

CSci 127: Introduction to Computer Science



hunter.cuny.edu/csci

Welcome



- This lecture will be recorded

Introductions: Course Designers



Dr. Katherine St. John

Professor,
Interim Chair

Dr. William Sakas

Associate Professor,
Chair

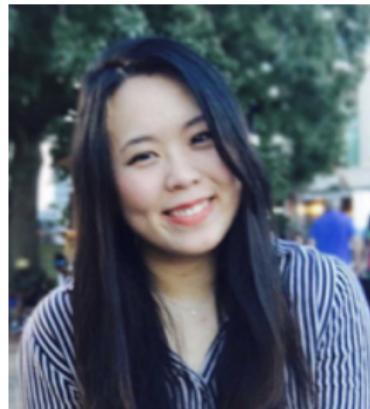
Prof. Eric Schweitzer

Undergraduate Program
Coordinator

Introductions: Instructors



Lola Samigjonova



Yoomin Song

Monday/Tuesday
Wednesday/Thursday

Monday/
Wednesday

Introductions: Advisors



Emely Peguero
Pre-majors & Early Majors
emely.pegueronova@hunter.cuny.edu



Eric Schweitzer
Undergraduate Program Coordinator
eschweitz@hunter.cuny.edu

Where to find Course Content

- Course Website: <https://huntercsci127.github.io/summer23.html>

Where to find Course Content

- Course Website: <https://huntercsci127.github.io/summer23.html>
- Blackboard

Where to find Course Content

- Course Website: <https://huntercsci127.github.io/summer23.html>
- Blackboard
- Gradescope (assessment)

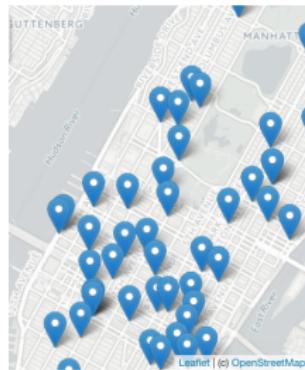
Syllabus

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...*

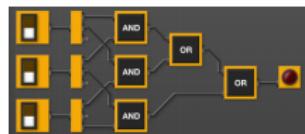
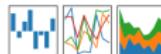
This course is pre-requisite to several introductory core courses in the CS Major. The course is also required for the CS minor. MATH 12500 or higher is strongly recommended as a co-req for intended Majors.

Syllabus: Topics

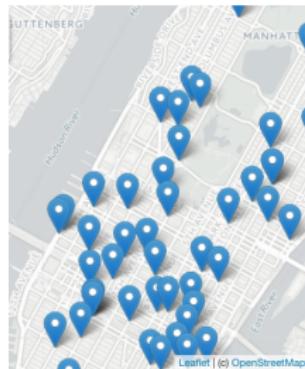


- This course assumes no previous programming experience.

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

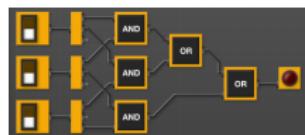
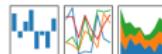


Syllabus: Topics

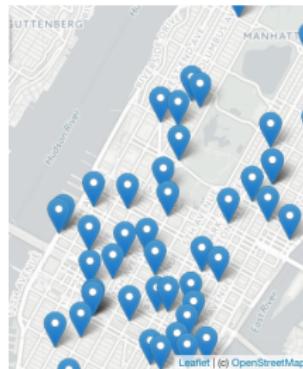


- **This course assumes no previous programming experience.**
- Organized like a fugue, with variations on this theme:

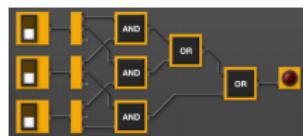
pandas
 $y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$



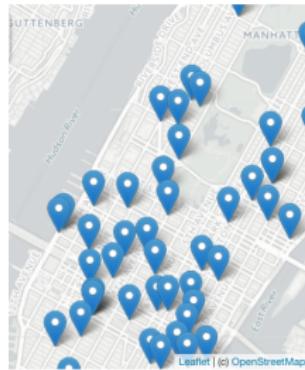
Syllabus: Topics



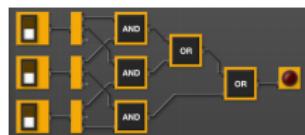
- **This course assumes no previous programming experience.**
- Organized like a fugue, with variations on this theme:
 - ▶ Introduce coding constructs in Python,



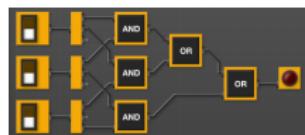
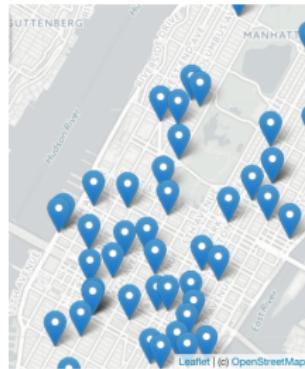
Syllabus: Topics



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

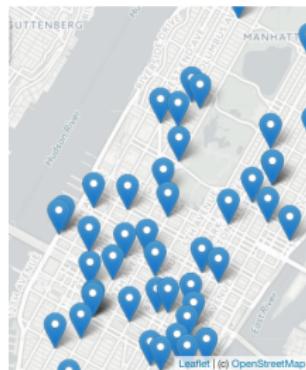


Syllabus: Topics

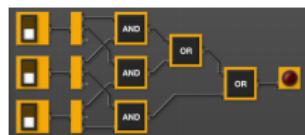
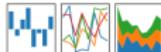


- **This course assumes no previous programming experience.**
- Organized like a fugue, with variations on this theme:
 - ▶ Introduce coding constructs in Python,
 - ▶ Apply those ideas to different problems (e.g. analyzing & mapping data),
 - ▶ See constructs again:

Syllabus: Topics

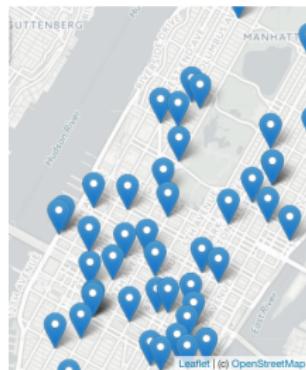


pandas
 $y_t = \beta' x_t + \mu_t + \epsilon_{t1}$

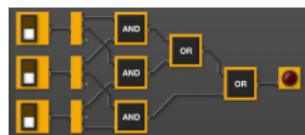
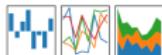


- **This course assumes no previous programming experience.**
- Organized like a fugue, with variations on this theme:
 - ▶ Introduce coding constructs in Python,
 - ▶ Apply those ideas to different problems (e.g. analyzing & mapping data),
 - ▶ See constructs again:
 - ★ for logical circuits,

Syllabus: Topics

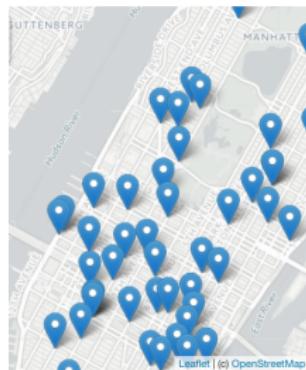


pandas
 $y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$

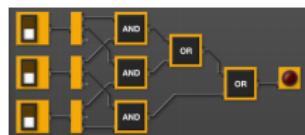
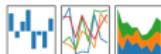


- **This course assumes no previous programming experience.**
- Organized like a fugue, with variations on this theme:
 - ▶ Introduce coding constructs in Python,
 - ▶ Apply those ideas to different problems (e.g. analyzing & mapping data),
 - ▶ See constructs again:
 - ★ for logical circuits,
 - ★ for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

Syllabus: Topics

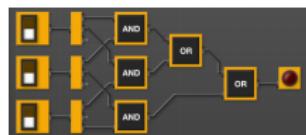
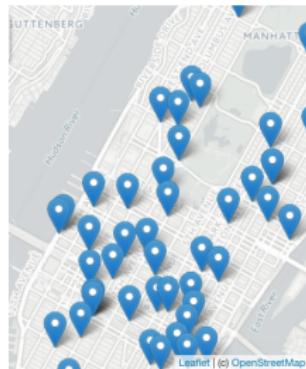


pandas
 $y_t = \beta' x_{it} + \mu_i + \epsilon_{it}$



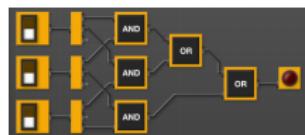
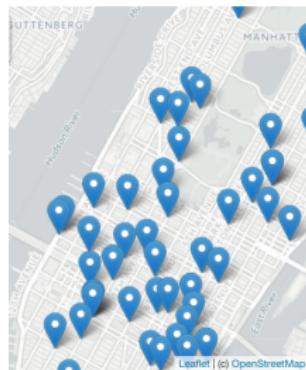
- **This course assumes no previous programming experience.**
- Organized like a fugue, with variations on this theme:
 - ▶ Introduce coding constructs in Python,
 - ▶ Apply those ideas to different problems (e.g. analyzing & mapping data),
 - ▶ See constructs again:
 - ★ for logical circuits,
 - ★ for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

Syllabus: Topics



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

Course Structure



Each Week:

- **Class Meets Four Times a Week**

First "computers"

ENIAC, 1945.

Course Structure



Each Week:

- **Class Meets Four Times a Week**
- While one class reviews Lecture Slides.

First "computers"

ENIAC, 1945.

Course Structure



Each Week:

- **Class Meets Four Times a Week**
- While one class reviews Lecture Slides.
- Another will review Interactive Labs.

First "computers"

ENIAC, 1945.

Course Structure



Each Week:

- **Class Meets Four Times a Week**
- While one class reviews Lecture Slides.
- Another will review Interactive Labs.
- *You are expected to code interactively.*

First "computers"

ENIAC, 1945.

Course Structure



Each Week:

- **Class Meets Four Times a Week**
- While one class reviews Lecture Slides.
- Another will review Interactive Labs.
- *You are expected to code interactively.*
- After class you must complete 15 minute quiz.

First "computers"

ENIAC, 1945.

Course Structure



First "computers"

ENIAC, 1945.

Each Week:

- **Class Meets Four Times a Week**
- While one class reviews Lecture Slides.
- Another will review Interactive Labs.
- *You are expected to code interactively.*
- After class you must complete 15 minute quiz.
- *The quiz is only available in person at the last 15 minutes of class.*

Homework



Each Class:

- **5 Programming Assignments.**

First "computers"

ENIAC, 1945.

Homework



Each Class:

- **5 Programming Assignments.**
- Description on Course Webpage.

First "computers"

ENIAC, 1945.

Homework



Each Class:

- **5 Programming Assignments.**
- Description on Course Webpage.
- Implement and test on your computer.

First "computers"

ENIAC, 1945.

Homework



Each Class:

- **5 Programming Assignments.**
- Description on Course Webpage.
- Implement and test on your computer.
- Submit to Gradescope.

First "computers"

ENIAC, 1945.

Homework



Each Class:

- **5 Programming Assignments.**
- Description on Course Webpage.
- Implement and test on your computer.
- Submit to Gradescope.
- Multiple submissions accepted.

First "computers"

ENIAC, 1945.

Homework



Each Class:

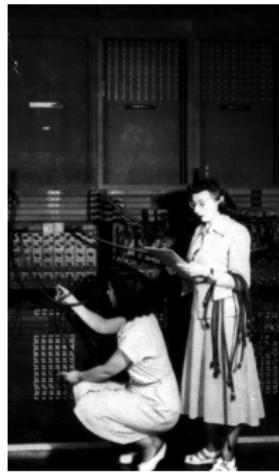
- **5 Programming Assignments.**
- Description on Course Webpage.
- Implement and test on your computer.
- Submit to Gradescope.
- Multiple submissions accepted.
- Assignments are due in Batches (see course calendar)

First "computers"

ENIAC, 1945.

Academic Dishonesty

- *The person who does the work gets the benefit! Learning is personal!!!*



First "computers"

ENIAC, 1945.

Academic Dishonesty



First "computers"

ENIAC, 1945.

- *The person who does the work gets the benefit! Learning is personal!!!*
- **Don't waste your time and money!**

Academic Dishonesty



First "computers"

ENIAC, 1945.

- *The person who does the work gets the benefit! Learning is personal!!!*
- **Don't waste your time and money!**
- A few semesters down the road will be too late to catch up on core knowledge and **skills**.

Academic Dishonesty

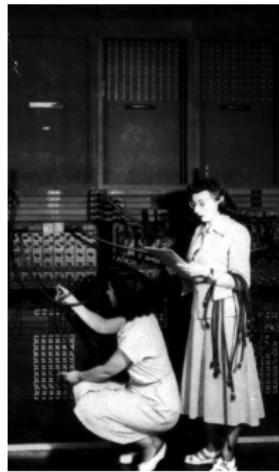


First "computers"

ENIAC, 1945.

- *The person who does the work gets the benefit! Learning is personal!!!*
- **Don't waste your time and money!**
- A few semesters down the road will be too late to catch up on core knowledge and **skills**.
- Cheating is immoral and it lowers the quality of our students and institution.

Academic Dishonesty



First "computers"

ENIAC, 1945.

- *The person who does the work gets the benefit! Learning is personal!!!*
- **Don't waste your time and money!**
- A few semesters down the road will be too late to catch up on core knowledge and **skills**.
- Cheating is immoral and it lowers the quality of our students and institution.
- Students that pose as experts often circulate bad/incorrect solutions

Academic Dishonesty



First "computers"

ENIAC, 1945.

- *The person who does the work gets the benefit! Learning is personal!!!*
- **Don't waste your time and money!**
- A few semesters down the road will be too late to catch up on core knowledge and **skills**.
- Cheating is immoral and it lowers the quality of our students and institution.
- Students that pose as experts often circulate bad/incorrect solutions
- **All instances of academic dishonesty will be reported to the office of Student Affairs**

Communication



First "computers"

ENIAC, 1945.

- Important weekly communication sent via Blackboard

Communication



- Important weekly communication sent via Blackboard
- Check your email account associated with Blackboard

First "computers"

ENIAC, 1945.

Communication



First "computers"

ENIAC, 1945.

- Important weekly communication sent via Blackboard
- Check your email account associated with Blackboard
- **Check your Spam folder**

Communication



- Important weekly communication sent via Blackboard
- Check your email account associated with Blackboard
- **Check your Spam folder**
- Email studenthelpdesk@hunter.cuny.edu if you need to change it

First "computers"

ENIAC, 1945.

How to Succeed in this Course

Each Week:

- Come to Class

How to Succeed in this Course

Each Week:

- Come to Class
 - ▶ Pay attention during lecture.

How to Succeed in this Course

Each Week:

- Come to Class
 - ▶ Pay attention during lecture.
 - ▶ Actively participate in lecture work: try to solve problems/challenges

How to Succeed in this Course

Each Week:

- Come to Class
 - ▶ Pay attention during lecture.
 - ▶ Actively participate in lecture work: try to solve problems/challenges
- Read the Lab and participate in Lab Review.

How to Succeed in this Course

Each Week:

- Come to Class
 - ▶ Pay attention during lecture.
 - ▶ Actively participate in lecture work: try to solve problems/challenges
- Read the Lab and participate in Lab Review.
- Take the weekly Lab Quiz.

How to Succeed in this Course

Each Week:

- Come to Class
 - ▶ Pay attention during lecture.
 - ▶ Actively participate in lecture work: try to solve problems/challenges
- Read the Lab and participate in Lab Review.
- Take the weekly Lab Quiz.
- Work on THIS CLASS'S Programming Assignments.

How to Succeed in this Course

Each Week:

- Come to Class
 - ▶ Pay attention during lecture.
 - ▶ Actively participate in lecture work: try to solve problems/challenges
- Read the Lab and participate in Lab Review.
- Take the weekly Lab Quiz.
- Work on THIS CLASS'S Programming Assignments.
- Ask for help.

Today's Topics



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

Today's Topics



- **Introduction to Python**
- Turtle Graphics
- Definite Loops (for-loops)
- 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 extensibility, supportive community with hundreds of open source libraries and frameworks.



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 extensibility, supportive community with hundreds of open source libraries and frameworks.
- 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 extensibility, supportive community with hundreds of open source libraries and frameworks.
- 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).

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)

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!
- We know that Hello, World! is a **string** (a sequence of characters) because it is surrounded by quotes

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!
- We know that Hello, World! is a **string** (a sequence of characters) because it is surrounded by quotes
- 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,')
```

*Spring18 here in Assembly Hall
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,')  
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.

Today's Topics



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

Turtles Introduction

- A simple, whimsical graphics package for Python.



Turtles Introduction

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



Turtles Introduction



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

Turtles Introduction



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

Today's Topics



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

Turtles Introduction

The screenshot shows a Python code editor interface. On the left, the code file `main.py` is open, containing the following Python script:

```
1 #A program that demonstrates turtles stamping
2
3 import turtle
4
5 taylor = turtle.Turtle()
6 taylor.color("purple")
7 taylor.shape("turtle")
8
9 for i in range(6):
10     taylor.forward(100)
11     taylor.stamp()
12     taylor.left(60)
```

On the right, there are two tabs: "Result" and "Instructions". The "Result" tab is active, displaying the output of the program: a purple turtle shape that has drawn a regular hexagon on the screen, with six purple star-like stamps at each vertex of the hexagon.

- Creates a turtle **variable**, called `taylor`.

Turtles Introduction

The screenshot shows a Python code editor interface. At the top, there are standard file operations: New, Open, Save, Print, and Exit. Below the toolbar, the file name is "main.py". The code area contains the following Python script:

```
1 #A program that demonstrates turtles stamping
2
3 import turtle
4
5 taylor = turtle.Turtle()
6 taylor.color("purple")
7 taylor.shape("turtle")
8
9 for i in range(6):
10     taylor.forward(100)
11     taylor.stamp()
12     taylor.left(60)
```

To the right of the code editor is a results panel. It has two tabs: "Result" and "Instructions". The "Result" tab is active, showing the output of the turtle program: a regular hexagon drawn in purple, with each vertex marked by a purple star-like stamp.

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

Turtles Introduction

The screenshot shows a Python code editor with a toolbar at the top. The file tab shows "main.py". The code in the editor is:

```
1 #A program that demonstrates turtles stamping
2
3 import turtle
4
5 taylor = turtle.Turtle()
6 taylor.color("purple")
7 taylor.shape("turtle")
8
9 for i in range(6):
10     taylor.forward(100)
11     taylor.stamp()
12     taylor.left(60)
```

To the right of the code editor is a "Result" panel showing the output of the program. It displays a purple turtle shape that has drawn a regular hexagon. The turtle's path is shown as a purple line connecting six purple star-shaped stamps. The turtle is currently positioned at the bottom-left vertex of the hexagon.

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

Turtles Introduction

The screenshot shows a Python code editor interface. On the left, the code file `main.py` is open, containing the following Python script:

```
1 #A program that demonstrates turtles stamping
2
3 import turtle
4
5 taylor = turtle.Turtle()
6 taylor.color("purple")
7 taylor.shape("turtle")
8
9 for i in range(6):
10     taylor.forward(100)
11     taylor.stamp()
12     taylor.left(60)
```

On the right, there are two tabs: "Result" and "Instructions". The "Result" tab displays the output of the program: a purple turtle shape that has drawn a regular hexagon on the screen, with each vertex marked by a purple star-like stamp.

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

Turtles Introduction

The screenshot shows a Python code editor with a toolbar at the top. The file tab shows "main.py". The code in the editor is:

```
1 #A program that demonstrates turtles stamping
2
3 import turtle
4
5 taylor = turtle.Turtle()
6 taylor.color("purple")
7 taylor.shape("turtle")
8
9 for i in range(6):
10     taylor.forward(100)
11     taylor.stamp()
12     taylor.left(60)
```

To the right of the code editor is a "Result" panel showing the output of the program: a purple hexagon drawn with turtle steps and stamps.

- Creates a turtle **variable**, called **taylor**.
- Changes the color (to purple) and shape (to turtle-shaped).
- Repeats 6 times:
 - ▶ Move forward; stamp; and turn left 60 degrees.
- Repeats any instructions **indented** in the "loop block"

Turtles Introduction

The screenshot shows a Python code editor with a toolbar at the top. The file tab shows "main.py". The code in the editor is:

```
1 #A program that demonstrates turtles stamping
2
3 import turtle
4
5 taylor = turtle.Turtle()
6 taylor.color("purple")
7 taylor.shape("turtle")
8
9 for i in range(6):
10     taylor.forward(100)
11     taylor.stamp()
12     taylor.left(60)
```

To the right of the code editor is a "Result" panel showing the output of the program: a purple turtle shape that has drawn a regular hexagon on the screen, with six black star-like stamps at each vertex.

- Creates a turtle **variable**, called `taylor`.
- Changes the color (to purple) and shape (to turtle-shaped).
- Repeats 6 times:
 - ▶ Move forward; stamp; and turn left 60 degrees.
- Repeats any instructions **indented** in the "loop block"
- This is a **definite** loop because it repeats a fixed number of times

Your Turn!!!

Try to solve this challenge:

- 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

The screenshot shows a Python code editor with a file named `main.py` containing the following code:

```
1 #A program that demonstrates turtles stamping
2
3 import turtle
4
5 taylor = turtle.Turtle()
6 taylor.color("purple")
7 taylor.shape("turtle")
8
9 for i in range(6):
10     taylor.forward(100)
11     taylor.stamp()
12     taylor.left(60)
```

The editor has a toolbar at the top with icons for file operations like Open, Save, and Run. Below the toolbar is a tab bar with `main.py` selected. To the right of the code area are two tabs: `Result` and `Instructions`. The `Result` tab displays a purple hexagon drawn by the turtle, with each vertex marked by a purple star.

- Start with the hexagon program.

Decagon Program

The screenshot shows a code editor window with a toolbar at the top. The file tab shows "main.py". The code in the editor is:

```
1 #A program that demonstrates turtles stamping
2
3 import turtle
4
5 taylor = turtle.Turtle()
6 taylor.color("purple")
7 taylor.shape("turtle")
8
9 for i in range(10):
10     taylor.forward(100)
11     taylor.stamp()
12     taylor.left(60)
```

The "Result" tab is active, displaying a purple decagon (10-sided polygon) drawn with a turtle. The turtle's path is purple, and it has stamped its position at each vertex of the decagon.

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

Decagon Program

The screenshot shows a code editor window with a toolbar at the top. The file name is 'main.py'. The code is as follows:

```
1 #A program that demonstrates turtles stamping
2
3 import turtle
4
5 taylor = turtle.Turtle()
6 taylor.color("purple")
7 taylor.shape("turtle")
8
9 for i in range(6):
10     taylor.forward(100)
11     taylor.stamp()
12     taylor.left(60)
```

The 'Result' tab is selected, showing a purple hexagon drawn on a white background. Each vertex of the hexagon has a small purple star-like stamp.

- 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!")
```

Today's Topics



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

What is an Algorithm?

From our textbook:

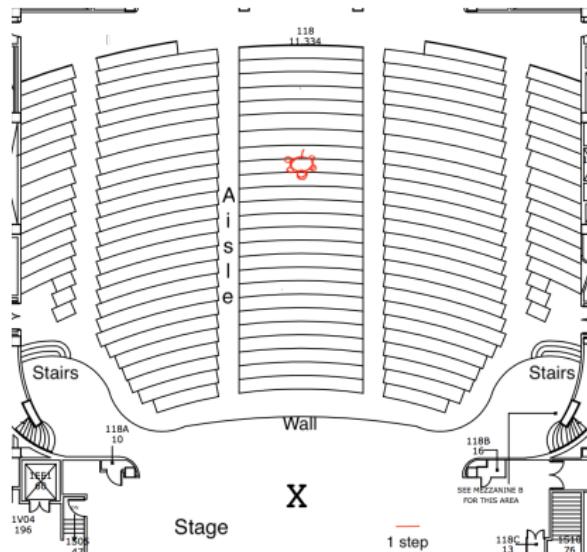
- An **algorithm** is a process or sequence of steps to be followed to solve a problem.

What is an Algorithm?

From our textbook:

- An **algorithm** is a process or sequence of steps 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 executed by a computer.

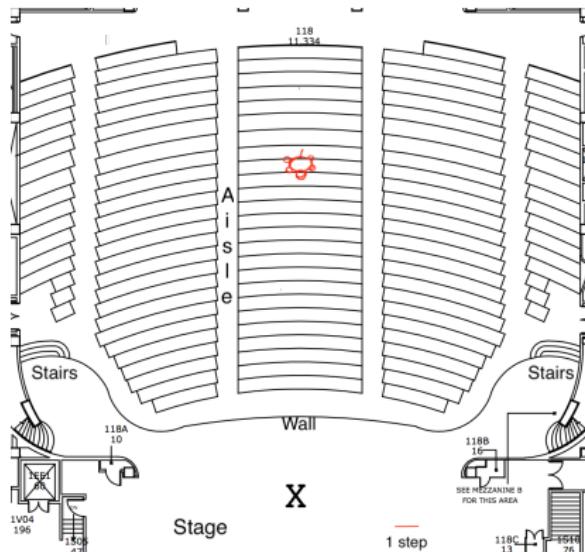
Your Turn!!!



Try to solve this challenge:

- ① This is the floor plan of Assembly Hall at Hunter College.

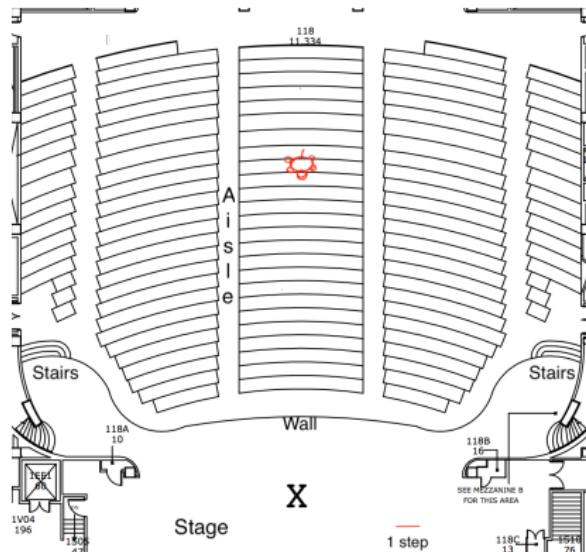
Your Turn!!!



Try to solve this challenge:

- ① This is the floor plan of Assembly Hall at Hunter College.
- ② Write an algorithm (step-by-step directions) to the red turtle to the X

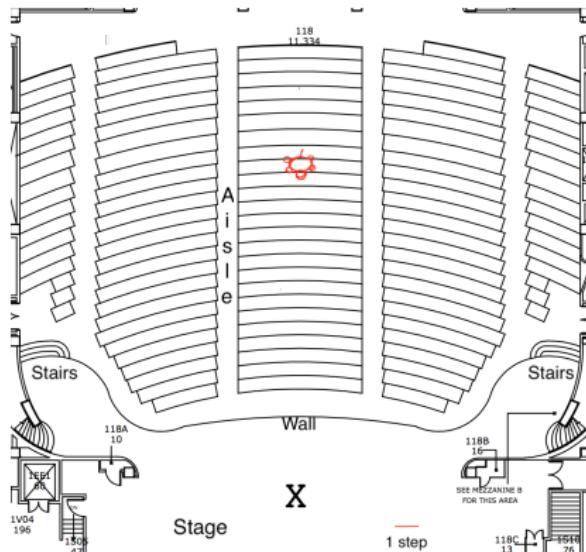
Your Turn!!!



Try to solve this challenge:

- ① This is the floor plan of Assembly Hall at Hunter College.
- ② Write an algorithm (step-by-step directions) to the red turtle to the X.
- ③ Basic Rules:

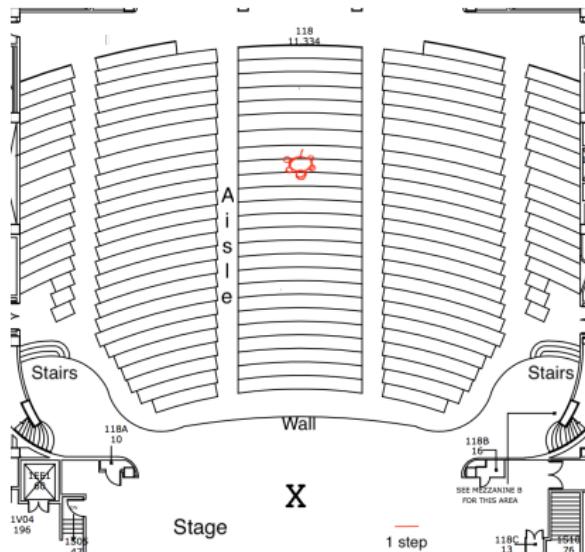
Your Turn!!!



Try to solve this challenge:

- ① This is the floor plan of Assembly Hall at Hunter College.
- ② Write an algorithm (step-by-step directions) to the red turtle to the X.
- ③ Basic Rules:
 - ▶ Use turtle commands.

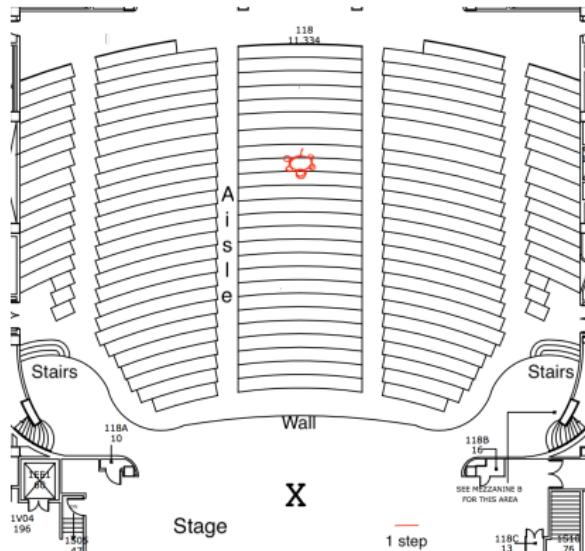
Your Turn!!!



Try to solve this challenge:

- ① This is the floor plan of Assembly Hall at Hunter College.
- ② Write an algorithm (step-by-step directions) to the red turtle to the X.
- ③ Basic Rules:
 - ▶ Use turtle commands.
 - ▶ Do not run turtles into walls, chairs, obstacles, etc.

Your Turn!!!



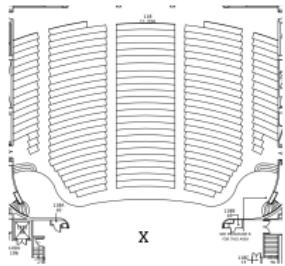
Try to solve this challenge:

- ① This is the floor plan of Assembly Hall at Hunter College.
- ② Write an algorithm (step-by-step directions) to the red turtle to the X.
- ③ Basic Rules:
 - ▶ Use turtle commands.
 - ▶ Do not run turtles into walls, chairs, obstacles, etc.
 - ▶ Turtles cannot climb walls, must use stairs (walk forward on steps).



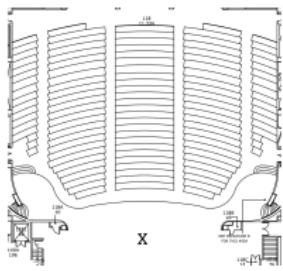
Recap

- Writing precise algorithms is difficult.

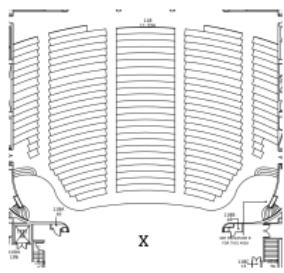


Recap

- Writing precise algorithms is difficult.
- In Python, we introduced:

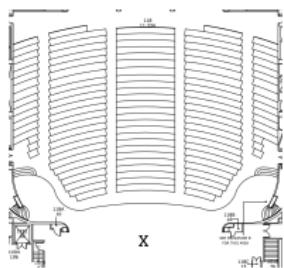


Recap



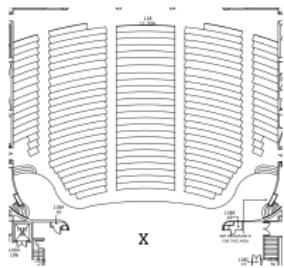
- Writing precise algorithms is difficult.
- In Python, we introduced:
 - ▶ **strings**, or sequences of characters,

Recap



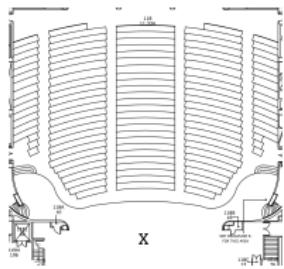
- Writing precise algorithms is difficult.
- In Python, we introduced:
 - ▶ `strings`, or sequences of characters,
 - ▶ `print()` statements,

Recap



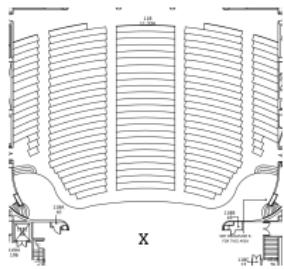
- Writing precise algorithms is difficult.
- In Python, we introduced:
 - ▶ `strings`, or sequences of characters,
 - ▶ `print()` statements,
 - ▶ `for-loops` with `range()` statements, &

Recap



- 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



- Writing precise algorithms is difficult.
- In Python, we introduced:
 - ▶ `strings`, or sequences of characters,
 - ▶ `print()` statements,
 - ▶ `for-loops` with `range()` statements, &
 - ▶ `variables` containing turtles.

Reminders!



Before next class, don't forget to:

- Review LAB 1!

Reminders!



Before next class, don't forget to:

- Review LAB 1!
- Submit this week's 5 programming assignments (programs 1-5)