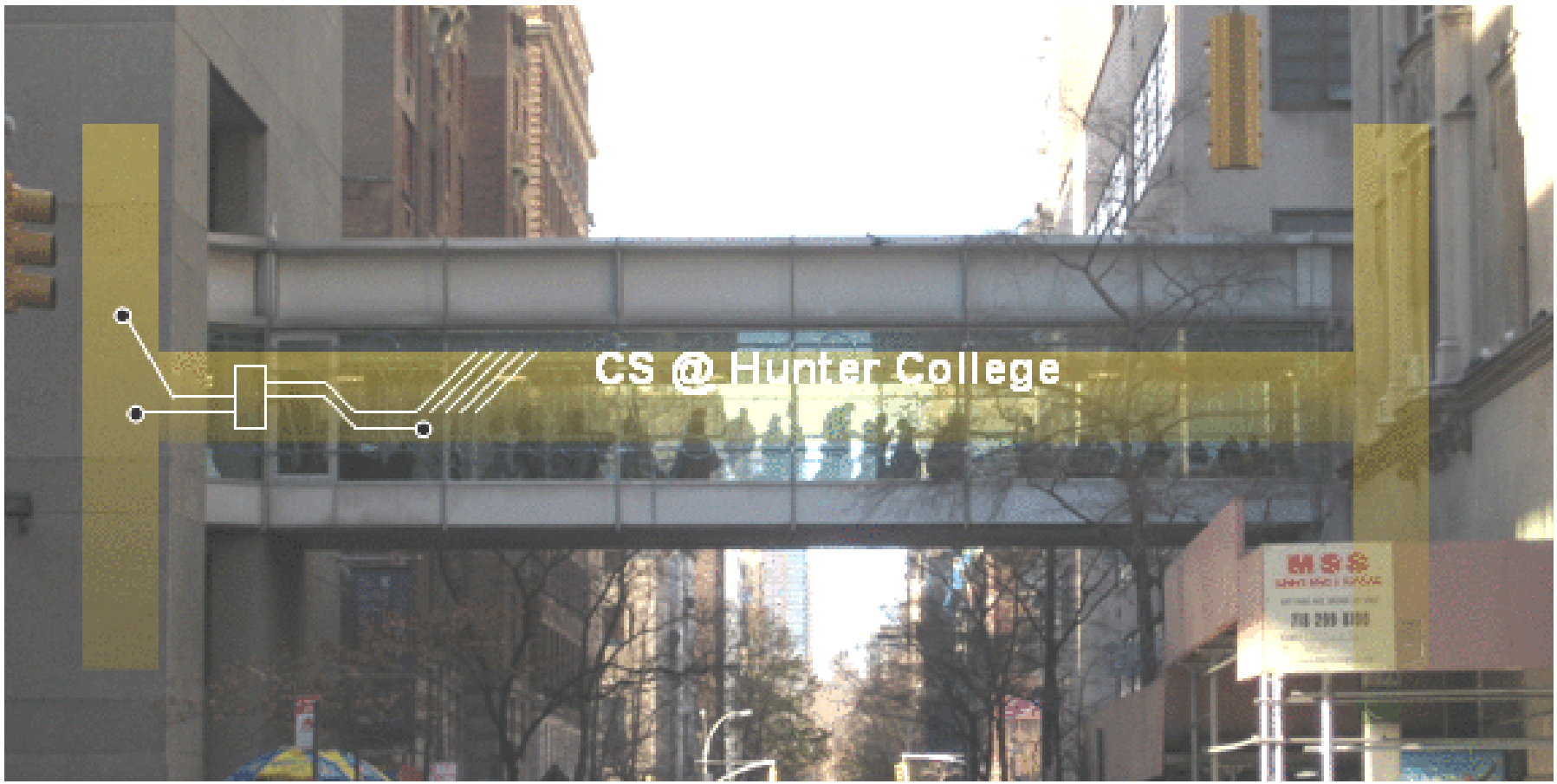
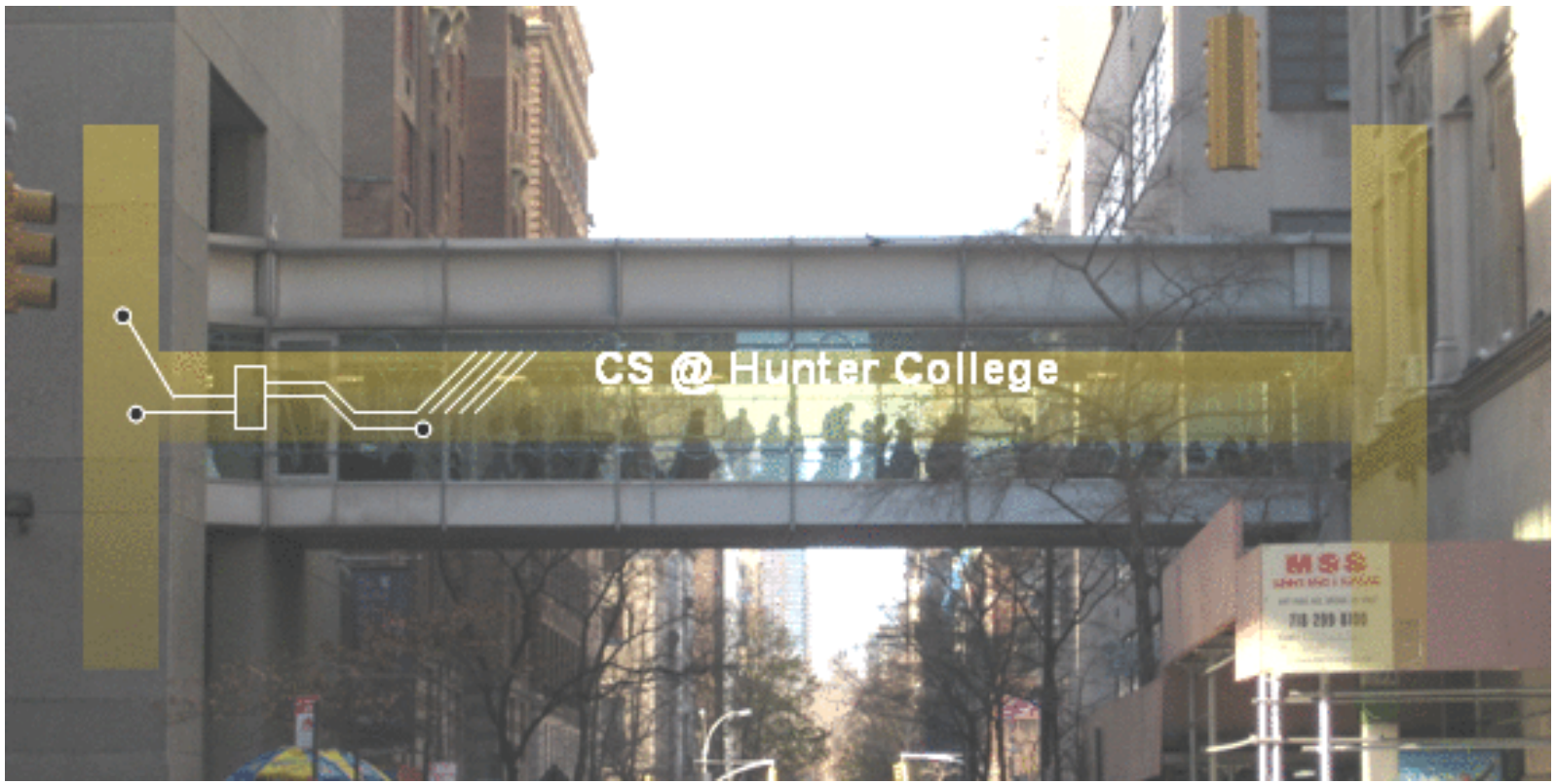
CSci 127: Introduction to Computer Science



hunter.cuny.edu/csci

Welcome



Acknowledgments

Thank you to the amazing support of:

President Raab

Dean Polsky
Arts & Science

Judy Spitz
WiTNY

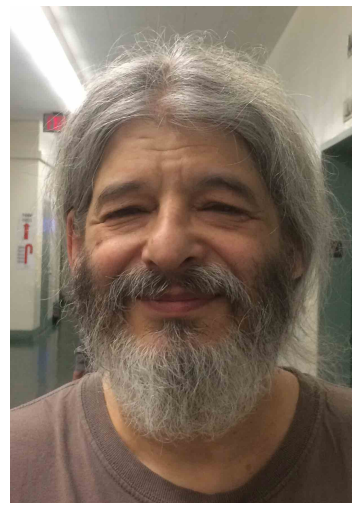
Introductions: Course Designers

Dr. Katherine St. John

Professor,
Course Coordinator

Dr. William Sakas

Associate Professor,
Chair



Prof. Eric Schweitzer

Undergraduate Program
Coordinator

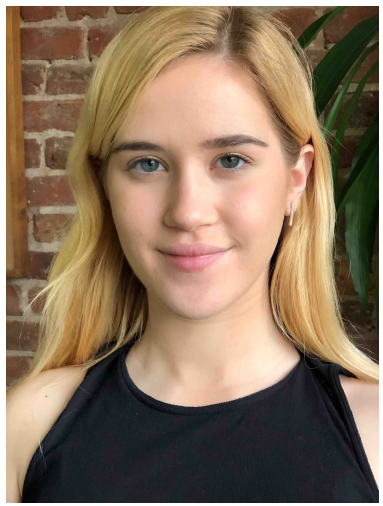
Introductions: Recitation Instructors


Alex Washburn


Alvin Lam


Jaime Canizales


Katherine Howitt

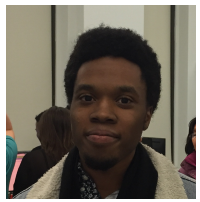

Melissa Lynch

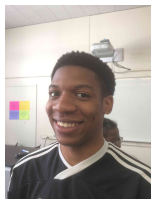

Tiziana Ligorio

Introductions: Undergraduate Teaching Assistants


Anna Lis


Antonio Bountouvas


Brian Campbell


Bryan Belmont


Camryn Buonamassa


David Yuen


Erin Williams

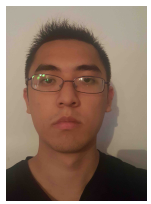

Ferdi Lesporis


Harry Wu


Mandy Yu


Michael Nurilov


Nick Szewczak


Nicky Cen


Owen Kunhardt

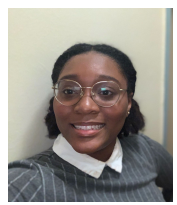

Parakram Basnet


Ralph Venté


Rhia Singh

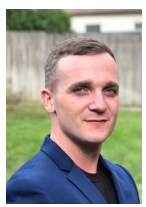

Savannah Nester


Shaina Lowenthal

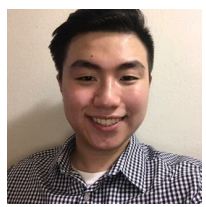

Shantel Dixon


Stephanie Yung


Stephen Milani


Thomas Joy


Tommi Ann Tsuruga


Vincent Zheng

Syllabus

CSci 127: Introduction to Computer Science

Catalog Description: 3 hours, 3 credits: This course presents an overview of computer science (CS) with an emphasis on problem-solving and computational thinking through 'coding': computer programming for beginners. Other topics include: organization of hardware, software, and how information is structured on contemporary computing devices. This course is pre-requisite to several introductory core courses in the CS Major. The course is also required for the CS minor. MATH 12500 or higher is strongly recommended as a co-req for intended Majors.

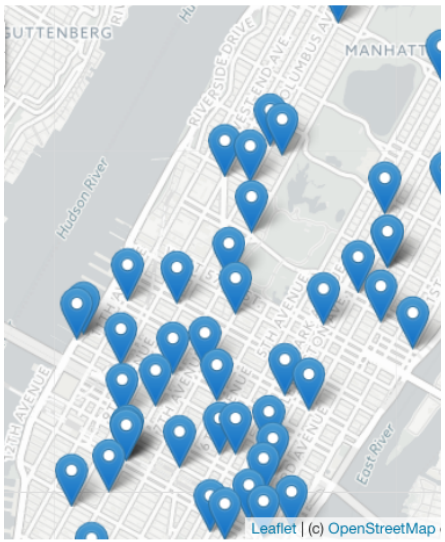
Syllabus

CSci 127: Introduction to Computer Science

Catalog Description: 3 hours, 3 credits: This course presents an overview of computer science (CS) with an emphasis on problem-solving and computational thinking through 'coding': computer programming for beginners. Other topics include: organization of hardware, software, and how information is structured on contemporary computing devices. This course is pre-requisite to several introductory core courses in the CS Major. The course is also required for the CS minor. MATH 12500 or higher is strongly recommended as a co-req for intended Majors.

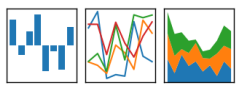
(Show syllabus webpage)

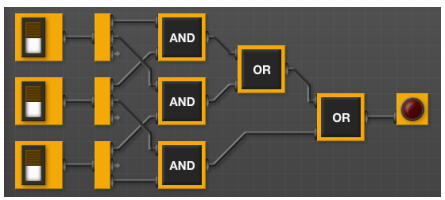
Syllabus: Topics



pandas

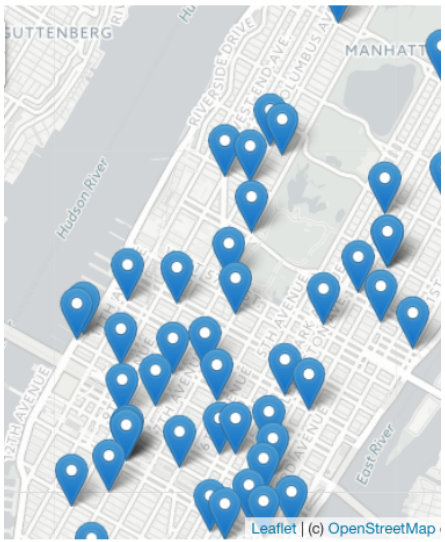
$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$

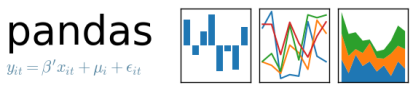


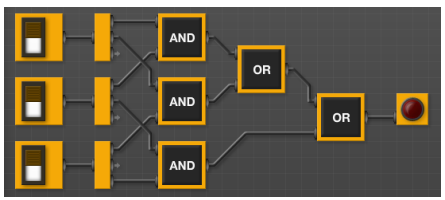


- **This course assumes no previous programming experience.**

Syllabus: Topics

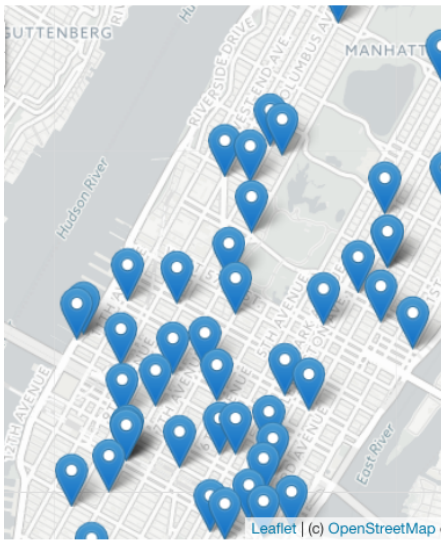


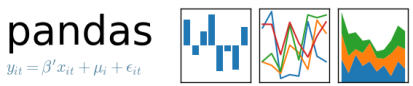


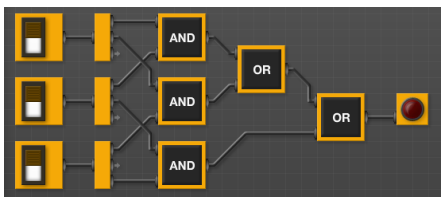


- **This course assumes no previous programming experience.**
- "... Emphasis on problem-solving and computational thinking through 'coding': computer programming for beginners..."

Syllabus: Topics

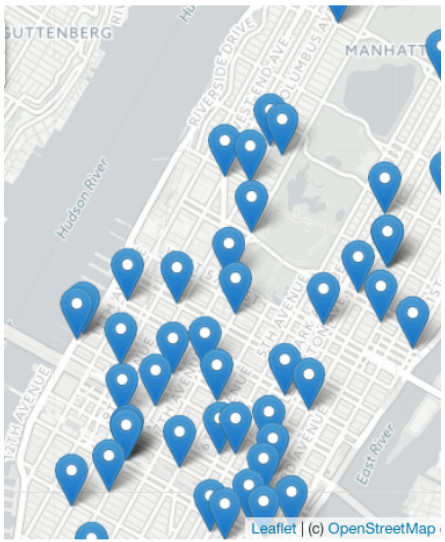


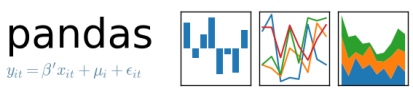


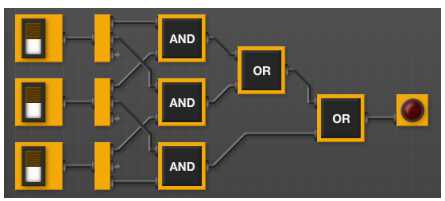


- **This course assumes no previous programming experience.**
- "... Emphasis on problem-solving and computational thinking through 'coding': computer programming for beginners..."
- Organized like a fugue, with variations on this theme:

Syllabus: Topics

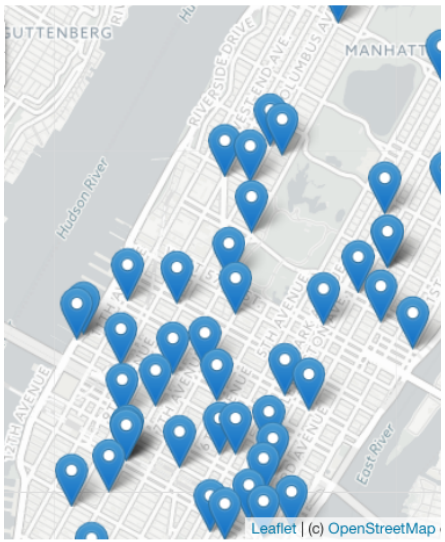






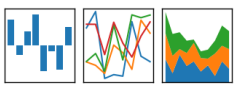
- **This course assumes no previous programming experience.**
- "... Emphasis on problem-solving and computational thinking through 'coding': computer programming for beginners..."
- Organized like a fugue, with variations on this theme:
  - Introduce coding constructs in Python,

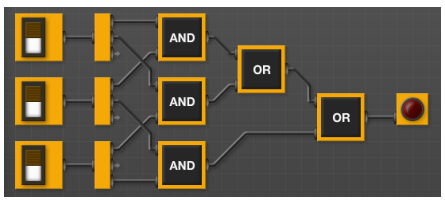
Syllabus: Topics



pandas

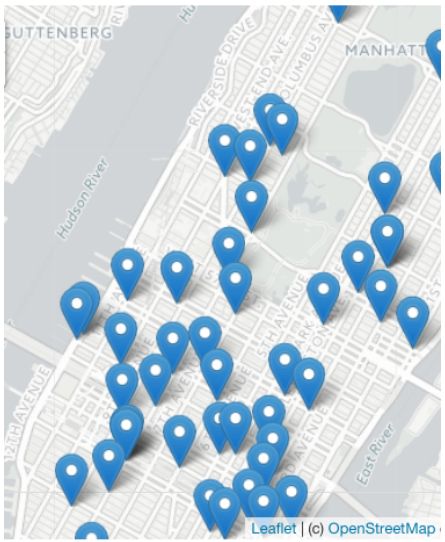
$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$





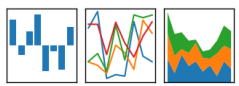
- **This course assumes no previous programming experience.**
- "... Emphasis on problem-solving and computational thinking through 'coding': computer programming for beginners..."
- Organized like a fugue, with variations on this theme:
  - Introduce coding constructs in Python,
  - Apply those ideas to different problems (e.g. analyzing & mapping data),

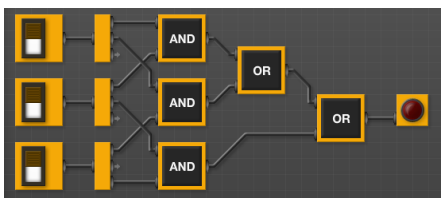
Syllabus: Topics



pandas

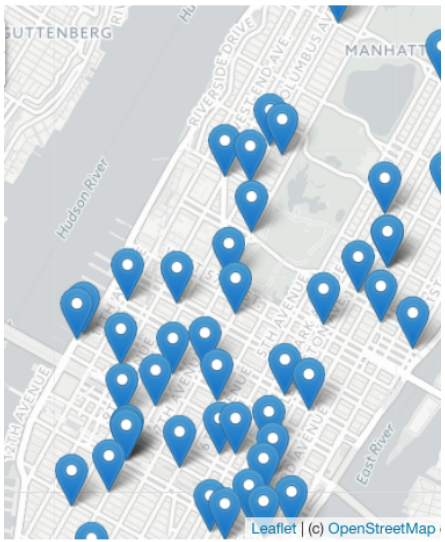
$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$

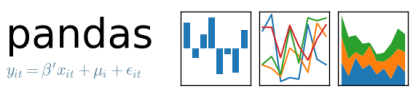


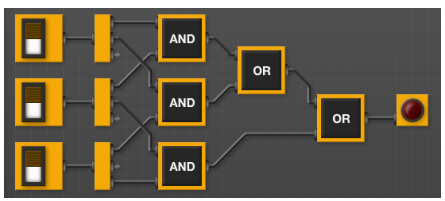


- **This course assumes no previous programming experience.**
- "... Emphasis on problem-solving and computational thinking through 'coding': computer programming for beginners..."
- Organized like a fugue, with variations on this theme:
  - Introduce coding constructs in Python,
  - Apply those ideas to different problems (e.g. analyzing & mapping data),
  - See constructs again:

Syllabus: Topics

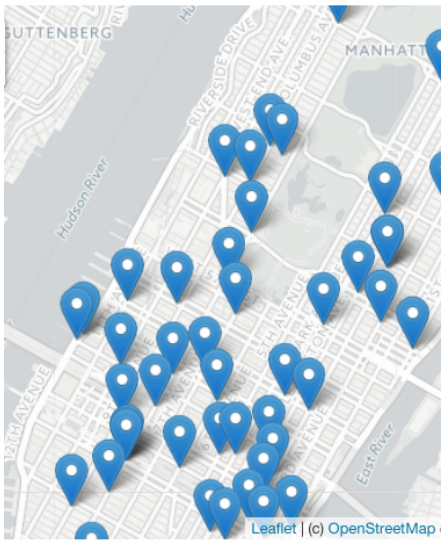


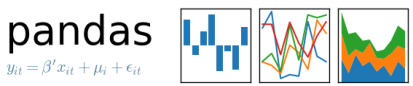


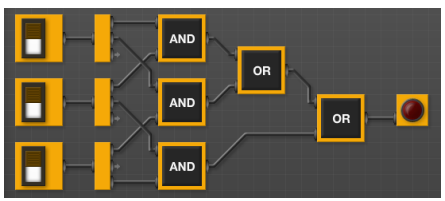


- **This course assumes no previous programming experience.**
- "... Emphasis on problem-solving and computational thinking through 'coding': computer programming for beginners..."
- Organized like a fugue, with variations on this theme:
  - Introduce coding constructs in Python,
  - Apply those ideas to different problems (e.g. analyzing & mapping data),
  - See constructs again:
    - ★ for logical circuits,

Syllabus: Topics

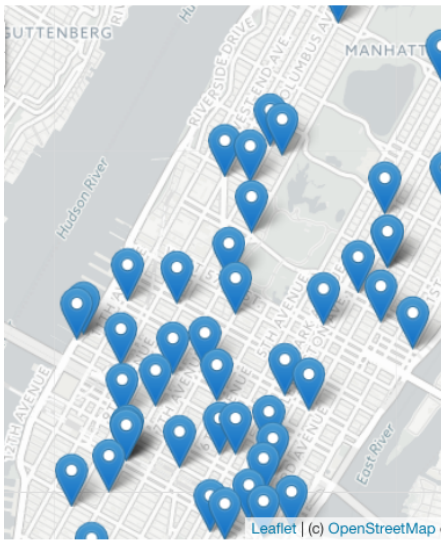


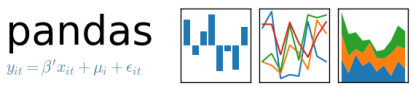


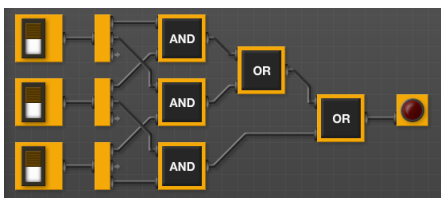


- **This course assumes no previous programming experience.**
- "... Emphasis on problem-solving and computational thinking through 'coding': computer programming for beginners..."
- Organized like a fugue, with variations on this theme:
  - Introduce coding constructs in Python,
  - Apply those ideas to different problems (e.g. analyzing & mapping data),
  - See constructs again:
    - ★ for logical circuits,
    - ★ for Unix command line interface,

Syllabus: Topics

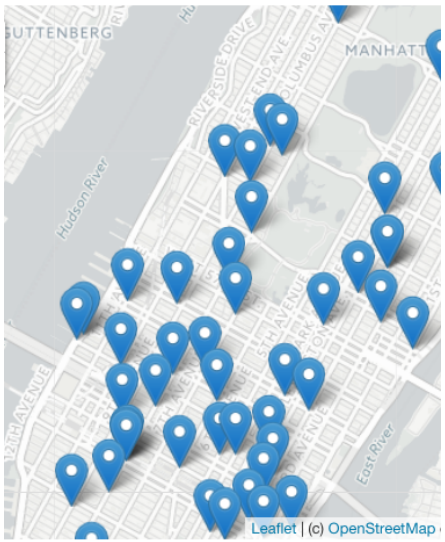






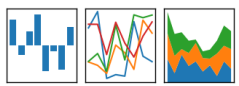
- **This course assumes no previous programming experience.**
- "... Emphasis on problem-solving and computational thinking through 'coding': computer programming for beginners..."
- Organized like a fugue, with variations on this theme:
  - Introduce coding constructs in Python,
  - Apply those ideas to different problems (e.g. analyzing & mapping data),
  - See constructs again:
    - ★ for logical circuits,
    - ★ for Unix command line interface,
    - ★ for the markup language for github,

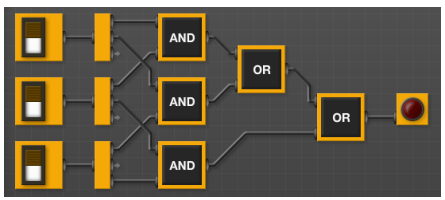
Syllabus: Topics



pandas

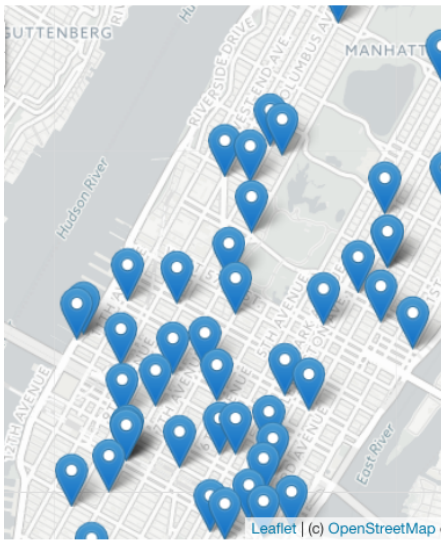
$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$

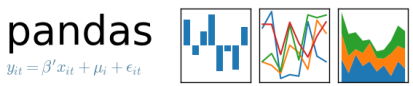


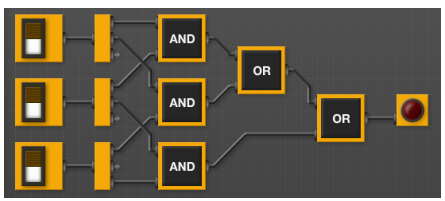


- **This course assumes no previous programming experience.**
- "... Emphasis on problem-solving and computational thinking through 'coding': computer programming for beginners..."
- Organized like a fugue, with variations on this theme:
  - Introduce coding constructs in Python,
  - Apply those ideas to different problems (e.g. analyzing & mapping data),
  - See constructs again:
    - ★ for logical circuits,
    - ★ for Unix command line interface,
    - ★ for the markup language for github,
    - ★ for the simplified machine language, &

Syllabus: Topics





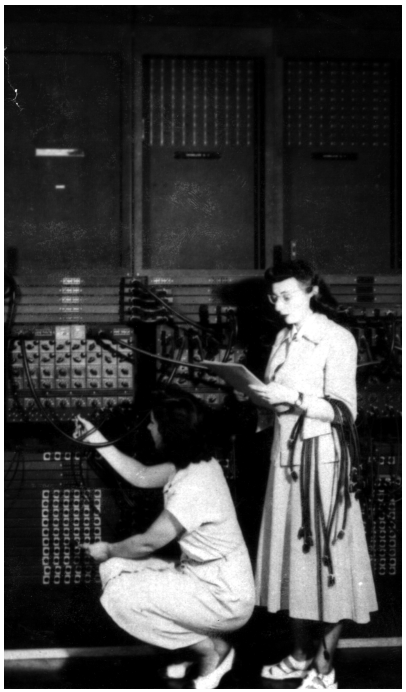


- **This course assumes no previous programming experience.**
- "... Emphasis on problem-solving and computational thinking through 'coding': computer programming for beginners..."
- Organized like a fugue, with variations on this theme:
  - Introduce coding constructs in Python,
  - Apply those ideas to different problems (e.g. analyzing & mapping data),
  - See constructs again:
    - ★ for logical circuits,
    - ★ for Unix command line interface,
    - ★ for the markup language for github,
    - ★ for the simplified machine language, &
    - ★ for C++.

Class Structure

Lecture:

- Tuesdays, 11:10am-12:25pm, 118 North.



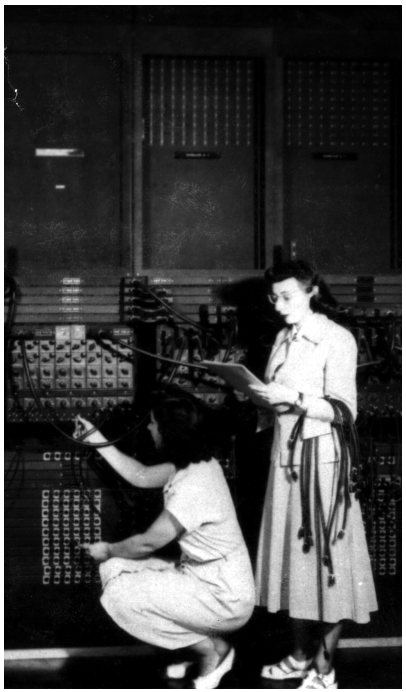
First "computers"

ENIAC, 1945.

Class Structure

Lecture:

- Tuesdays, 11:10am-12:25pm, 118 North.
- Lecture Preview: on-line prior to each lecture.



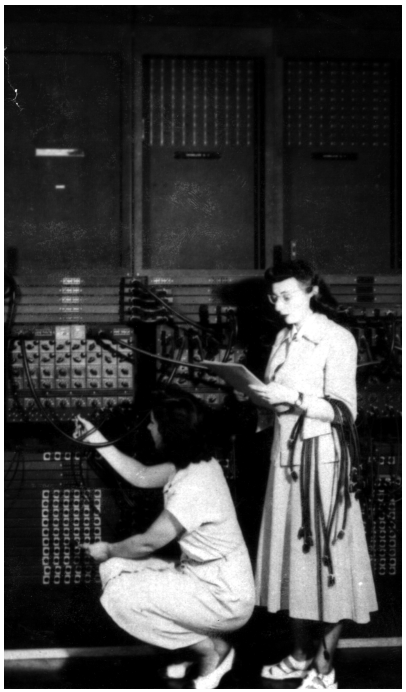
First "computers"

ENIAC, 1945.

Class Structure

Lecture:

- Tuesdays, 11:10am-12:25pm, 118 North.
- Lecture Preview: on-line prior to each lecture.
- Lecture Slips: only help your grade, final exam replaces incomplete or missing slips.



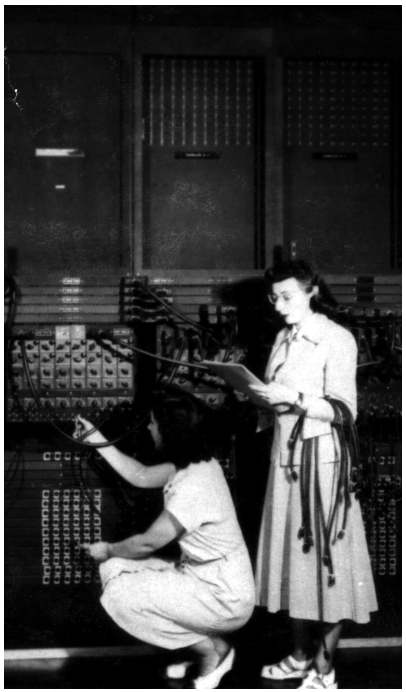
First "computers"

ENIAC, 1945.

Class Structure

Lecture:

- Tuesdays, 11:10am-12:25pm, 118 North.
- Lecture Preview: on-line prior to each lecture.
- Lecture Slips: only help your grade, final exam replaces incomplete or missing slips.
- Mix of explanation, challenges, & group work.



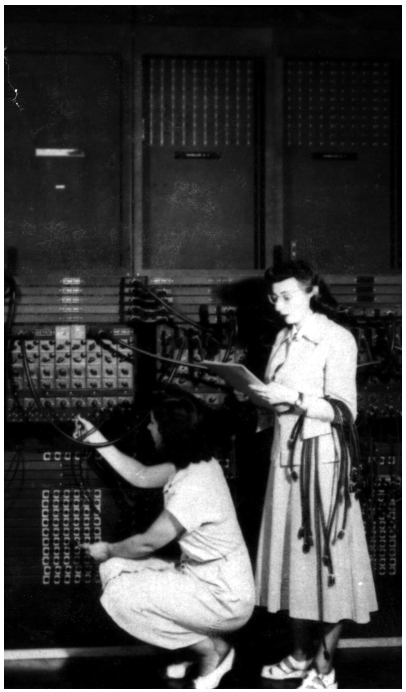
First "computers"

ENIAC, 1945.

Class Structure

Lecture:

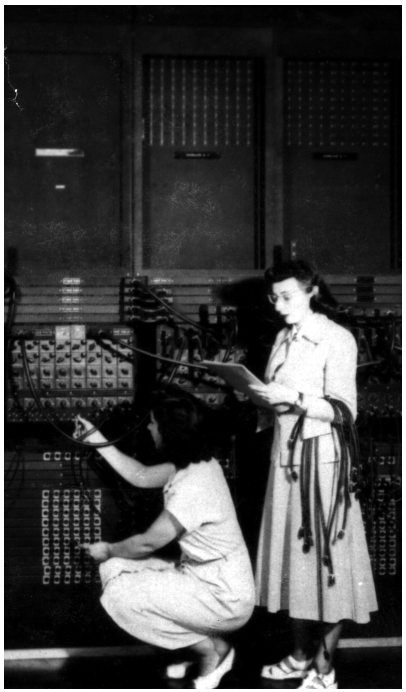
- Tuesdays, 11:10am-12:25pm, 118 North.
- Lecture Preview: on-line prior to each lecture.
- Lecture Slips: only help your grade, final exam replaces incomplete or missing slips.
- Mix of explanation, challenges, & group work.
- Hard to ask questions in a large lecture, ask UTAs & instructors during group work.



First "computers"

ENIAC, 1945.

Class Structure



First "computers"

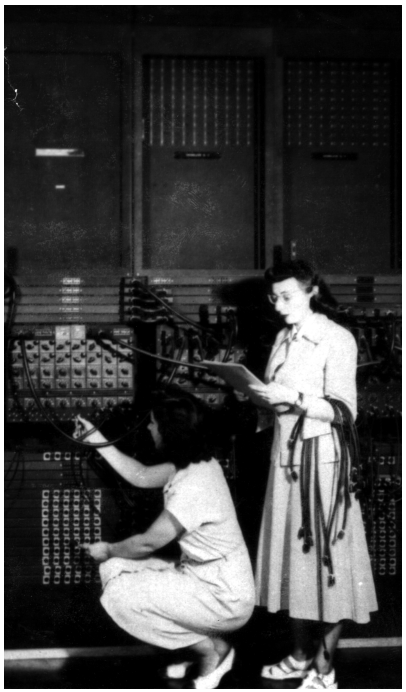
ENIAC, 1945.

Lecture:

- Tuesdays, 11:10am-12:25pm, 118 North.
- Lecture Preview: on-line prior to each lecture.
- Lecture Slips: only help your grade, final exam replaces incomplete or missing slips.
- Mix of explanation, challenges, & group work.
- Hard to ask questions in a large lecture, ask UTAs & instructors during group work.

Recitation Section:

Class Structure



First "computers"

ENIAC, 1945.

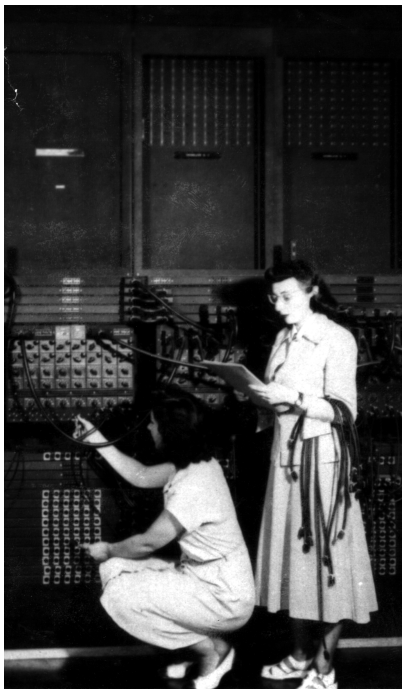
Lecture:

- Tuesdays, 11:10am-12:25pm, 118 North.
- Lecture Preview: on-line prior to each lecture.
- Lecture Slips: only help your grade, final exam replaces incomplete or missing slips.
- Mix of explanation, challenges, & group work.
- Hard to ask questions in a large lecture, ask UTAs & instructors during group work.

Recitation Section:

- Quiz: final exam replaces low/missing quizzes.

Class Structure



First "computers"

ENIAC, 1945.

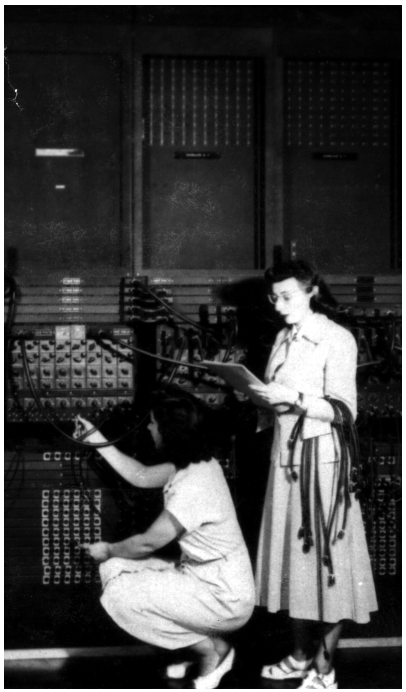
Lecture:

- Tuesdays, 11:10am-12:25pm, 118 North.
- Lecture Preview: on-line prior to each lecture.
- Lecture Slips: only help your grade, final exam replaces incomplete or missing slips.
- Mix of explanation, challenges, & group work.
- Hard to ask questions in a large lecture, ask UTAs & instructors during group work.

Recitation Section:

- Quiz: final exam replaces low/missing quizzes.
- Brief overview of lab & programs for the week.

Class Structure



First "computers"

ENIAC, 1945.

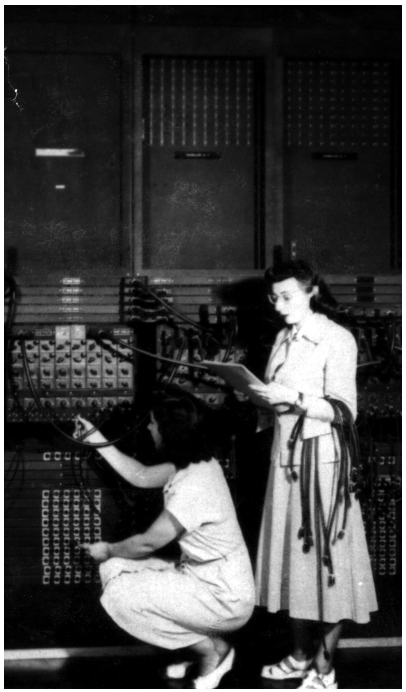
Lecture:

- Tuesdays, 11:10am-12:25pm, 118 North.
- Lecture Preview: on-line prior to each lecture.
- Lecture Slips: only help your grade, final exam replaces incomplete or missing slips.
- Mix of explanation, challenges, & group work.
- Hard to ask questions in a large lecture, ask UTAs & instructors during group work.

Recitation Section:

- Quiz: final exam replaces low/missing quizzes.
- Brief overview of lab & programs for the week.
- One-on-one help with instructors & UTAs.

Class Structure



First "computers"

ENIAC, 1945.

Lecture:

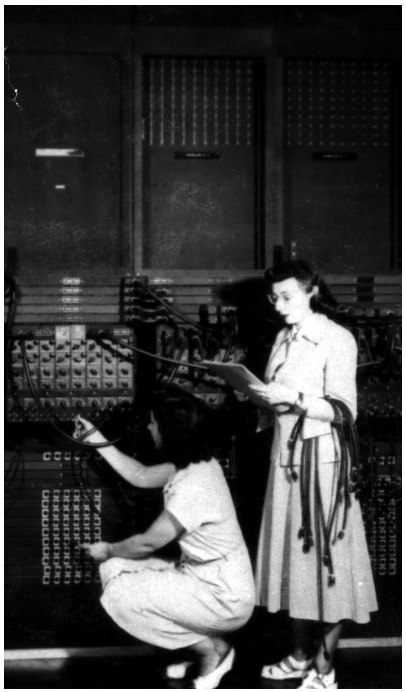
- Tuesdays, 11:10am-12:25pm, 118 North.
- Lecture Preview: on-line prior to each lecture.
- Lecture Slips: only help your grade, final exam replaces incomplete or missing slips.
- Mix of explanation, challenges, & group work.
- Hard to ask questions in a large lecture, ask UTAs & instructors during group work.

Recitation Section:

- Quiz: final exam replaces low/missing quizzes.
- Brief overview of lab & programs for the week.
- One-on-one help with instructors & UTAs.

Software Platforms:

Class Structure



First "computers"

ENIAC, 1945.

Lecture:

- Tuesdays, 11:10am-12:25pm, 118 North.
- Lecture Preview: on-line prior to each lecture.
- Lecture Slips: only help your grade, final exam replaces incomplete or missing slips.
- Mix of explanation, challenges, & group work.
- Hard to ask questions in a large lecture, ask UTAs & instructors during group work.

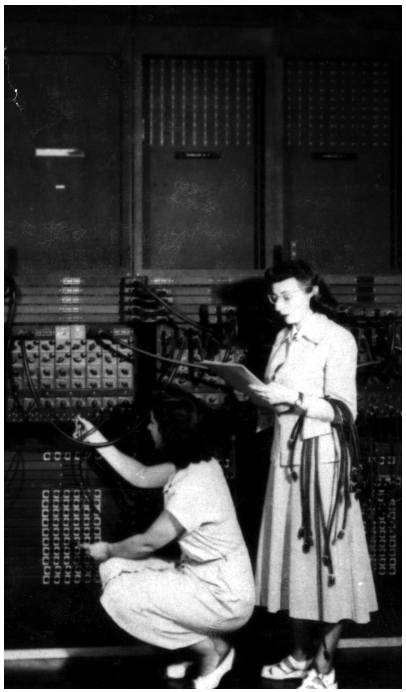
Recitation Section:

- Quiz: final exam replaces low/missing quizzes.
- Brief overview of lab & programs for the week.
- One-on-one help with instructors & UTAs.

Software Platforms:

- Blackboard: visit ICIT for access issues.

Class Structure



First "computers"

ENIAC, 1945.

Lecture:

- Tuesdays, 11:10am-12:25pm, 118 North.
- Lecture Preview: on-line prior to each lecture.
- Lecture Slips: only help your grade, final exam replaces incomplete or missing slips.
- Mix of explanation, challenges, & group work.
- Hard to ask questions in a large lecture, ask UTAs & instructors during group work.

Recitation Section:

- Quiz: final exam replaces low/missing quizzes.
- Brief overview of lab & programs for the week.
- One-on-one help with instructors & UTAs.

Software Platforms:

- Blackboard: visit ICIT for access issues.
- Gradescope: email invite sent Wednesday.

Philosophy (Or Why We Do What We Do)

Grading:

- Do you curve grades?

Philosophy (Or Why We Do What We Do)

Grading:

- Do you curve grades?
  No, we grade on your mastery of the material and do not have a set number of A's, B's, C's that we curve grades to match (i.e. your demonstrated mastery over your relative performance to the class).

Philosophy (Or Why We Do What We Do)

Grading:

- Do you curve grades?
  No, we grade on your mastery of the material and do not have a set number of A's, B's, C's that we curve grades to match (i.e. your demonstrated mastery over your relative performance to the class).
- Okay, then could everyone get an A?

Philosophy (Or Why We Do What We Do)

Grading:

- Do you curve grades?
  No, we grade on your mastery of the material and do not have a set number of A's, B's, C's that we curve grades to match (i.e. your demonstrated mastery over your relative performance to the class).
- Okay, then could everyone get an A?
  Yes, we are not a filter course. Our goal is for students to succeed in the course and in the CS major and minor.

Philosophy (Or Why We Do What We Do)

Grading:

- Do you curve grades?
  No, we grade on your mastery of the material and do not have a set number of A's, B's, C's that we curve grades to match (i.e. your demonstrated mastery over your relative performance to the class).
- Okay, then could everyone get an A?
  Yes, we are not a filter course. Our goal is for students to succeed in the course and in the CS major and minor.
- What happens to my grade if I miss a lecture or quiz?

Philosophy (Or Why We Do What We Do)

Grading:

- Do you curve grades?
  No, we grade on your mastery of the material and do not have a set number of A's, B's, C's that we curve grades to match (i.e. your demonstrated mastery over your relative performance to the class).
- Okay, then could everyone get an A?
  Yes, we are not a filter course. Our goal is for students to succeed in the course and in the CS major and minor.
- What happens to my grade if I miss a lecture or quiz?
  We replace missing or low grades on lecture slips or quizzes with your final exam grade.
  Lecture slips and quizzes only help your grade.

Philosophy (Or Why We Do What We Do)

Course Structure:

- Why 60 programs assignments? My friend only has to do 10.

Philosophy (Or Why We Do What We Do)

Course Structure:

- Why 60 programs assignments? My friend only has to do 10.
  Traditionally, it's 10 long 'all-nighters' assignments.

Philosophy (Or Why We Do What We Do)

Course Structure:

- Why 60 programs assignments? My friend only has to do 10.
  Traditionally, it's 10 long 'all-nighters' assignments.
  While this mimics some jobs, students master skills better with smaller, challenges.

Philosophy (Or Why We Do What We Do)

Course Structure:

- Why 60 programs assignments? My friend only has to do 10.
  Traditionally, it's 10 long 'all-nighters' assignments.
  While this mimics some jobs, students master skills better with smaller, challenges.
- Why weekly quizzes instead of midterms?

Philosophy (Or Why We Do What We Do)

Course Structure:

- Why 60 programs assignments? My friend only has to do 10.
  Traditionally, it's 10 long 'all-nighters' assignments.
  While this mimics some jobs, students master skills better with smaller, challenges.
- Why weekly quizzes instead of midterms?
  Weekly quizzes increase pass rates and mastery of material.

Philosophy (Or Why We Do What We Do)

Course Structure:

- Why 60 programs assignments? My friend only has to do 10.
  Traditionally, it's 10 long 'all-nighters' assignments.
  While this mimics some jobs, students master skills better with smaller, challenges.
- Why weekly quizzes instead of midterms?
  Weekly quizzes increase pass rates and mastery of material.
  Actively using knowledge increases your brain's ability to retain knowledge.

Philosophy (Or Why We Do What We Do)

Course Structure:

- Why 60 programs assignments? My friend only has to do 10.
  Traditionally, it's 10 long 'all-nighters' assignments.
  While this mimics some jobs, students master skills better with smaller, challenges.
- Why weekly quizzes instead of midterms?
  Weekly quizzes increase pass rates and mastery of material.
  Actively using knowledge increases your brain's ability to retain knowledge.
- Why pre-testing? Why do we get asked (ungraded) questions on stuff we have never seen before?

Philosophy (Or Why We Do What We Do)

Course Structure:

- Why 60 programs assignments? My friend only has to do 10.
  Traditionally, it's 10 long 'all-nighters' assignments.
  While this mimics some jobs, students master skills better with smaller, challenges.
- Why weekly quizzes instead of midterms?
  Weekly quizzes increase pass rates and mastery of material.
  Actively using knowledge increases your brain's ability to retain knowledge.
- Why pre-testing? Why do we get asked (ungraded) questions on stuff we have never seen before?
  While counter-intuitive, it gives a "mental scaffold" to store new material.

Philosophy (Or Why We Do What We Do)

Course Structure:

- Why 60 programs assignments? My friend only has to do 10.
  Traditionally, it's 10 long 'all-nighters' assignments.
  While this mimics some jobs, students master skills better with smaller, challenges.
- Why weekly quizzes instead of midterms?
  Weekly quizzes increase pass rates and mastery of material.
  Actively using knowledge increases your brain's ability to retain knowledge.
- Why pre-testing? Why do we get asked (ungraded) questions on stuff we have never seen before?
  While counter-intuitive, it gives a "mental scaffold" to store new material.
- I like working by myself. Why do I have to work in groups during class?

Philosophy (Or Why We Do What We Do)

Course Structure:

- Why 60 programs assignments? My friend only has to do 10.
  Traditionally, it's 10 long 'all-nighters' assignments.
  While this mimics some jobs, students master skills better with smaller, challenges.
- Why weekly quizzes instead of midterms?
  Weekly quizzes increase pass rates and mastery of material.
  Actively using knowledge increases your brain's ability to retain knowledge.
- Why pre-testing? Why do we get asked (ungraded) questions on stuff we have never seen before?
  While counter-intuitive, it gives a "mental scaffold" to store new material.
- I like working by myself. Why do I have to work in groups during class?
  Active learning increases student performance.

Philosophy (Or Why We Do What We Do)

Course Structure:

- Why 60 programs assignments? My friend only has to do 10.
  Traditionally, it's 10 long 'all-nighters' assignments.
  While this mimics some jobs, students master skills better with smaller, challenges.
- Why weekly quizzes instead of midterms?
  Weekly quizzes increase pass rates and mastery of material.
  Actively using knowledge increases your brain's ability to retain knowledge.
- Why pre-testing? Why do we get asked (ungraded) questions on stuff we have never seen before?
  While counter-intuitive, it gives a "mental scaffold" to store new material.
- I like working by myself. Why do I have to work in groups during class?
  Active learning increases student performance.
  Also, provides excellent practice explaining technical ideas (i.e. tech interviews).

Philosophy (Or Why We Do What We Do)

Help:

- What's the best way to study for this course?

Philosophy (Or Why We Do What We Do)

Help:

- What's the best way to study for this course?
  - *Most efficient way: do the programs*

Philosophy (Or Why We Do What We Do)

Help:

- What's the best way to study for this course?
  - *Most efficient way: do the programs (more programs on old finals).*

Philosophy (Or Why We Do What We Do)

Help:

- What's the best way to study for this course?
  - *Most efficient way: do the programs (more programs on old finals).*
  - *Aim to complete when related ideas covered in lab.*

Philosophy (Or Why We Do What We Do)

Help:

- What's the best way to study for this course?
  - *Most efficient way: do the programs (more programs on old finals).*
  - *Aim to complete when related ideas covered in lab.*
  - *If stuck, there's reading and tutorials on webpage.*

Philosophy (Or Why We Do What We Do)

Help:

- What's the best way to study for this course?
  - *Most efficient way: do the programs (more programs on old finals).*
  - *Aim to complete when related ideas covered in lab.*
  - *If stuck, there's reading and tutorials on webpage.*
- I'm stuck on a program. Where can I get help?

Philosophy (Or Why We Do What We Do)

Help:

- What's the best way to study for this course?
  - *Most efficient way: do the programs (more programs on old finals).*
  - *Aim to complete when related ideas covered in lab.*
  - *If stuck, there's reading and tutorials on webpage.*
- I'm stuck on a program. Where can I get help?
  - *Tutoring hours when classes are in session:*

Philosophy (Or Why We Do What We Do)

Help:

- What's the best way to study for this course?
  - *Most efficient way: do the programs (more programs on old finals).*
  - *Aim to complete when related ideas covered in lab.*
  - *If stuck, there's reading and tutorials on webpage.*
- I'm stuck on a program. Where can I get help?
  - *Tutoring hours when classes are in session:*
    - ⋆ *Mondays-Fridays, 9:30am-6:30pm*

Philosophy (Or Why We Do What We Do)

Help:

- What's the best way to study for this course?
  - *Most efficient way: do the programs (more programs on old finals).*
  - *Aim to complete when related ideas covered in lab.*
  - *If stuck, there's reading and tutorials on webpage.*
- I'm stuck on a program. Where can I get help?
  - *Tutoring hours when classes are in session:*
    - ⋆ *Mondays-Fridays, 9:30am-6:30pm*
  - *On-line help: email questions to:*

Philosophy (Or Why We Do What We Do)

Help:

- What's the best way to study for this course?
  - *Most efficient way: do the programs (more programs on old finals).*
  - *Aim to complete when related ideas covered in lab.*
  - *If stuck, there's reading and tutorials on webpage.*
- I'm stuck on a program. Where can I get help?
  - *Tutoring hours when classes are in session:*
    - ★ *Mondays-Fridays, 9:30am-6:30pm*
  - *On-line help: email questions to:*
    - ★ `huntercsci127help@gmail.com`

Philosophy (Or Why We Do What We Do)

Advising:

- Do you coordinate with advisors?

Philosophy (Or Why We Do What We Do)

Advising:

- Do you coordinate with advisors?
  Yes! We work closely with the computer science advisors!

Philosophy (Or Why We Do What We Do)

Advising:

- Do you coordinate with advisors?
  Yes! We work closely with the computer science advisors!

Amanda Bell
Pre-majors & Early Majors

Justin Tojeira
Internships & Upper Division

Introductions: Your Turn

- Introduce yourself to two classmates (that you have not met before).
- Write down names & interesting fact on lecture slip.

Today's Topics

- Introduction to Python
- Definite Loops (`for`-loops)
- Turtle Graphics
- Algorithms

Introduction to Python

- We will be writing programs– commands to the computer to do something.

Introduction to Python

- We will be writing programs– commands to the computer to do something.
- A **programming language** is a stylized way of writing those commands.

Introduction to Python

- We will be writing programs– commands to the computer to do something.
- A **programming language** is a stylized way of writing those commands.
- If you can write a logical argument or persuasive essay, you can write a program.

Introduction to Python

- We will be writing programs– commands to the computer to do something.
- A **programming language** is a stylized way of writing those commands.
- If you can write a logical argument or persuasive essay, you can write a program.
- Our first language, Python, is popular for its ease-of-use, flexibility, and extendibility.

Introduction to Python

- We will be writing programs– commands to the computer to do something.
- A **programming language** is a stylized way of writing those commands.
- If you can write a logical argument or persuasive essay, you can write a program.
- Our first language, Python, is popular for its ease-of-use, flexibility, and extendibility.
- The first lab goes into step-by-step details of getting Python running.

Introduction to Python

- We will be writing programs– commands to the computer to do something.
- A **programming language** is a stylized way of writing those commands.
- If you can write a logical argument or persuasive essay, you can write a program.
- Our first language, Python, is popular for its ease-of-use, flexibility, and extendibility.
- The first lab goes into step-by-step details of getting Python running.
- We'll look at the design and basic structure (no worries if you haven't tried it yet in lab).

First Program: `Hello, World!`

Demo in `pythonTutor`

First Program: Hello, World!

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

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

First Program: Hello, World!

```
#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)
```

First Program: Hello, World!

```
#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
```

First Program: `Hello, World!`

```
#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
```

- Output to the screen is: `Hello, World!`

First Program: `Hello, World!`

```
#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
```

- Output to the screen is: `Hello, World!`
- Can replace `Hello, World!` with another string to be printed.

Variations on Hello, World!

```
#Name:  L-M Miranda
#Date:  Hunter College HS '98
#This program prints intro lyrics

print('Get your education,')
```

Variations on Hello, World!

```
#Name:  L-M Miranda
#Date:  Hunter College HS '98
#This program prints intro lyrics

print('Get your education,')
```

Who is L-M Miranda?

Variations on Hello, World!

```
#Name:  L-M Miranda
#Date:  Hunter College HS '98
#This program prints intro lyrics

print('Get your education,')
```

Who is L-M Miranda?

Variations on Hello, World!

```
#Name:  L-M Miranda
#Date:  Hunter College HS '98
#This program prints intro lyrics

print('Get your education,')
```

Variations on Hello, World!

```
#Name:  L-M Miranda
#Date:  Hunter College HS '98
#This program prints intro lyrics

print('Get your education,')
print("don't forget from whence you came, and")
```

Variations on Hello, World!

```
#Name:  L-M Miranda
#Date:  Hunter College HS '98
#This program prints intro lyrics

print('Get your education,')
print("don't forget from whence you came, and")
print("The world's gonna know your name.")
```

Variations on Hello, World!

```
#Name:  L-M Miranda
#Date:  Hunter College HS '98
#This program prints intro lyrics

print('Get your education,')
print("don't forget from whence you came, and")
print("The world's gonna know your name.")
```

- Each print statement writes its output on a new line.

Variations on Hello, World!

```
#Name:  L-M Miranda
#Date:  Hunter College HS '98
#This program prints intro lyrics

print('Get your education,')
print("don't forget from whence you came, and")
print("The world's gonna know your name.")
```

- Each print statement writes its output on a new line.
- Results in three lines of output.

Variations on Hello, World!

```
#Name:  L-M Miranda
#Date:  Hunter College HS '98
#This program prints intro lyrics

print('Get your education,')
print("don't forget from whence you came, and")
print("The world's gonna know your name.")
```

- Each print statement writes its output on a new line.
- Results in three lines of output.
- Can use single or double quotes, just need to match.

Turtles Introduction

- A simple, whimsical graphics package for Python.

Turtles Introduction

- A simple, whimsical graphics package for Python.
- Dates back to Logos Turtles in the 1960s.

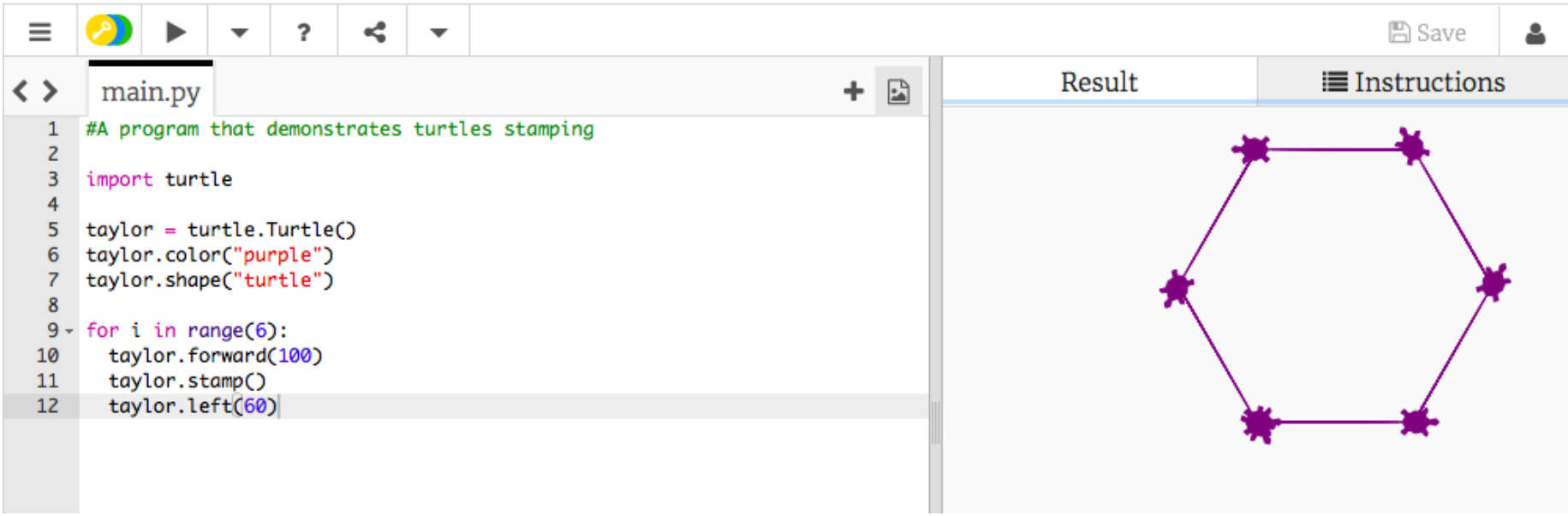
Turtles Introduction

- A simple, whimsical graphics package for Python.
- Dates back to Logos Turtles in the 1960s.
- (Demo from webpage)

Turtles Introduction

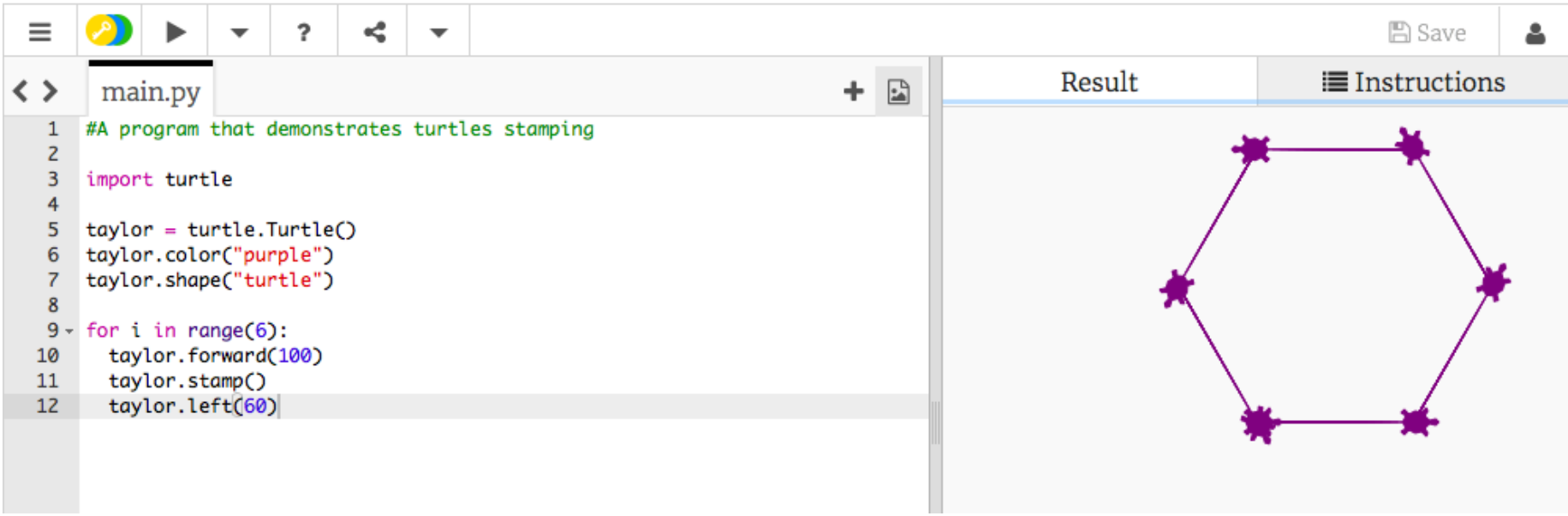
- A simple, whimsical graphics package for Python.
- Dates back to Logos Turtles in the 1960s.
- (Demo from webpage)
- (Fancier turtle demo)

Turtles Introduction



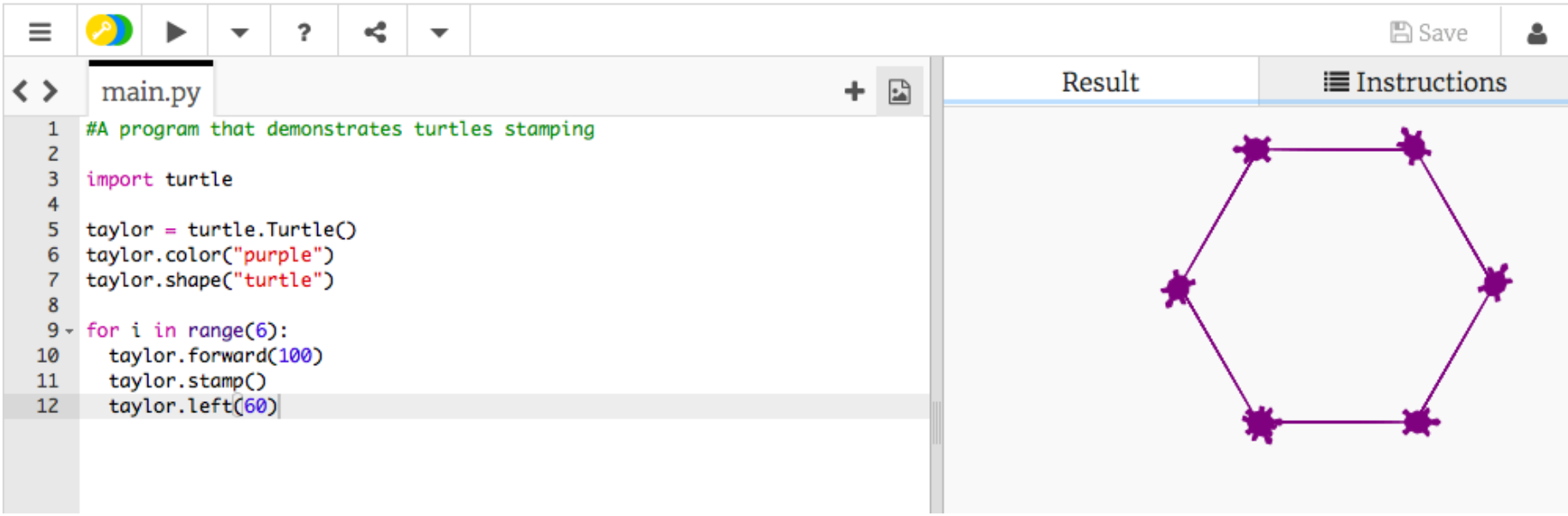
- Creates a turtle, called `taylor`

Turtles Introduction



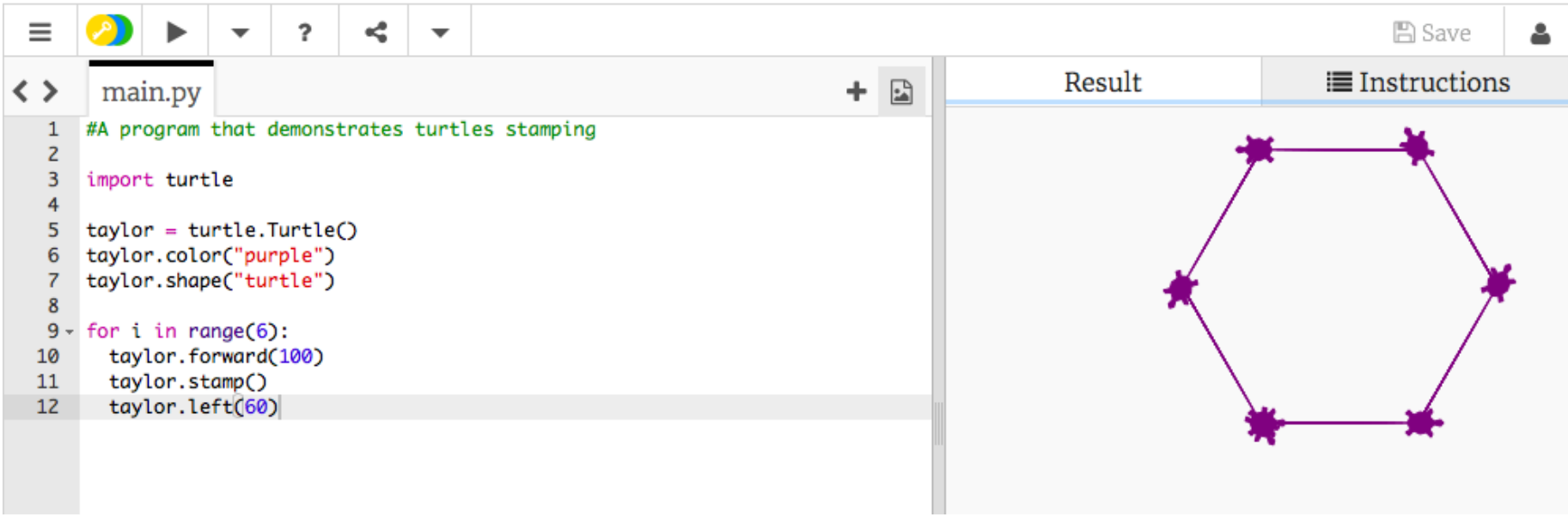
- Creates a turtle, called `taylor`
- Changes the color (to purple) and shape (to turtle-shaped)

Turtles Introduction



- Creates a turtle, called `taylor`
- Changes the color (to purple) and shape (to turtle-shaped)
- Repeats 6 times:

Turtles Introduction



- Creates a turtle, called `taylor`
- Changes the color (to purple) and shape (to turtle-shaped)
- Repeats 6 times:
  - Move forward; stamp; and turn left 60 degrees

Group Work

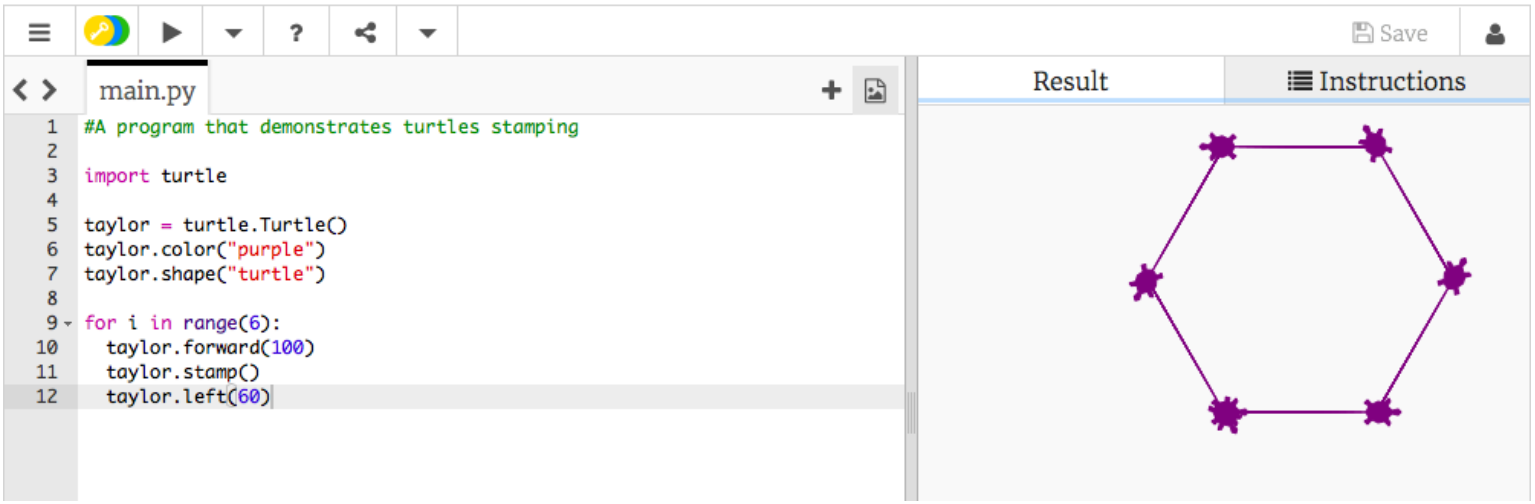
Working in pairs or triples:

1. Write a program that will draw a 10-sided polygon.
2. Write a program that will repeat the line:

   ```
   I'm lookin' for a mind at work!
   ```

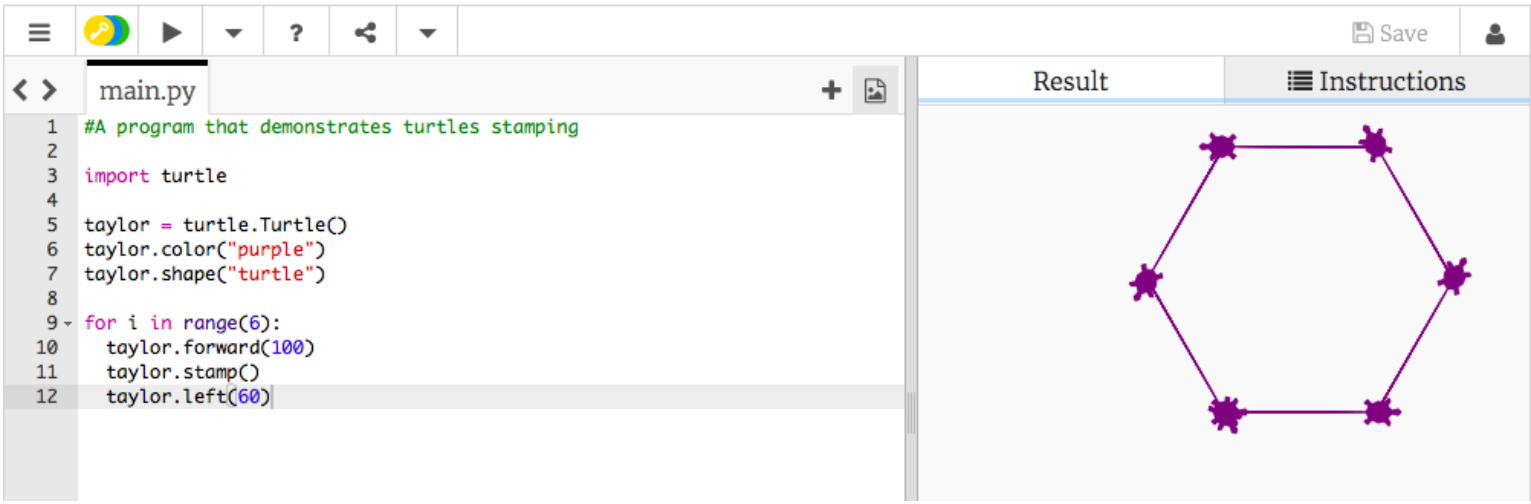
   three times.

Decagon Program



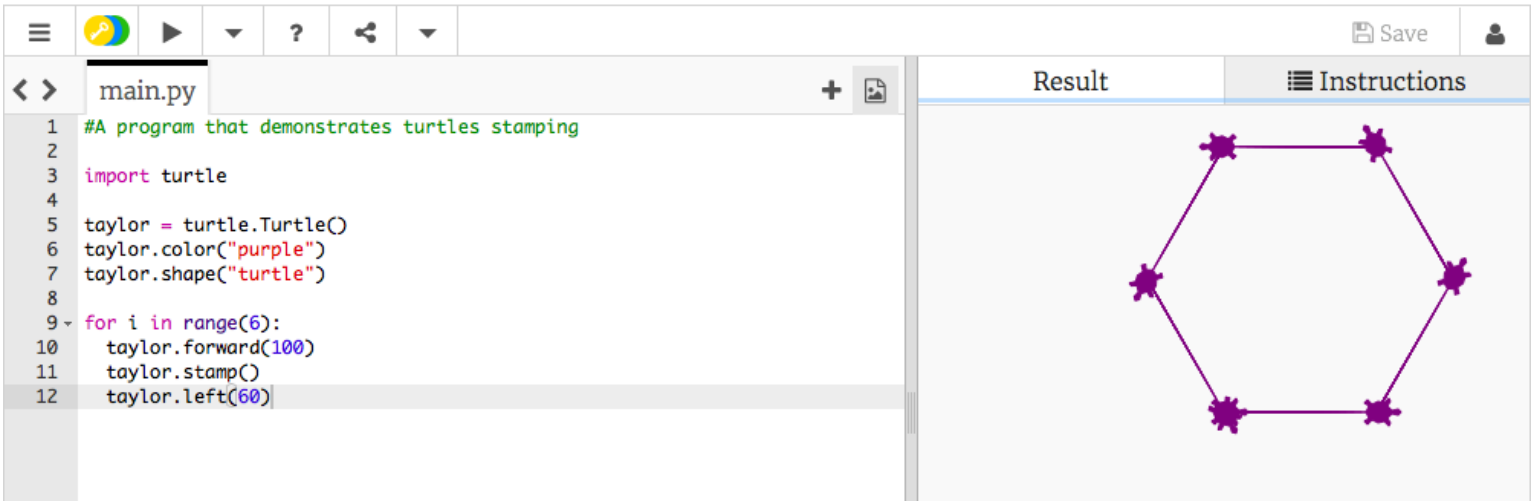
- Start with the hexagon program.

Decagon Program



- Start with the hexagon program.
- Has 10 sides (instead of 6), so change the `range(6)` to `range(10)`.

Decagon Program



- Start with the hexagon program.
- Has 10 sides (instead of 6), so change the `range(6)` to `range(10)`.
- Makes 10 turns (instead of 6),
  so change the `taylor.left(60)` to `taylor.left(360/10)`.

Work Program

2. Write a program that will repeat the line:

   ```
   I'm lookin' for a mind at work!
   ```

   three times.

Work Program

2. Write a program that will repeat the line:

   ```
   I'm lookin' for a mind at work!
   ```

   three times.

- Repeats three times, so, use `range(3)`:

  ```
  for i in range(3):
  ```

Work Program

2. Write a program that will repeat the line:

   ```
   I'm lookin' for a mind at work!
   ```

   three times.

- Repeats three times, so, use `range(3)`:

  ```
  for i in range(3):
  ```

- Instead of turtle commands, repeating a print statement.

Work Program

2. Write a program that will repeat the line:

   ```
   I'm lookin' for a mind at work!
   ```

   three times.

- Repeats three times, so, use `range(3)`:

  ```
  for i in range(3):
  ```

- Instead of turtle commands, repeating a print statement.
- Completed program:

  ```
  # Your name here!
  for i in range(3):
      print("I'm lookin' for a mind at work!")
  ```

What is an Algorithm?

From our textbook:

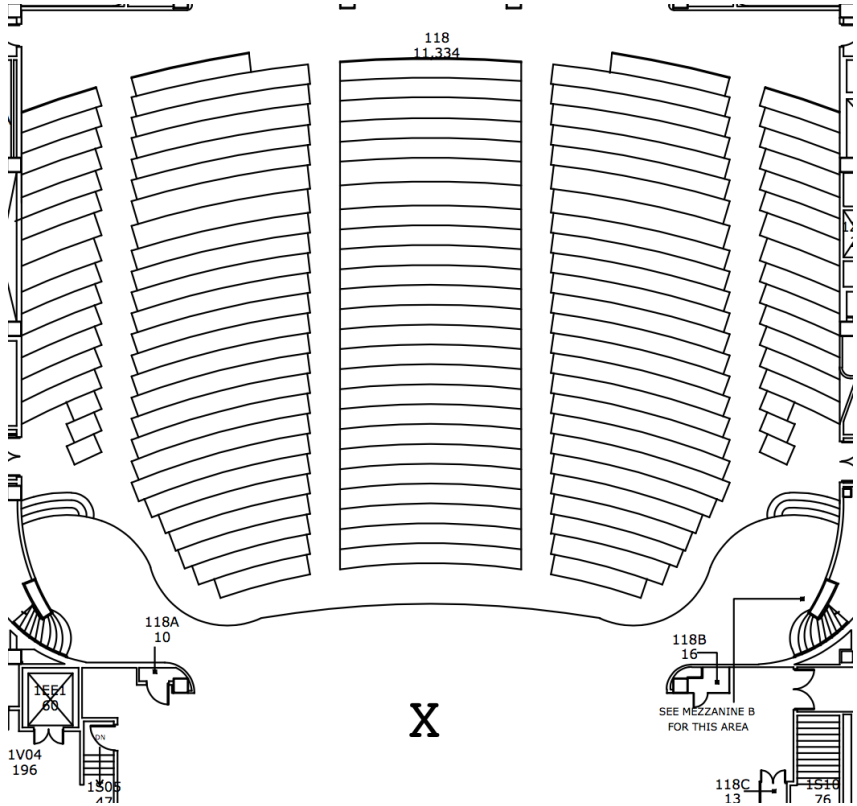
- An algorithm is a process or set of rules to be followed to solve a problem.

What is an Algorithm?

From our textbook:

- An algorithm is a process or set of rules to be followed to solve a problem.
- Programming is a skill that allows a computer scientist to take an algorithm and represent it in a notation (a program) that can be followed by a computer.

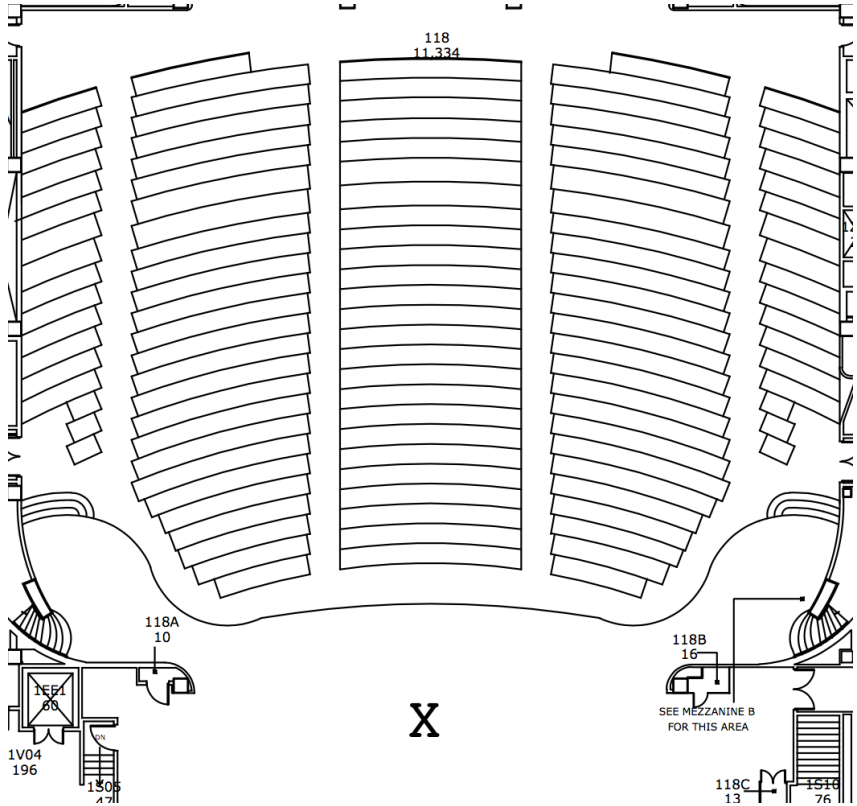
Group Work



Working in pairs or triples:

1. On the floorplan, mark your current location.
2. Write an algorithm (step-by-step directions) to get to `X`.

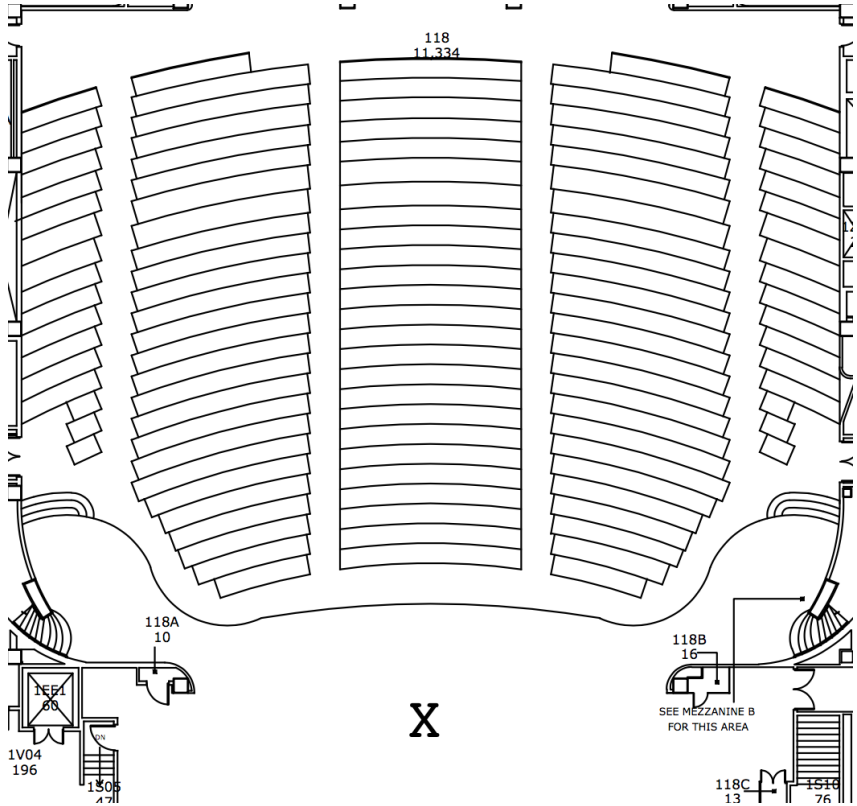
Group Work



Working in pairs or triples:

1. On the floorplan, mark your current location.
2. Write an algorithm (step-by-step directions) to get to `X`.
3. Basic Rules:

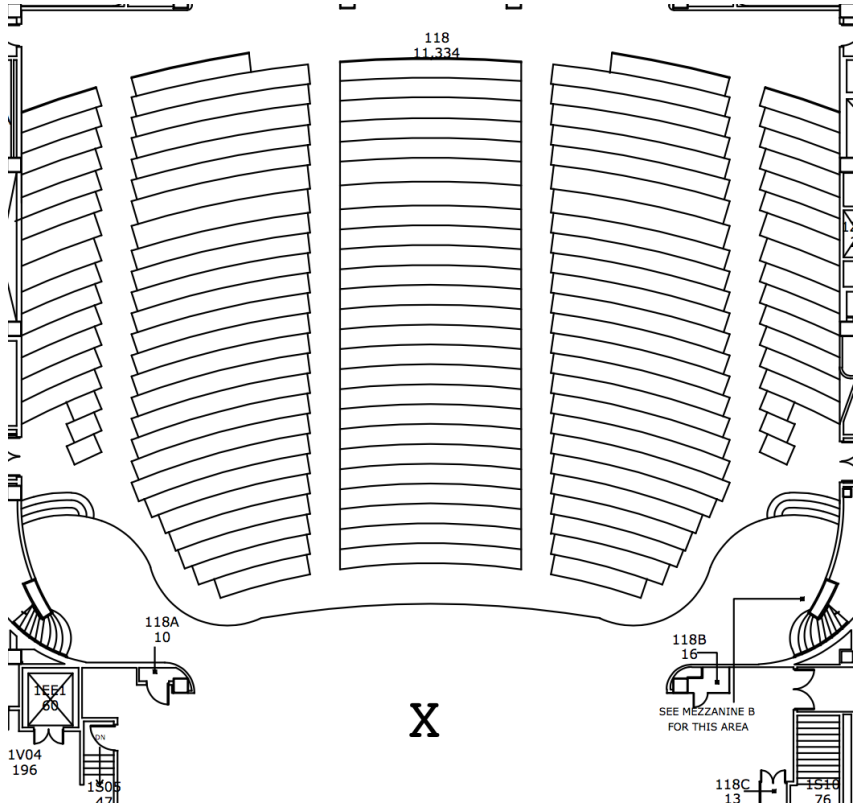
Group Work



Working in pairs or triples:

1. On the floorplan, mark your current location.
2. Write an algorithm (step-by-step directions) to get to `X`.
3. Basic Rules:
   - Use `turtle` commands.

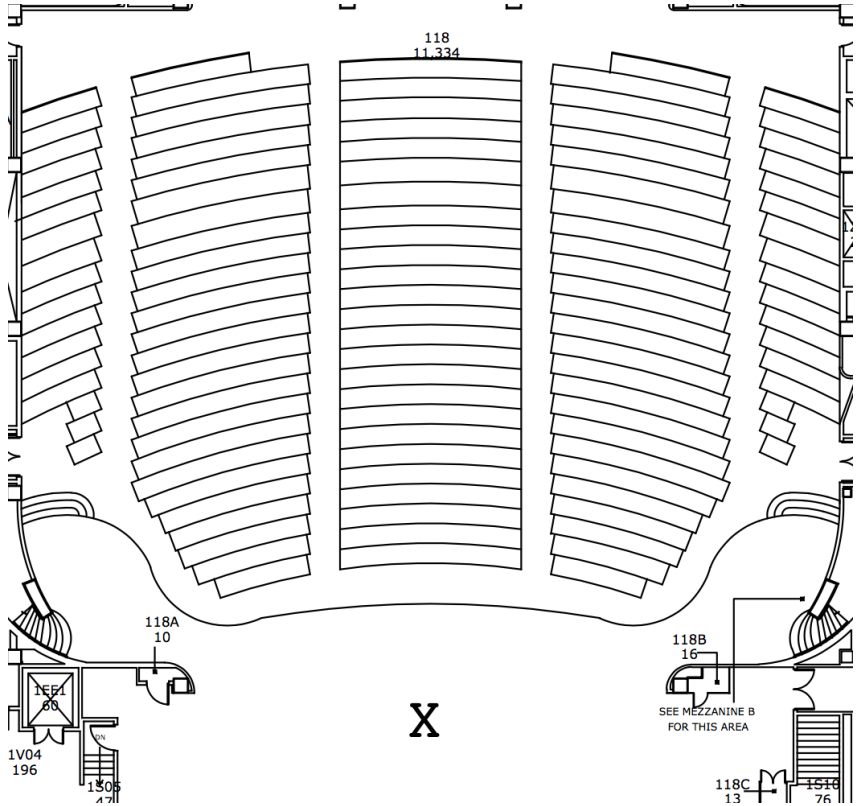
Group Work



Working in pairs or triples:

1. On the floorplan, mark your current location.
2. Write an algorithm (step-by-step directions) to get to `X`.
3. Basic Rules:
   - Use `turtle` commands.
   - Do not run turtles into walls, chairs, obstacles, etc.

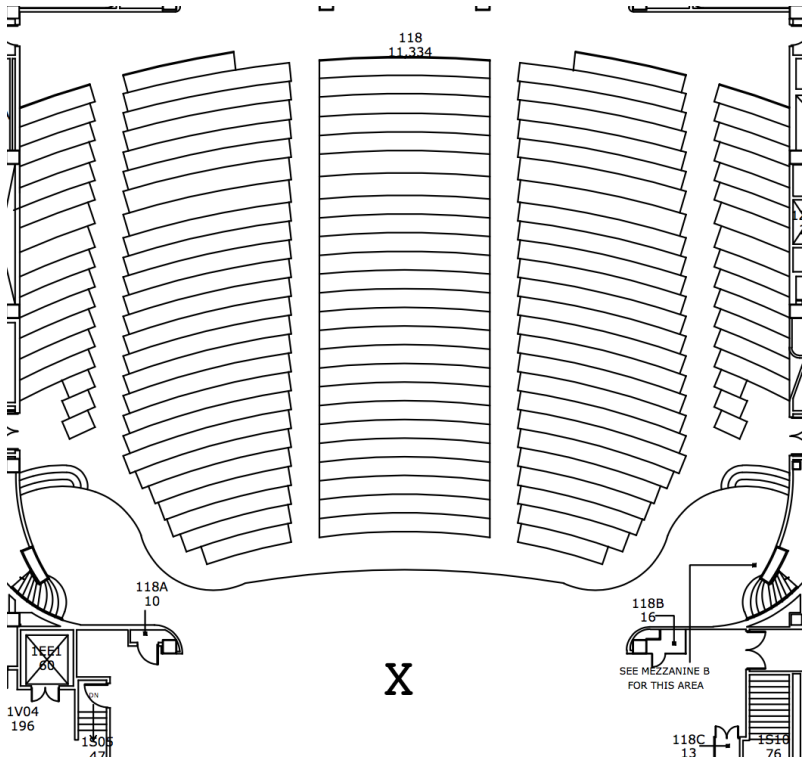
Group Work



Working in pairs or triples:

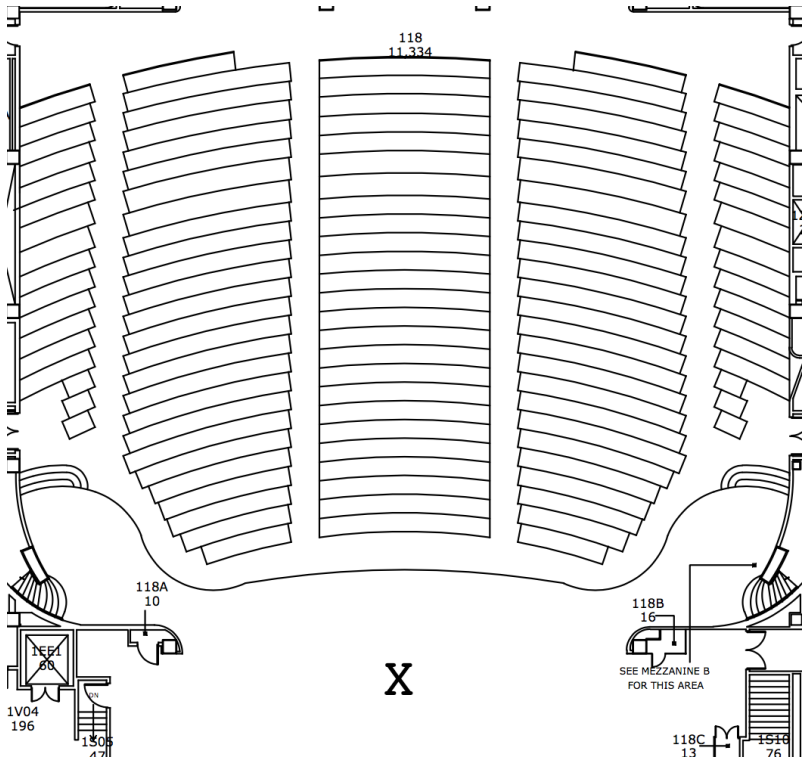
1. On the floorplan, mark your current location.
2. Write an algorithm (step-by-step directions) to get to `X`.
3. Basic Rules:
   - Use `turtle` commands.
   - Do not run turtles into walls, chairs, obstacles, etc.
   - Turtles cannot climb walls, must use stairs.

Group Work



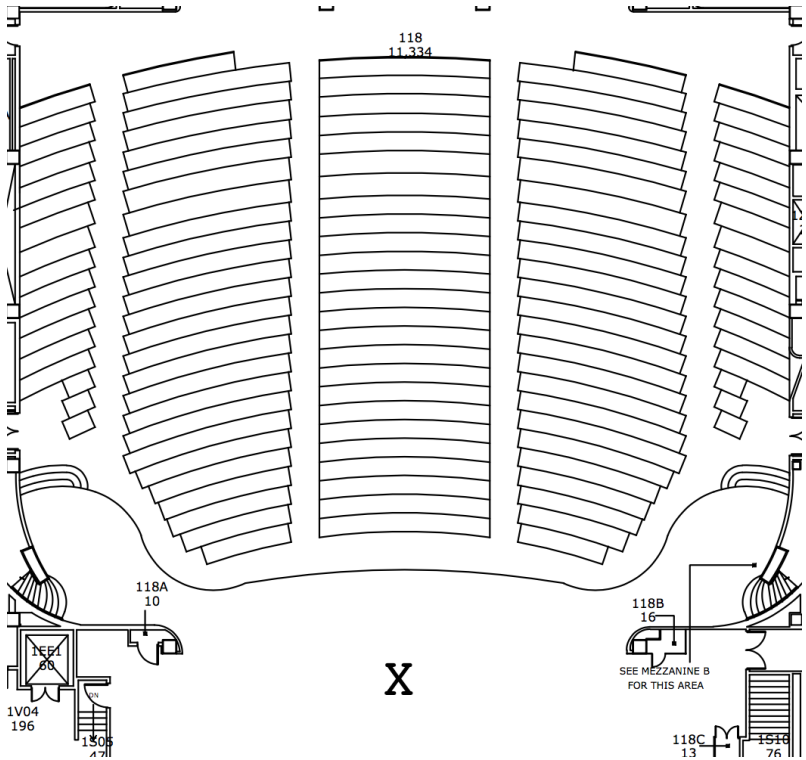
- Have one person in your group be the "turtle."

Group Work



- Have one person in your group be the "turtle."
- Follow the directions to get to `X`.

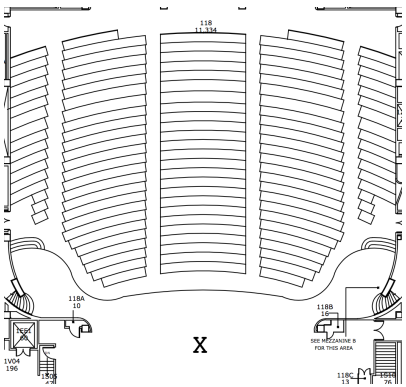
Group Work



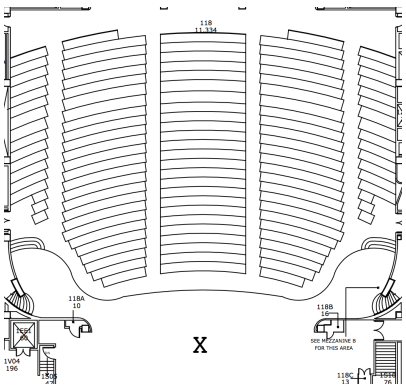
- Have one person in your group be the "turtle."
- Follow the directions to get to `X`.
- Annotate any changes needed to the directions (i.e. debug your work).

Recap

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

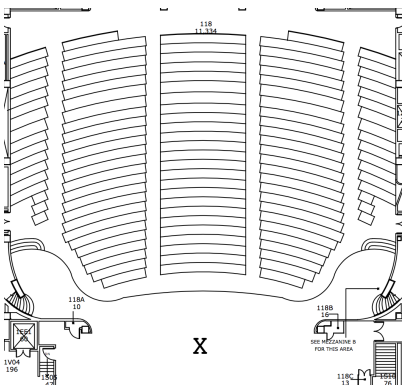


Recap



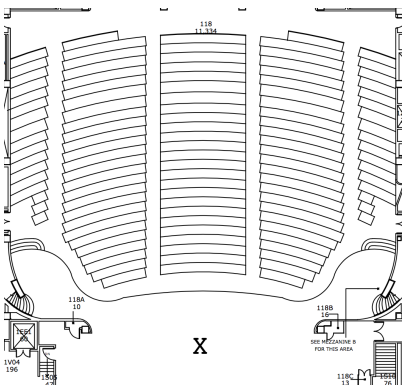
- On lecture slip, write down a topic you wish we had spent more time (and why).
- Writing precise algorithms is difficult.

Recap



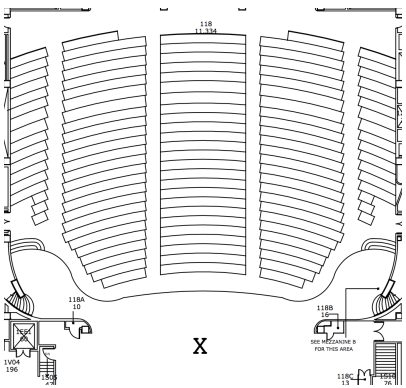
- On lecture slip, write down a topic you wish we had spent more time (and why).
- Writing precise algorithms is difficult.
- In Python, we introduced:

Recap



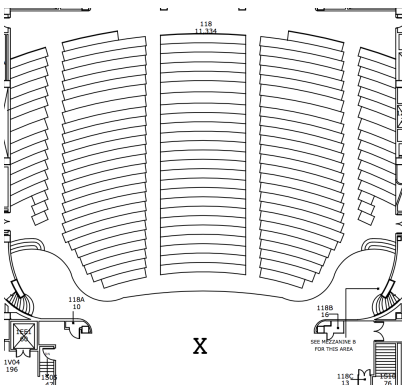
- On lecture slip, write down a topic you wish we had spent more time (and why).
- Writing precise algorithms is difficult.
- In Python, we introduced:
  - strings, or sequences of characters,

Recap



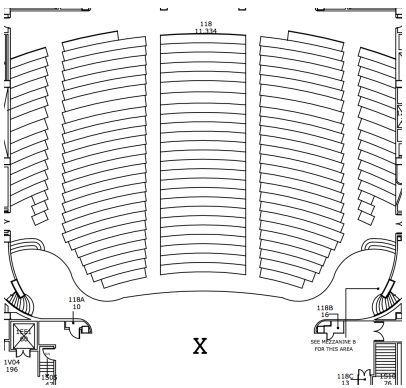
- On lecture slip, write down a topic you wish we had spent more time (and why).
- Writing precise algorithms is difficult.
- In Python, we introduced:
  - strings, or sequences of characters,
  - `print()` statements,

Recap



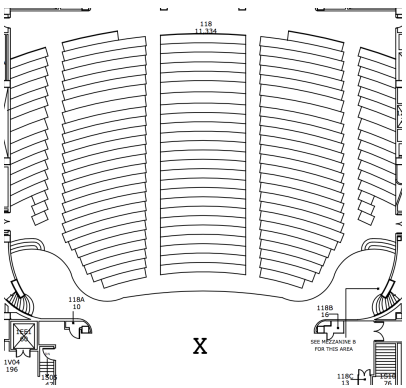
- On lecture slip, write down a topic you wish we had spent more time (and why).
- Writing precise algorithms is difficult.
- In Python, we introduced:
  - strings, or sequences of characters,
  - `print()` statements,
  - `for`-loops with `range()` statements, &

Recap



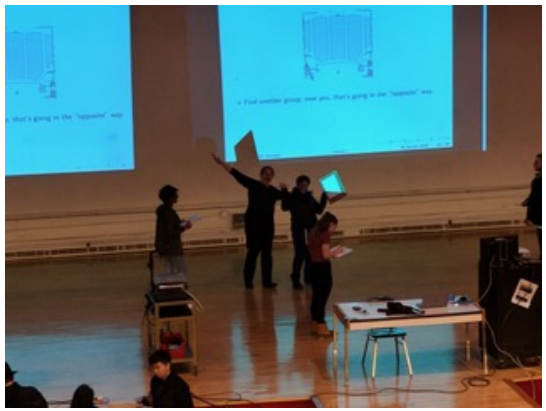
- On lecture slip, write down a topic you wish we had spent more time (and why).
- Writing precise algorithms is difficult.
- In Python, we introduced:
  - strings, or sequences of characters,
  - `print()` statements,
  - `for`-loops with `range()` statements, &
  - variables containing turtles.

Recap



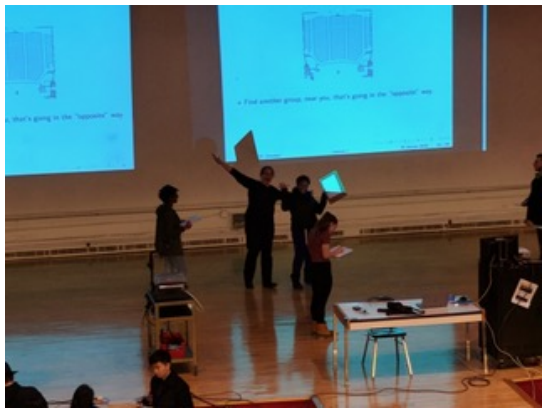
- On lecture slip, write down a topic you wish we had spent more time (and why).
- Writing precise algorithms is difficult.
- In Python, we introduced:
  - strings, or sequences of characters,
  - `print()` statements,
  - `for`-loops with `range()` statements, &
  - variables containing turtles.
- Pass your lecture slips to the aisle for the UTA's to collect.

Practice Quiz & Final Questions



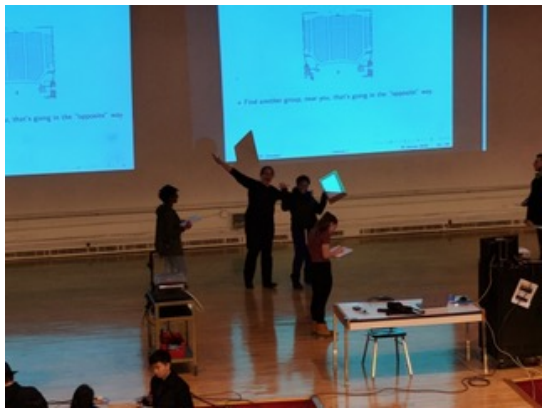
- Since you must pass the final exam to pass the course, we end every lecture with final exam review.

Practice Quiz & Final Questions



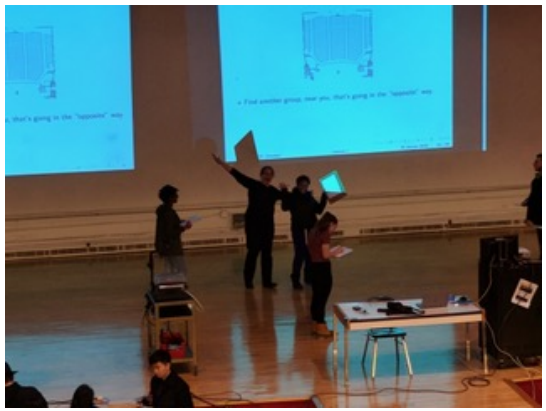
- Since you must pass the final exam to pass the course, we end every lecture with final exam review.
- Pull out something to write on (not to be turned in).

Practice Quiz & Final Questions



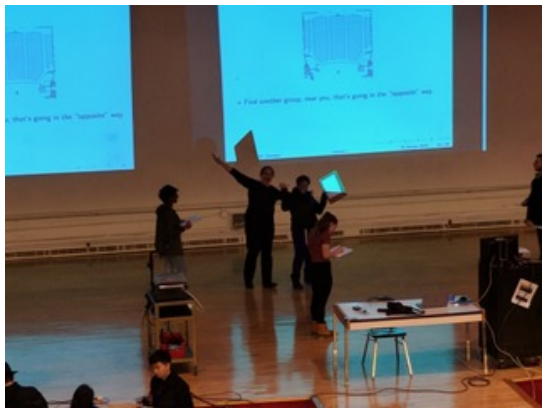
- Since you must pass the final exam to pass the course, we end every lecture with final exam review.
- Pull out something to write on (not to be turned in).
- Lightning rounds:

Practice Quiz & Final Questions



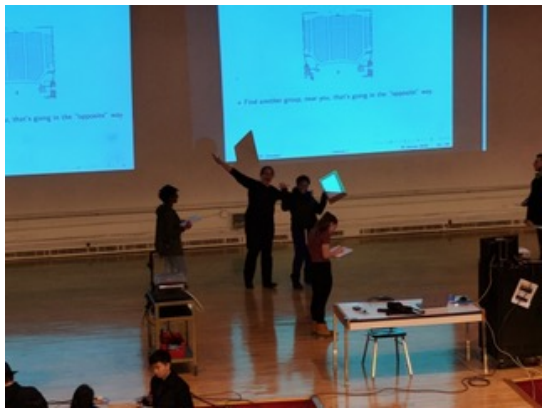
- Since you must pass the final exam to pass the course, we end every lecture with final exam review.
- Pull out something to write on (not to be turned in).
- Lightning rounds:
  - write as much you can for 60 seconds;

Practice Quiz & Final Questions



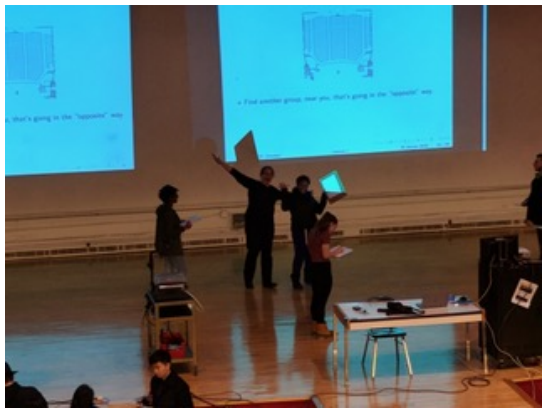
- Since you must pass the final exam to pass the course, we end every lecture with final exam review.
- Pull out something to write on (not to be turned in).
- Lightning rounds:
  - write as much you can for 60 seconds;
  - followed by answer; and

Practice Quiz & Final Questions



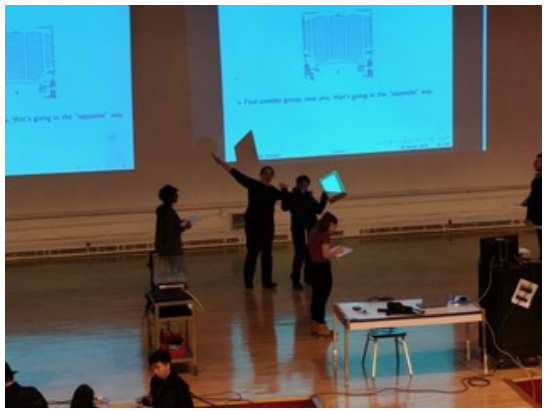
- Since you must pass the final exam to pass the course, we end every lecture with final exam review.
- Pull out something to write on (not to be turned in).
- Lightning rounds:
  - write as much you can for 60 seconds;
  - followed by answer; and
  - repeat.

Practice Quiz & Final Questions



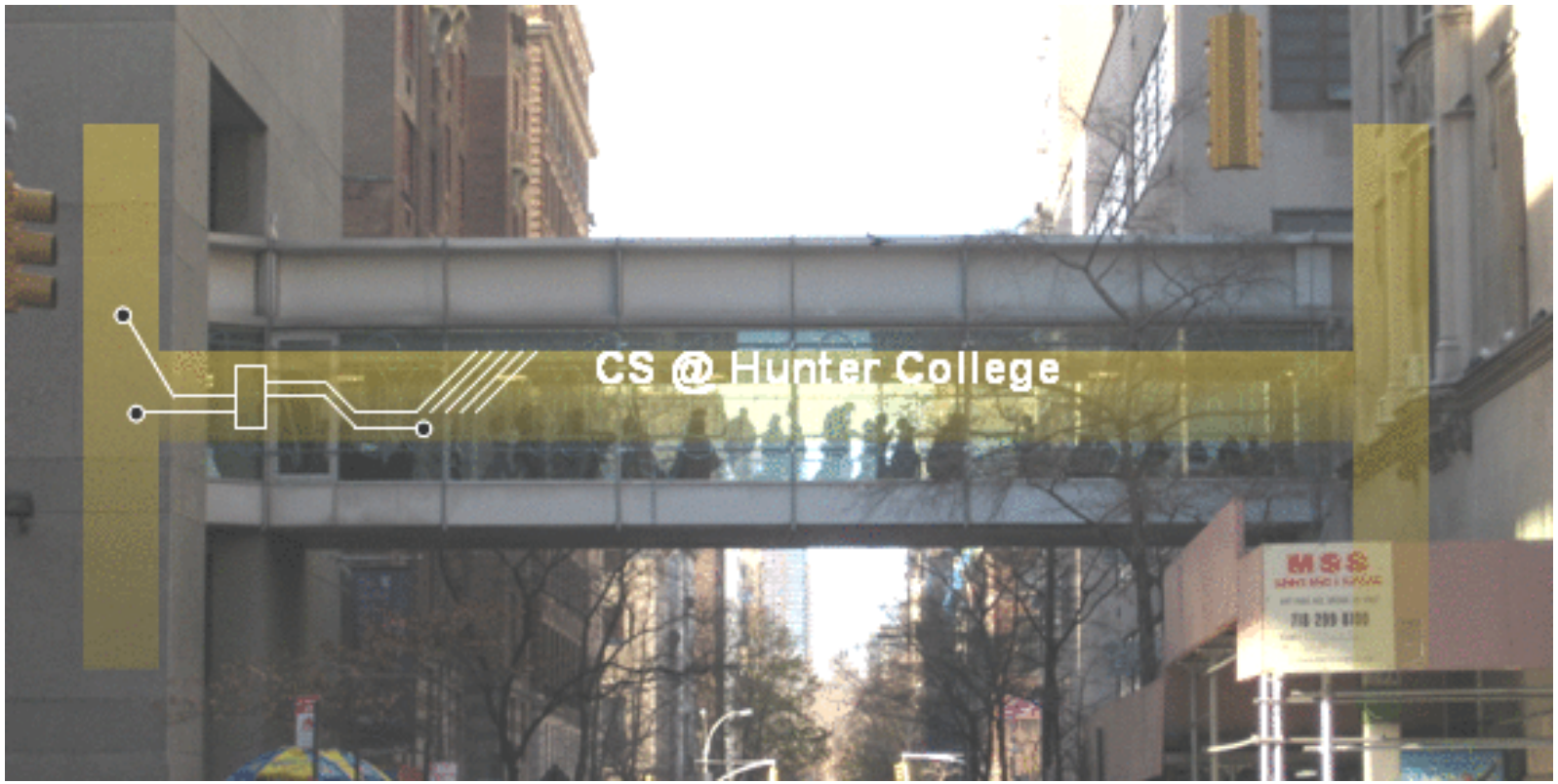
- Since you must pass the final exam to pass the course, we end every lecture with final exam review.
- Pull out something to write on (not to be turned in).
- Lightning rounds:
  - write as much you can for 60 seconds;
  - followed by answer; and
  - repeat.
- Past exams are on the webpage (under Final Exam Information).

Practice Quiz & Final Questions



- Since you must pass the final exam to pass the course, we end every lecture with final exam review.
- Pull out something to write on (not to be turned in).
- Lightning rounds:
  - write as much you can for 60 seconds;
  - followed by answer; and
  - repeat.
- Past exams are on the webpage (under Final Exam Information).
- We're starting with Fall 2017, Version 1.

Writing Boards



- Return writing boards as you leave...