.
- It is on paper. No use of computers, phones, etc. allowed.
- You may have 1 piece of **8.5" x 11"** piece of paper.
 - ▶ With notes, examples, programs: what will help you on the exam.
 - ▶ No origami– it's distracting to others taking the exam.
 - ▶ Best if you design/write yours since excellent way to study.
- The exam format:
 - ▶ 10 questions, each worth 10 points.
 - ▶ Questions correspond to the course topics, and are variations on the programming assignments, lab exercises, and lecture design challenges.
 - ▶ Style of questions: what does the code do? short answer, write functions, top down design, & write complete programs.

Final Overview: Format

- The exam is 2 hours long.
- There are 4 different versions to discourage copying.
- It is on paper. No use of computers, phones, etc. allowed.
- You may have 1 piece of **8.5" x 11"** piece of paper.
 - ▶ With notes, examples, programs: what will help you on the exam.
 - ▶ No origami– it's distracting to others taking the exam.
 - ▶ Best if you design/write yours since excellent way to study.
- The exam format:
 - ▶ 10 questions, each worth 10 points.
 - ▶ Questions correspond to the course topics, and are variations on the programming assignments, lab exercises, and lecture design challenges.
 - ▶ Style of questions: what does the code do? short answer, write functions, top down design, & write complete programs.
 - ▶ More on logistics next lecture.

Final Overview: Format

- The exam is 2 hours long.
- There are 4 different versions to discourage copying.
- It is on paper. No use of computers, phones, etc. allowed.
- You may have 1 piece of **8.5" x 11"** piece of paper.
 - ▶ With notes, examples, programs: what will help you on the exam.
 - ▶ No origami– it's distracting to others taking the exam.
 - ▶ Best if you design/write yours since excellent way to study.
- The exam format:
 - ▶ 10 questions, each worth 10 points.
 - ▶ Questions correspond to the course topics, and are variations on the programming assignments, lab exercises, and lecture design challenges.
 - ▶ Style of questions: what does the code do? short answer, write functions, top down design, & write complete programs.
 - ▶ More on logistics next lecture.
- Past exams available on webpage (includes answer keys).

Exam Options

Exam Times:

FINAL EXAM, VERSION 3
CSci 127: Introduction to Computer Science
Hunter College, City University of New York

19 December 2019

Exam Rules

- Show all your work. Your grade will be based on the work shown.
- The exam is closed-book and closed-notes with the exception of an 8 1/2" x 11" piece of paper filled with notes, programs, etc.
- When taking the exam, you may have with you pens and pencils, and your watch only.
- You may not use a computer, calculator, tablet, smart watch, or other electronic device.
- Do not open this exam until instructed to do so.

Hunter College regards acts of academic dishonesty (i.e., plagiarism, cheating on examinations, obtaining unfair advantage, and distribution of exams and official documents to various officers) against the values of intellectual honesty. The College is committed to enforcing the CUNY Policy on Academic Integrity and will pursue cases of academic dishonesty according to its Hunter College Academic Integrity Procedures.

I am aware that all cases of academic dishonesty will be reported to the Dean of Hunter College.
Name:
Signature:
Date:
Signature:

Exam Options

Exam Times:

- **Default: Regular Time: Monday, 16 December, 9-11am.**
- **Alternate Time: Reading Day, Friday, 13 December, 8:30am-10:30am.**
- **Accessibility Testing Center: Paperwork required. Must be completed on 13 December.**

FINAL EXAM, VERSION 3
CSci 127: Introduction to Computer Science
Hunter College, City University of New York

13 December 2019

Exam Rules

- Show all your work. Your grade will be based on the work shown.
- The exam is closed book and closed notes with the exception of an 8 1/2" x 11" piece of paper filled with notes, programs, etc.
- When taking the exam, you may have with you pens and pencils, and your watch clock.
- You may not use a computer, calculator, cell phone, smart watch, or other electronic device.
- Do not open this exam until instructed to do so.

Hunter College regards acts of academic dishonesty (i.e., plagiarism, cheating on examinations, obtaining unfair advantage, and distribution of exams and official documents to various officers) against the values of intellectual honesty. The College is committed to enforcing the CUNY Policy on Academic Integrity and will pursue cases of academic dishonesty according to its Hunter College Academic Integrity Procedures.

I, the undersigned, have read the rules of academic dishonesty and I agree to the Hunter College Academic Integrity Procedures.	
Name	
Signature	
Date	
Signature	

Exam Options

Exam Times:

- **Default: Regular Time: Monday, 16 December, 9-11am.**
- **Alternate Time: Reading Day, Friday, 13 December, 8:30am-10:30am.**
- **Accessibility Testing Center: Paperwork required. Must be completed on 13 December.**

FINAL EXAM, VERSION 3
CSci 127: Introduction to Computer Science
Hunter College, City University of New York

13 December 2019

Exam Rules

- Show all your work. Your grade will be based on the work shown.
- The exam is closed book and closed notes with the exception of an 8 1/2" x 11" piece of paper filled with notes prepared in advance.
- When taking the exam, you may have with you pens and pencils, and your watch clock.
- You may not use a computer, calculator, cell phone, smart watch, or other electronic device.
- Do not open this exam until instructed to do so.

Hunter College regards acts of academic dishonesty (i.e., plagiarism, cheating on examinations, obtaining unfair advantage, and distribution of exams and official documents to various officers) against the values of intellectual honesty. The College is committed to enforcing the CUNY Policy on Academic Integrity and will pursue cases of academic dishonesty according to its Hunter College Academic Integrity Procedures.

I, _____, declare that all work of academic dishonesty will be reported to the Dean of Hunter College.
Name
Signature
Date
Signature

Grading Options:

Exam Options

Exam Times:

- **Default: Regular Time: Monday, 16 December, 9-11am.**
- **Alternate Time: Reading Day, Friday, 13 December, 8:30am-10:30am.**
- **Accessibility Testing Center: Paperwork required. Must be completed on 13 December.**

FINAL EXAM, VERSION 3
CSci 127: Introduction to Computer Science
Hunter College, City University of New York

13 December 2019

Exam Rules

- Show all your work. Your grade will be based on the work shown.
- The exam is closed book and closed notes with the exception of an 8 1/2" x 11" piece of paper filled with notes prepared in advance.
- When taking the exam, you may have with you pens and pencils, and your watch.
- You may not use a computer, calculator, cell phone, smart watch, or other electronic device.
- Do not open this exam until instructed to do so.

Hunter College regards acts of academic dishonesty (i.e., plagiarism, cheating on examinations, obtaining unfair advantage, and distribution of exams and official documents to various officers) against the values of intellectual honesty. The College is committed to enforcing the CUNY Policy on Academic Integrity and will pursue cases of academic dishonesty according to its Hunter College Academic Integrity Procedures.

I, _____, declare that all content of academic dishonesty will be reported to the Dean of Hunter College.
Name:
Signature:
Date:
Signature:

Grading Options:

- **Default: Letter Grade.**
- **Credit/NoCredit grade— availability depends on major and academic standing.**

Exam Options

Exam Times:

- **Default: Regular Time: Monday, 16 December, 9-11am.**
- **Alternate Time: Reading Day, Friday, 13 December, 8:30am-10:30am.**
- **Accessibility Testing Center: Paperwork required. Must be completed on 13 December.**

FINAL EXAM, VERSION 3
CSci 127: Introduction to Computer Science
Hunter College, City University of New York
13 December 2019

Exam Rules

- Show all your work. Your grade will be based on the work shown.
- The exam is closed book and closed notes with the exception of an 8 1/2" x 11" piece of paper filled with notes, prepared in advance.
- When taking the exam, you may have with you pens and pencils, and your watch only.
- You may not use a calculator, cell phone, smart watch, or other electronic device.
- Do not open this exam until instructed to do so.

Hunter College regards acts of academic dishonesty (i.e., plagiarism, cheating on examinations, obtaining unfair advantage, and distribution of exams and official documents to persons off-campus) as a violation of academic integrity. The College is committed to enforcing the CUNY Policy on Academic Integrity and will pursue cases of academic dishonesty according to the Hunter College Academic Integrity Procedures.

I, _____, declare that all work on this exam is my own and that I have not received any unauthorized assistance.	
Name:	
Signature:	
Date:	
Signature:	

Grading Options:

- **Default: Letter Grade.**
- **Credit/NoCredit grade— availability depends on major and academic standing.**

Forms for your choices (“pink slips”) available next lecture.

Writing Boards



- Return writing boards as you leave...