Summer Prep Course for Discrete Mathematics:

- Focuses on increasing algebraic mastery, especially with manipulating expressions involving multiple variables, indices, as well as summation notation.
- The focus is on the mastery of skills critical to success in CSci 150.
- The course is 1 hour, 1 credit, Wednesdays, 9:50-11:50am.
- See CUNYFirst for registration.
Announcements

- Summer Prep Course for Discrete Mathematics:
  - Math10N03: “Pre-Discrete Mathematics Workshop”
Announcements

- Summer Prep Course for Discrete Mathematics:
  - Math10N03: “Pre-Discrete Mathematics Workshop”
  - Focuses on increasing algebraic mastery, especially with manipulating expressions involving multiple variables, indices, as well as summation notation.

See CUNYFirst for registration.
Announcements

- Summer Prep Course for Discrete Mathematics:
  - Math10N03: “Pre-Discrete Mathematics Workshop”
  - Focuses on increasing algebraic mastery, especially with manipulating expressions involving multiple variables, indices, as well as summation notation.
  - The focus is on the mastery of skills critical to success in CSci 150.
Announcements

- Summer Prep Course for Discrete Mathematics:
  - Math10N03: “Pre-Discrete Mathematics Workshop”
  - Focuses on increasing algebraic mastery, especially with manipulating expressions involving multiple variables, indices, as well as summation notation.
  - The focus is on the mastery of skills critical to success in CSci 150.
  - The course is 1 hour, 1 credit, Wednesdays, 9:50-11:50am.

See CUNYFirst for registration.
Summer Prep Course for Discrete Mathematics:

- Math10N03: “Pre-Discrete Mathematics Workshop”
- Focuses on increasing algebraic mastery, especially with manipulating expressions involving multiple variables, indices, as well as summation notation.
- The focus is on the mastery of skills critical to success in CSci 150.
- The course is 1 hour, 1 credit, Wednesdays, 9:50-11:50am.
- See CUNYFirst for registration.
Today’s Topics

- Recap of Python & Circuits
- High vs. Low-Level Programming
- A Simplified Machine Language
- Final Exam Overview
A whirlwind tour of the semester, so far...
Week 1: print(), loops, comments, & turtles
Week 1: print(), loops, comments, & turtles

- Introduced comments & print():

```python
#Name: Thomas Hunter
#Date: September 1, 2017
#This program prints: Hello, World!

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

- As well as definite loops & the turtle package:

```python
# Prints the string "Hello, World!" to the screen
```
Week 1: print(), loops, comments, & turtles

- Introduced comments & print():

  ```python
  # Name: Thomas Hunter
  # Date: September 1, 2017
  # This program prints: Hello, World!
  
  print("Hello, World!")
  ```

  These lines are comments
  (for us, not computer to read)
  (this one also)

  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()

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:
A variable is a reserved memory location for storing a value.
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
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

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.
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
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

For example, `[3, 1, 4, 5, 9]` or `['violet', 'purple', 'indigo']`
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.
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:

```python
#Predict what will be printed:
for num in [2, 4, 6, 8, 10]:
    print(num)

sum = 0
for x in range(0, 12, 2):
    print(x)
    sum = sum + x

print(x)

for c in "ABCD":
    print(c)
```
### Week 3: colors, hex, slices, numpy & images

<table>
<thead>
<tr>
<th>Color Name</th>
<th>HEX</th>
<th>Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black</td>
<td>#000000</td>
<td></td>
</tr>
<tr>
<td>Navy</td>
<td>#000080</td>
<td></td>
</tr>
<tr>
<td>DarkBlue</td>
<td>#000088</td>
<td></td>
</tr>
<tr>
<td>MediumBlue</td>
<td>#0000CD</td>
<td></td>
</tr>
<tr>
<td>Blue</td>
<td>#0000FF</td>
<td></td>
</tr>
</tbody>
</table>

![Hand Diagram]
Week 3: colors, hex, slices, numpy & images

<table>
<thead>
<tr>
<th>Color Name</th>
<th>HEX</th>
<th>Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black</td>
<td>#000000</td>
<td>Black</td>
</tr>
<tr>
<td>Navy</td>
<td>#000080</td>
<td>Navy</td>
</tr>
<tr>
<td>DarkBlue</td>
<td>#00008B</td>
<td>DarkBlue</td>
</tr>
<tr>
<td>MediumBlue</td>
<td>#0000CD</td>
<td>MediumBlue</td>
</tr>
<tr>
<td>Blue</td>
<td>#0000FF</td>
<td>Blue</td>
</tr>
</tbody>
</table>

© www.scratchapixel.com
Week 3: colors, hex, slices, numpy & images

<table>
<thead>
<tr>
<th>Color Name</th>
<th>HEX</th>
<th>Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black</td>
<td>#000000</td>
<td>Black</td>
</tr>
<tr>
<td>Navy</td>
<td>#000080</td>
<td>Navy</td>
</tr>
<tr>
<td>DarkBlue</td>
<td>#00008B</td>
<td>Dark Blue</td>
</tr>
<tr>
<td>MediumBlue</td>
<td>#0000CD</td>
<td>Medium Blue</td>
</tr>
<tr>
<td>Blue</td>
<td>#0000FF</td>
<td>Blue</td>
</tr>
</tbody>
</table>

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

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

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

```python
>>> a[2:,2::2]
array([[20,22,24],
       [40,42,44]])
```
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.
First: specify inputs/outputs. *Input file name, output file name, upper, lower, left, right ("bounding box")*
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.
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

```python
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

<table>
<thead>
<tr>
<th>in1</th>
<th>in2</th>
<th>returns:</th>
</tr>
</thead>
<tbody>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
</tbody>
</table>

```
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 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
1690, 4957, 4957, 4957, 4957, 4957
1771, 21603, 3623, 2867, 2043
1790, 32321, 4540, 6529, 7931, 3227, 4947
1800, 65515, 5740, 6642, 1755, 4563, 79215
1810, 96735, 8053, 7443, 2751, 5217, 121973
1820, 132706, 11187, 8264, 2700, 6135, 152058
1830, 202589, 20935, 8945, 3033, 7088, 242278
1840, 312710, 47613, 14680, 5346, 10965, 391114
1850, 512567, 138682, 185930, 8034, 19661, 696115
1860, 814661, 279122, 23903, 25995, 24902, 1147379
1870, 942592, 419921, 43468, 37393, 30309, 1478103
1880, 1164475, 599495, 56559, 51980, 39993, 1915698
1890, 1441216, 838547, 67050, 89608, 51693, 2307416
1900, 1850993, 1246982, 122099, 209027, 67921, 2931202
1910, 2331542, 1634351, 280461, 430980, 85994, 4766883
1920, 2864100, 2018385, 469042, 72216, 141651, 5620548
1930, 3673912, 2508601, 107929, 1262528, 158346, 6930446
1940, 4089264, 2698205, 1297643, 1194711, 174641, 7540995
1950, 4961011, 2738675, 155048, 1701277, 191558, 7891557
1960, 6052081, 2627649, 1809578, 1948615, 221941, 7781994
1970, 1592930, 2409312, 1896673, 171701, 295643, 7948062
1980, 1422695, 2230936, 109126, 1168972, 352121, 7071639
1990, 1487396, 2300646, 155159, 1297909, 378977, 7322564
2000, 1537195, 2465326, 2229379, 1332650, 443728, 8089270
2010, 1585073, 2504700, 2230722, 1395109, 488720, 8175133
2015, 1644518, 2638735, 2339150, 1455446, 474550, 8550405

nycHistPop.csv

In Lab 6
Week 6: structured data, pandas, & more design

import matplotlib.pyplot as plt
import pandas as pd

nycHistPop.csv

In Lab 6
import matplotlib.pyplot as plt
import pandas as pd

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

nycHistPop.csv

In Lab 6
import matplotlib.pyplot as plt
import pandas as pd

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

pop.plot(x="Year")
plt.show()
Week 6: structured data, pandas, & more design

import matplotlib.pyplot as plt
import pandas as pd

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

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

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
1689, 6957, 7176, 7481, 671, 1760
1700, 7218, 7549, 785, 711, 1798
1720, 9203, 9547, 987, 914, 2021
1780, 60515, 5740, 6642, 7158, 5683, 79215
1810, 96373, 8033, 7446, 22867, 5347, 119734
1820, 123706, 11087, 8246, 2760, 6138, 152056
1830, 202860, 20358, 8446, 3031, 7088, 242378
1840, 312710, 47613, 16480, 13446, 10965, 391114
1850, 515547, 138882, 18093, 9034, 15061, 696115
1860, 814600, 279122, 29503, 25902, 25492, 1574279
1870, 942592, 41992, 64468, 37593, 33029, 1478103
1880, 1164675, 59945, 65559, 3190, 8895, 191569
1890, 1444126, 830847, 67059, 89080, 51693, 2057416
1900, 1859093, 116992, 122999, 20907, 67202, 2597202
1910, 2331542, 1634351, 204041, 430980, 85949, 467683
1920, 2264103, 2018286, 46942, 723161, 14351, 562048
1930, 1867312, 2568401, 107019, 1265258, 158346, 693048
1940, 1889942, 2693985, 1297634, 1346711, 1746441, 745095
1950, 1966101, 2373815, 153049, 1461277, 171555, 781957
1960, 1582001, 2637349, 180978, 1486035, 2199814, 778154
1970, 1539233, 2460912, 1906673, 147101, 256433, 759462
1980, 1428296, 2230994, 18193, 1160972, 522121, 7071639
1990, 1487346, 2308644, 195158, 128789, 738977, 722264
2000, 1537195, 2455246, 2229738, 1328509, 643728, 800276
2010, 1585879, 2504700, 2230722, 139509, 688730, 8175133
2015, 1645018, 2638735, 2339150, 1455464, 745558, 8550405

nycHistPop.csv

In Lab 6
Functions are a way to break code into pieces, that can be easily reused.

```python
# 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

- 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.

```python
# 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

- 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()

```python
# 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:

```python
# 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

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,

```python
# This program, uses functions, says hello to the world!

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

if __name__ == "__main__":
    main()
```
Week 7: functions

- 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.
Functions can have input parameters.

```python
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).

```python
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**.

```python
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**.

The ones in the function call: **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.
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

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

```python
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.
Week 10: indefinite loops, searching data, random()

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

```python
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.
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
```
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/hashe 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`
A whirlwind tour with 10 (or so) challenges...
Predict what the code will do:

```python
# Predict what will be printed:

for i in range(4):
    print('The world turned upside down')

for j in [0, 1, 2, 3, 4, 5]:
    print(j)

for count in range(6):
    print(count)

for color in ['red', 'green', 'blue']:
    print(color)

print()
print()

for i in range(2):
    for j in range(2):
        print('Look around,')
        print('How lucky we are to be alive!')
```
Predict what the code will do:

```python
#Predict what will be printed:
for c in range(65,90):
    print(chr(c))
message = "I love Python"
newMessage = ""
for c in message:
    print(ord(c))    #Print the Unicode of each number
    print(chr(ord(c)+1))    #Print the next character
    newMessage = newMessage + chr(ord(c)+1) #add to the new message
print("The coded message is", newMessage)

word = "zebra"
codedWord = ""
for ch in word:
    offset = ord(ch) - ord('a') + 1 #how many letters past 'a'
    wrap = offset % 26    #if larger than 26, wrap back to 0
    newChar = chr(ord('a') + wrap) #compute the new letter
    print(wrap, chr(ord('a') + wrap))    #print the wrap & new letter
    codedWord = codedWord + newChar #add the newChar to the coded word
print("The coded word (with wrap) is", codedWord)
```
Predict what the code will do:

```python
import turtle

names = ["violet", "purple", "indigo", "lavender"]
for c in names:
    teddy.color(c)
    teddy.left(60)
    teddy.forward(40)
    teddy.dot(10)

teddy.penup()
teddy.forward(100)
teddy.pendown()

hexNames = ["#FF00FF", "#990099", "#550055", "#111111"]
for c in hexNames:
    teddy.color(c)
    teddy.left(60)
    teddy.forward(40)
    teddy.dot(10)
```
Extend this program to also allow drawing in purple & stamping:

```python
import turtle

tess = turtle.Turtle()
myWin = turtle.Screen()  # The graphics window
commands = input("Please enter a command string: ")

for ch in commands:
    # perform action indicated by the character
    if ch == 'f':  # move forward
        tess.forward(50)
    elif ch == 'l':  # turn left
        tess.left(90)
    elif ch == 'r':  # turn right
        tess.right(90)
    elif ch == '^':  # lift pen
        tess.penup()
    elif ch == 'v':  # lower pen
        tess.pendown()
    elif ch == 'b':  # go backwards
        tess.backward(50)
    elif ch == 'r':  # turn red
        tess.color("red")
    elif ch == 'g':  # turn green
        tess.color("green")
    elif ch == 'b':  # turn blue
        tess.color("blue")
    else:  # for any other character
        print("Error: do not know the command:", c)
```
In Pairs or Triples: Week 5

*When does this circuit yield true?*

*That is, what values for the inputs give an output value of true?*
Predict what the following will do:

- `print("Queens:", pop["Queens"].min())`
- `print("S I:", pop["Staten Island"].mean())`
- `print("S I:", pop["Staten Island"].std())`
- `pop.plot.bar(x="Year")`
- `pop.plot.scatter(x="Brooklyn", y= "Total")`
- `pop["Fraction"] = pop["Bronx"]/pop["Total"]`
Fill in the function body:

```python
def monthString(monthNum):
    
    Takes as input a number, monthNum, and
    returns the corresponding month name as a string.
    Example: monthString(1) returns "January".
    Assumes that input is an integer ranging from 1 to 12
    
    monthString = ""

    """""""""""""""""""""""""""""""""""""""""""""""""
    ### FILL IN YOUR CODE HERE  ###
    ### Other than your name above, ###
    ### this is the only section  ###
    ### you change in this program. ###
    """"""""""""""""""""""""""""""""""""""""""""""""

    return(monthString)

def main():
    n = int(input('Enter the number of the month: '))
    mString = monthString(n)
    print('The month is', mString)
```
In Pairs or Triples: Week 8

```python
def bar(n):
    if n <= 8:
        return 1
    else:
        return 0

def foo(l):
    n = bar(l[-1])
    return l[n]
```

- What are the formal parameters for the functions?

- What is the output of:
  
  ```python
  r = foo([1,2,3,4])
  print("Return: ", r)
  ```

- What is the output of:
  
  ```python
  r = foo([1024,512,256,128])
  print("Return: ", r)
  ```
In Pairs or Triples: Week 9

What does this code do?

```python
import folium
import pandas as pd

cuny = pd.read_csv('cunyLocations.csv')
mapCUNY = folium.Map(location=[40.75, -74.125])

for index, row in cuny.iterrows():
    lat = row["Latitude"]
    lon = row["Longitude"]
    name = row["Campus"]
    if row["College or Institution Type"] == "Senior Colleges":
        collegeIcon = folium.Icon(color="purple")
    else:
        collegeIcon = folium.Icon(color="blue")
    newMarker = folium.Marker([lat, lon], popup=name, icon=collegeIcon)
    newMarker.add_to(mapCUNY)

mapCUNY.save(outfile='cunyLocationsSenior.html')
```
In Pairs or Triples: Week 10

- Predict what the code will do:

```python
nums = [1, 4, 10, 6, 5, 42, 9, 8, 12]
maxNum = 0
for n in nums:
    if n > maxNum:
        maxNum = n
print('The max is', maxNum)
```

- Write a function that asks a user for number after 2000 but before 2018. The function should repeatedly ask the user for a number until they enter one within the range and return the number.
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
Lecture Slip: Commenting Code

In pairs or triples:
- What does the code do?
- Add comments to explain each line.
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**.
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

(wiki)
We will be writing programs in a simplified machine language, WeMIPS.
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

- 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

- 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

```assembly
# Store 'Hello world!' at the top of the stack
ADDI $sp, $sp, -13
ADDI $t0, $zero, 72 # H
SB $t0, 0($sp)
ADDI $t0, $zero, 101 # e
SB $t0, 1($sp)
ADDI $t0, $zero, 108 # l
SB $t0, 2($sp)
ADDI $t0, $zero, 108 # l
SB $t0, 3($sp)
ADDI $t0, $zero, 111 # o
SB $t0, 4($sp)
ADDI $t0, $zero, 32 # (space)
SB $t0, 5($sp)
ADDI $t0, $zero, 119 # w
SB $t0, 6($sp)
ADDI $t0, $zero, 111 # o
SB $t0, 7($sp)
ADDI $t0, $zero, 114 # r
SB $t0, 8($sp)
ADDI $t0, $zero, 108 # l
SB $t0, 9($sp)
ADDI $t0, $zero, 100 # d
SB $t0, 10($sp)
ADDI $t0, $zero, 33 # !
SB $t0, 11($sp)
ADDI $t0, $zero, 0 # (null)
SB $t0, 12($sp)
ADDI $s0, $zero, 4 # 4 is for print string
ADDI $a0, $sp, 0
syscall # print to the log
```

(WeMIPS)
Write a program that prints out the alphabet: a b c d ... x y z
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.
  - 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.
  - Style of questions: what does the code do? short answer, write functions, top down design, & write complete programs.
- Past exams available on webpage (includes answer keys).
On lecture slip, write down a topic you wish we had spent more time (and why).

```python
# 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()
```
Recap: Python, Languages, & Design

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

```python
# 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()
```
Recap: Python, Languages, & Design

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

- Python language
- Logical Circuits

```python
# 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()
```
Recap: Python, Languages, & Design

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

- Python language
- Logical Circuits
- Simplified Machine Language

```python
#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()
```
Recap: Python, Languages, & Design

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

- Python language
- Logical Circuits
- Simplified Machine Language
- Design: from written description (‘specs’) to function inputs & outputs (‘APIs’)

```python
# 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()
```
Final Overview: Top-Down Design & APIs

For each question, write only the function header (name & inputs) and return values (often called the Application Programming Interface (API)):
Final Overview: Top-Down Design & APIs

For each question, write **only the function header (name & inputs) and return values** (often called the Application Programming Interface (API)):

- Write a function that takes a weight in kilograms and returns the weight in pounds.
- Write a function that takes a string and returns its length.
- Write a function that, given a DataFrame, returns the minimal value in the first column.
- Write a function that takes a whole number and returns the corresponding binary number as a string.
- Write a function that computes the total monthly payment when given the initial loan amount, annual interest rate, number of years of the loan.
Final Overview: Top-Down Design & APIs
For each question, write only the function header (name & inputs) and return values (often called the Application Programming Interface (API)):

- Write a function that takes a weight in kilograms and returns the weight in pounds.
- Write a function that takes a string and returns its length.
- Write a function that, given a DataFrame, returns the minimal value in the first column.
- Write a function that takes a whole number and returns the corresponding binary number as a string.
- Write a function that computes the total monthly payment when given the initial loan amount, annual interest rate, number of years of the loan.

(Hint: highlight key words, make list of inputs, list of outputs, then put together.)
Final Overview

For each question, write the function header (name & inputs) and return values (often called the Application Programming Interface (API)):

- Write a function that takes a weight in kilograms and returns the weight in pounds.

```python
def kg2lbs(kg):
    lbs = 2.20462 * kg
    return lbs
```
Final Overview

For each question, write the function header (name & inputs) and return values (often called the Application Programming Interface (API)):

- Write a function that takes a weight in kilograms and returns the weight in pounds.

```
def kg2lbs(kg):
    ...
    return(lbs)
```
Final Overview

For each question, write the function header (name & inputs) and return values (often called the Application Programming Interface (API)):

- Write a function that takes a weight in kilograms and returns the weight in pounds.

```python
def kg2lbs(kg):
    lbs = kg * 2.2
    return(lbs)
```
Final Overview

For each question, write the function header (name & inputs) and return values (often called the Application Programming Interface (API)):

- Write a function that takes a string and returns its length.
Final Overview

For each question, write the function header (name & inputs) and return values (often called the Application Programming Interface (API)):

- Write a function that takes a string and returns its length.

```python
def sLength(str):
    ...
    return(length)
```
Final Overview

For each question, write the function header (name & inputs) and return values (often called the Application Programming Interface (API)):

- Write a function that takes a string and returns its length.

```python
def sLength(str):
    length = len(str)
    return(length)
```
Final Overview

For each question, write the function header (name & inputs) and return values (often called the Application Programming Interface (API)):

- Write a function that, given a DataFrame, returns the minimal value in the “Manhattan” column.
Final Overview

For each question, write the function header (name & inputs) and return values (often called the Application Programming Interface (API)):

- Write a function that, given a DataFrame, returns the minimal value in the “Manhattan” column.

```python
def getMin(df):
    ...
    return(min)
```
For each question below, write the function header (name & inputs) and return values (often called the Application Programming Interface (API)):

- Write a function that, given a DataFrame, returns the minimal value in the “Manhattan” column.

```python
def getMin(df):
    mM = df['Manhattan'].min()
    return(mM)
```
Final Overview

For each question, write the function header (name & inputs) and return values (often called the Application Programming Interface (API)):

- Write a function that takes a whole number and returns the corresponding binary number as a string.
Final Overview

For each question, write the function header (name & inputs) and return values (often called the Application Programming Interface (API)):

- Write a function that takes a whole number and returns the corresponding binary number as a string.

```python
def num2bin(num):
    ...
    return(bin)
```
Final Overview
For each question, write the function header (name & inputs) and return values (often called the Application Programming Interface (API)):

- Write a function that takes a whole number and returns the corresponding binary number as a string.

def num2bin(num):
    binStr = ""
    while (num > 0):
        #Divide by 2, and add the remainder to the string
        r = num %2
        binString = str(r) + binStr
        num = num / 2
    return(binStr)
Final Overview

For each question, write the function header (name & inputs) and return values (often called the Application Programming Interface (API)):

- Write a function that computes the total monthly payment when given the initial loan amount, annual interest rate, number of years of the loan.

```python
def computePayment(loan, rate, year):
    ...
    return(payment)
```
Final Overview

For each question, write the function header (name & inputs) and return values (often called the Application Programming Interface (API)):

- Write a function that computes the total monthly payment when given the initial loan amount, annual interest rate, number of years of the loan.

```python
def computePayment(loan, rate, year):
    ....
    return(payment)
```
Final Overview

For each question below, write the function header (name & inputs) and return values (often called the Application Programming Interface (API)):

- Write a function that computes the total monthly payment when given the initial loan amount, annual interest rate, number of years of the loan.

```python
def computePayment(loan, rate, year):
    # Some formula for payment
    payment = ...
    return payment
```