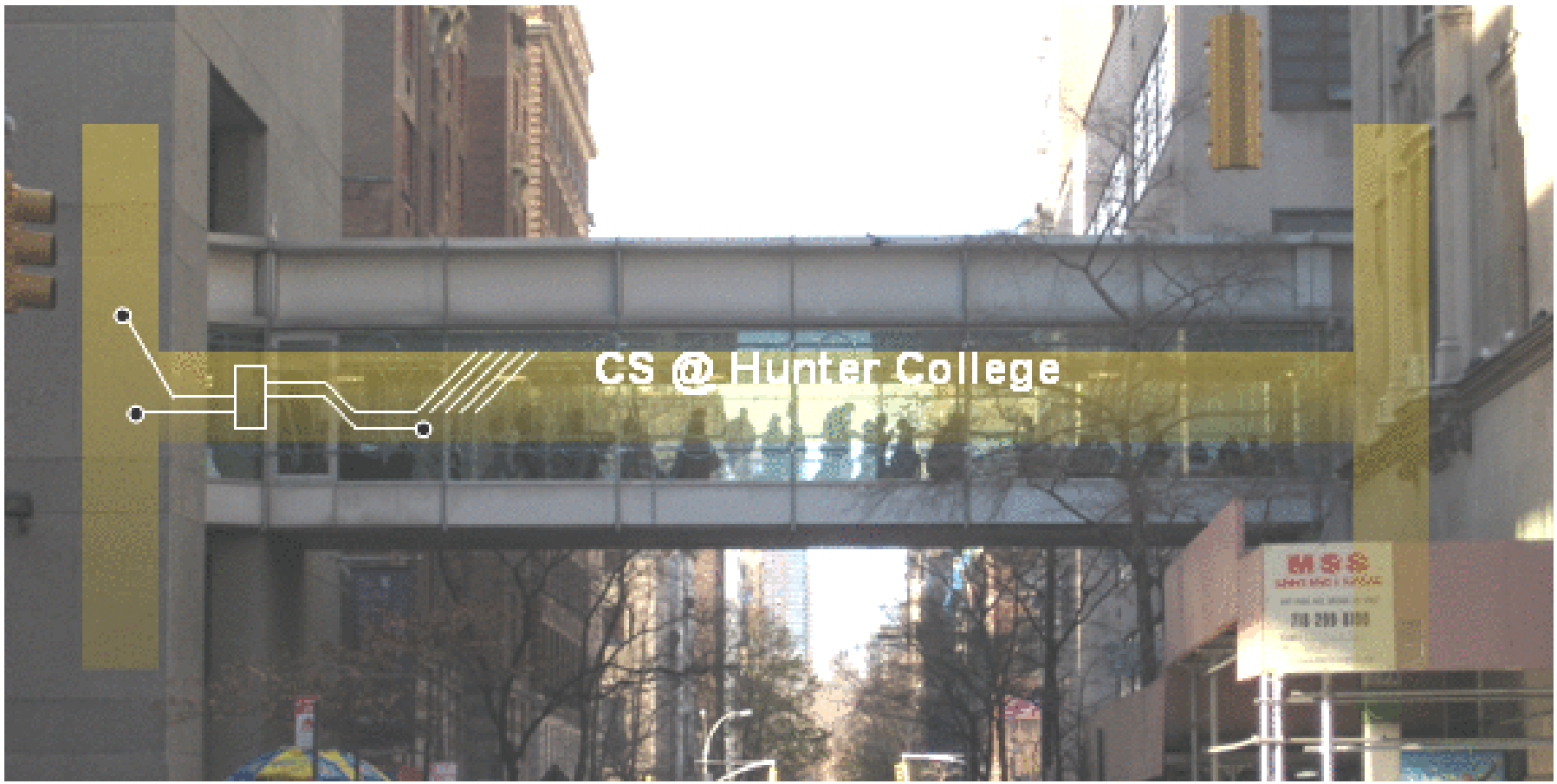
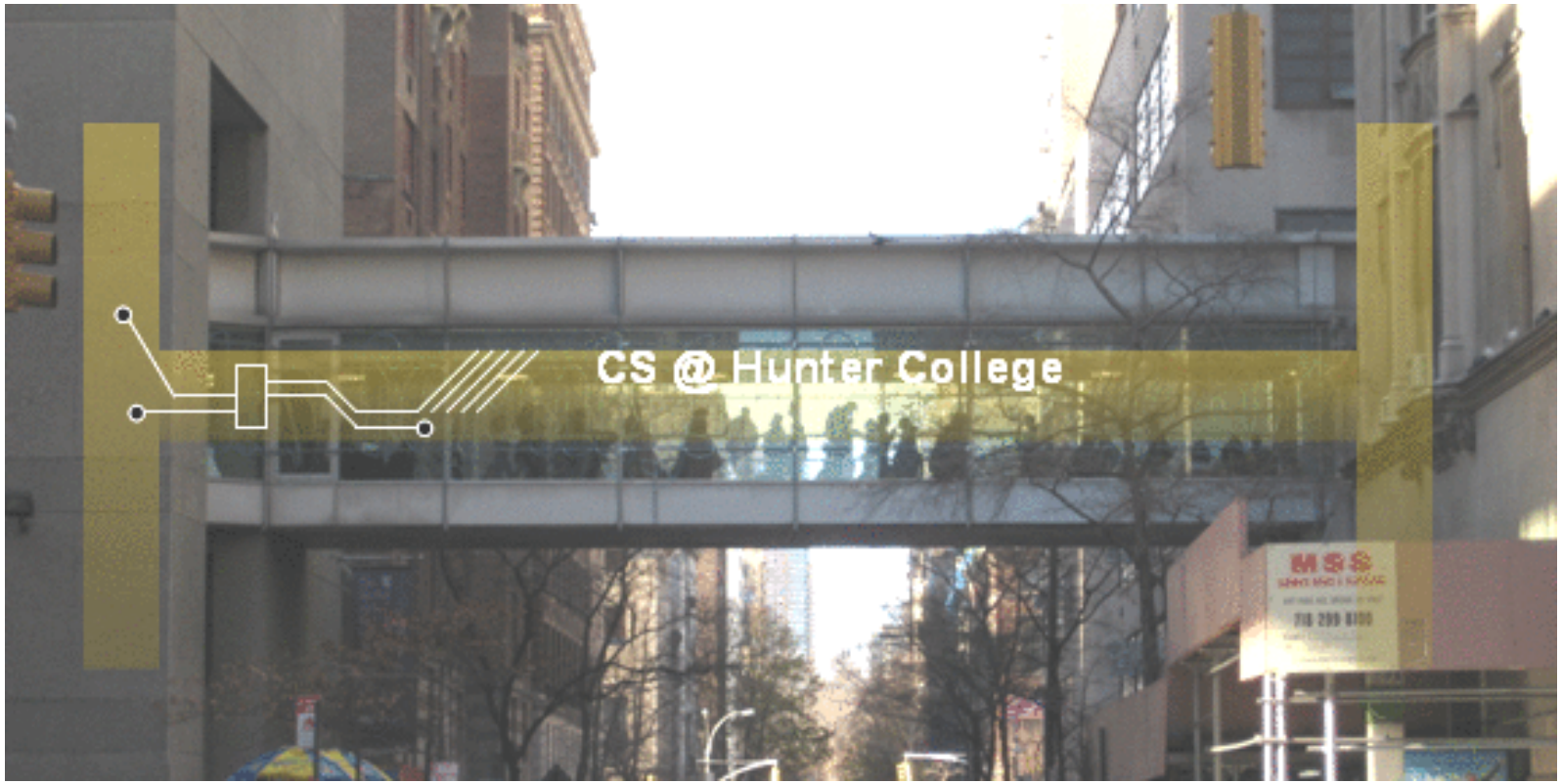
CSci 127: Introduction to Computer Science



hunter.cuny.edu/csci

Welcome



Acknowledgments

Thank you to the amazing support of:

President Raab

Dean Polsky
Arts & Science

Judy Spitz
Break Through Tech

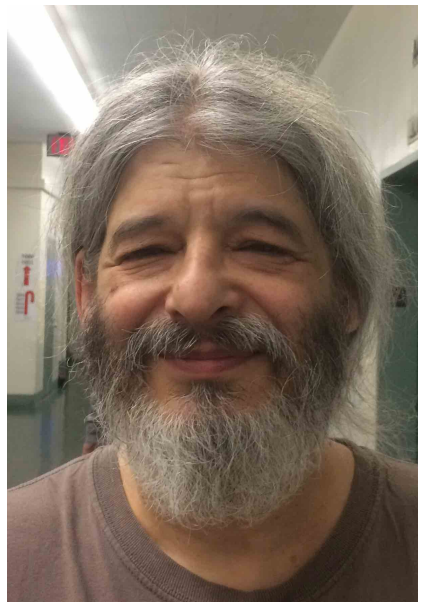
Introductions: Course Designers

Dr. Katherine St. John

Professor,
Course Coordinator

Dr. William Sakas

Associate Professor,
Chair



Prof. Eric Schweitzer

Undergraduate Program
Coordinator

Introductions: Instructors

Katherine Howitt

Early College Initiative

Dr. Tiziana Ligorio

Large Lecture
Macaulay Honors Section

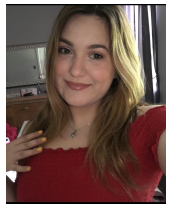
Introductions: Undergraduate Teaching Assistants

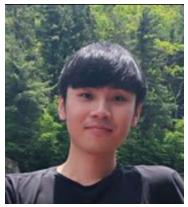

Aida Jevric

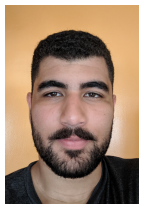

Ajani Stewart


Arterio Rodrigues


Brian Chambers

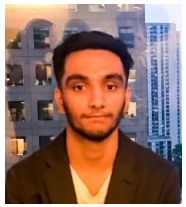

Caiitlin Selca

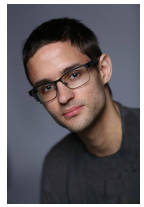

Chi Shing Lee


David Moncayo


David Yuen

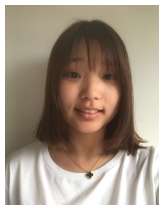

Destiny Barbery


Ghazanfar Shahbaz


Illya Baburashvili

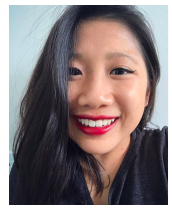

Kevin Wong


Leonardo Matone


Liulan Zheng


Lola Samigjonova

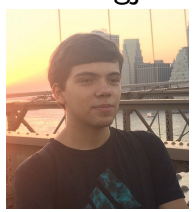

Mandy Yu


Nancy Ng

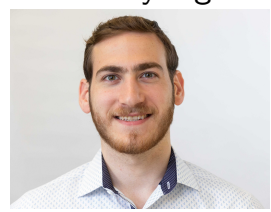

Nga Yu Lo


Owen Kunhardt


Patrick Chaca

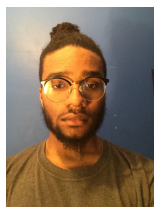

Ryan Chevarria

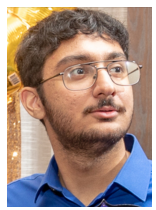

Sadab Hafiz


Seth Spiegel


Shantel Dixon

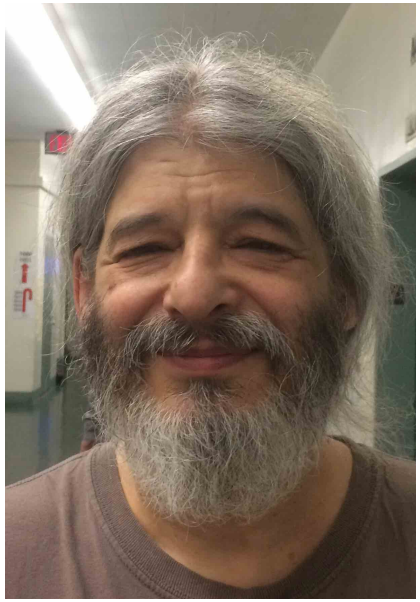

Stephanie Yung


Tyler Robinson


Yash Mahtani

Introductions: Advisors

Emely Peguero
Pre-majors &
Early Majors



Eric Schweitzer
Undergraduate Program
Coordinator

Justin Tojeira
Internships &
Upper Division

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.*

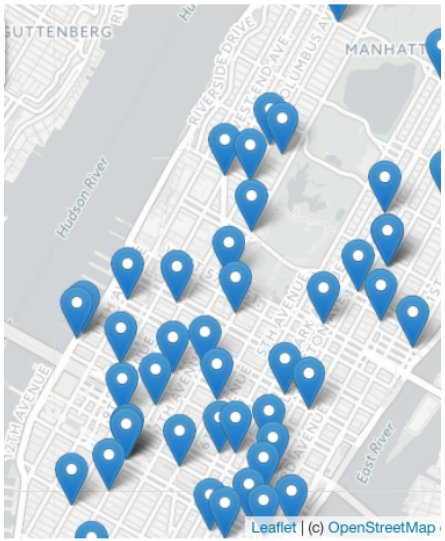
Syllabus

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.*

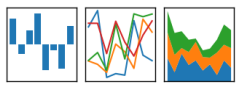
(Show syllabus webpage)

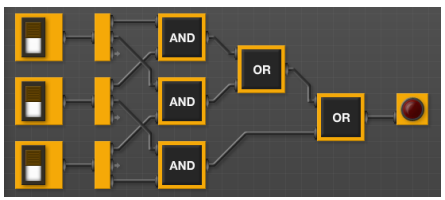
Syllabus: Topics



pandas

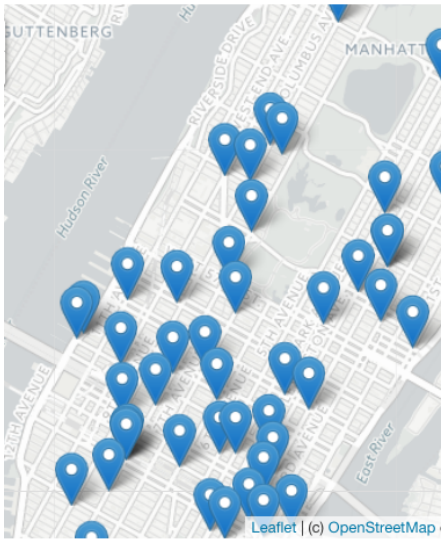
$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$





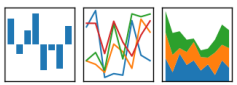
- **This course assumes no previous programming experience.**

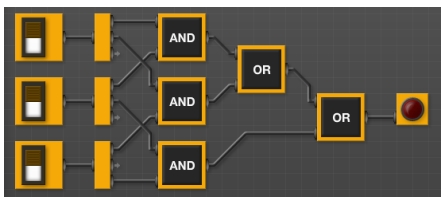
Syllabus: Topics



pandas

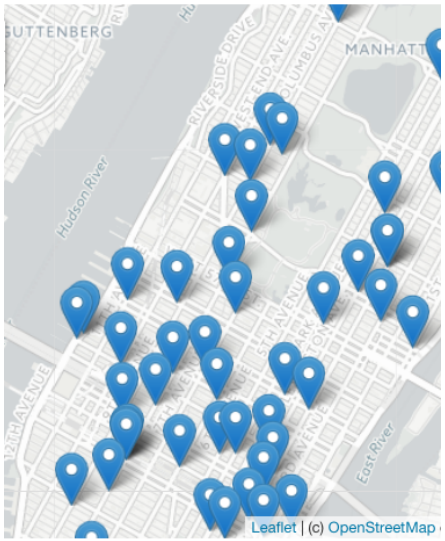
$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$

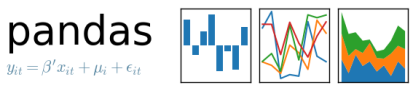


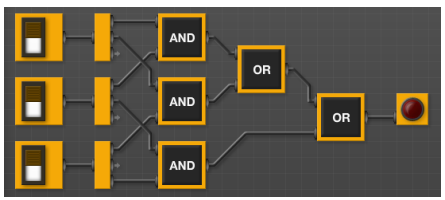


- **This course assumes no previous programming experience.**
- "... Emphasis on problem-solving and computational thinking through 'coding': computer programming for beginners..."

Syllabus: Topics

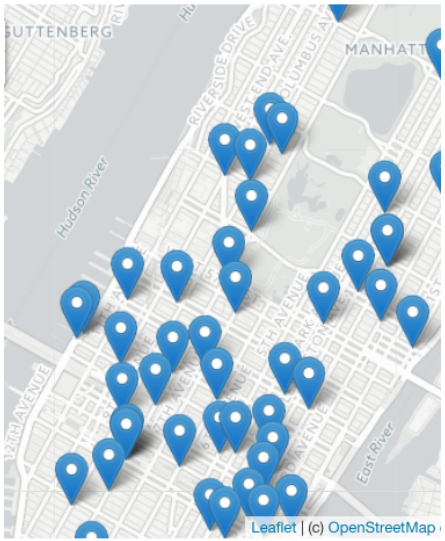


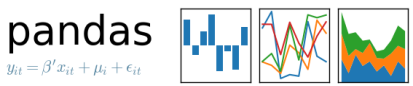


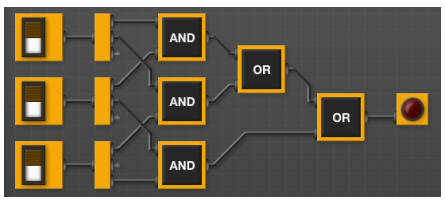


- **This course assumes no previous programming experience.**
- "... Emphasis on problem-solving and computational thinking through 'coding': computer programming for beginners..."
- Organized like a fugue, with variations on this theme:

Syllabus: Topics

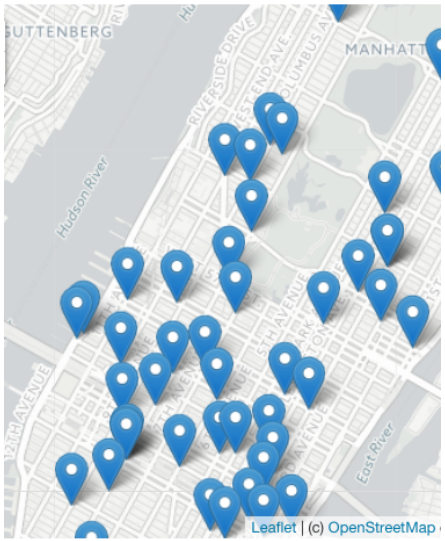


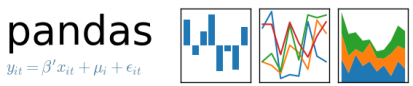


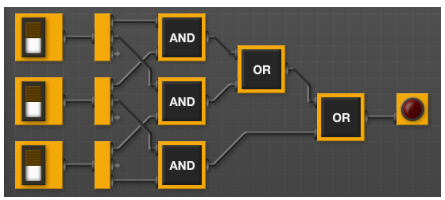


- **This course assumes no previous programming experience.**
- "... Emphasis on problem-solving and computational thinking through 'coding': computer programming for beginners..."
- Organized like a fugue, with variations on this theme:
  - Introduce coding constructs in Python,

Syllabus: Topics

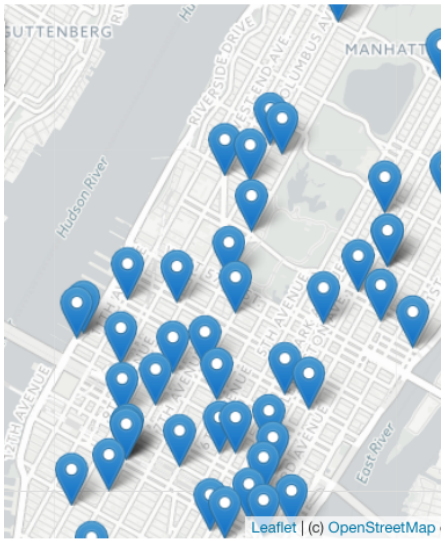


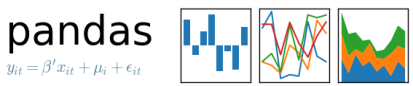


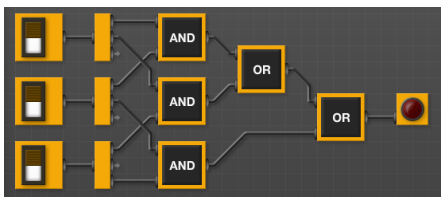


- **This course assumes no previous programming experience.**
- "... Emphasis on problem-solving and computational thinking through 'coding': computer programming for beginners..."
- Organized like a fugue, with variations on this theme:
  - Introduce coding constructs in Python,
  - Apply those ideas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

Syllabus: Topics

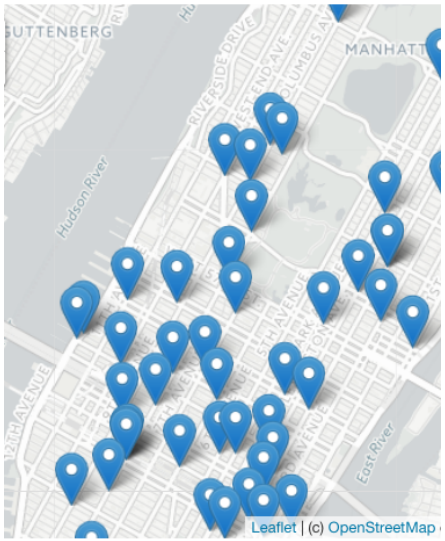


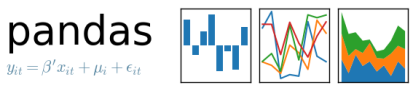


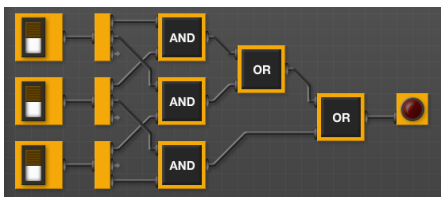


- **This course assumes no previous programming experience.**
- "... Emphasis on problem-solving and computational thinking through 'coding': computer programming for beginners..."
- Organized like a fugue, with variations on this theme:
  - Introduce coding constructs in Python,
  - Apply those ideas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

Syllabus: Topics

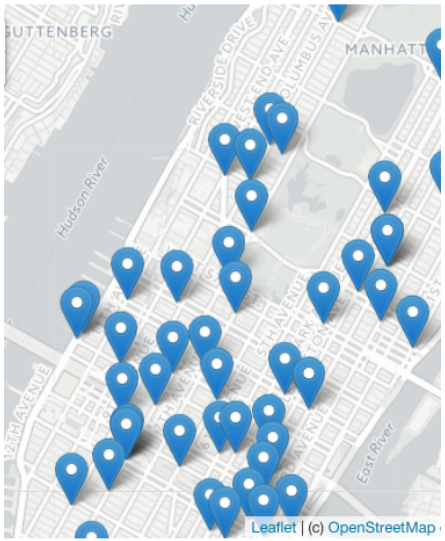


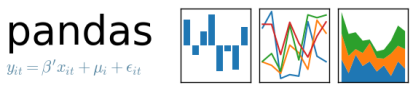


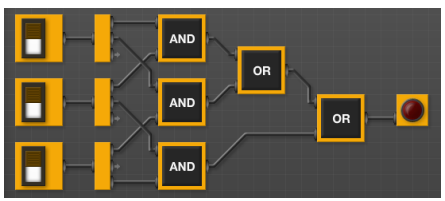


- **This course assumes no previous programming experience.**
- "... Emphasis on problem-solving and computational thinking through 'coding': computer programming for beginners..."
- Organized like a fugue, with variations on this theme:
  - Introduce coding constructs in Python,
  - Apply those ideas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

Syllabus: Topics

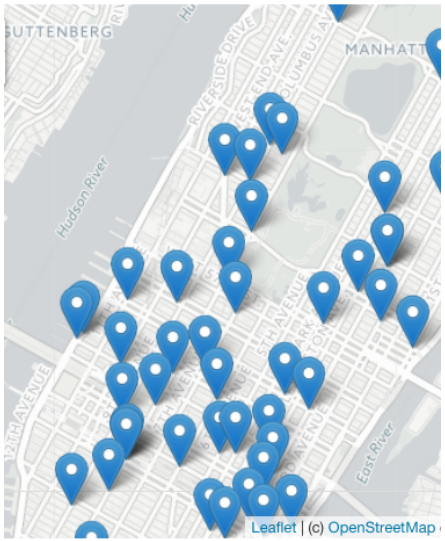


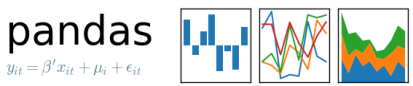


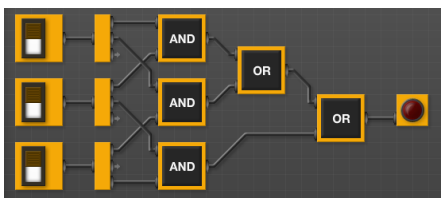


- **This course assumes no previous programming experience.**
- "... Emphasis on problem-solving and computational thinking through 'coding': computer programming for beginners..."
- Organized like a fugue, with variations on this theme:
  - Introduce coding constructs in Python,
  - Apply those ideas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

Syllabus: Topics

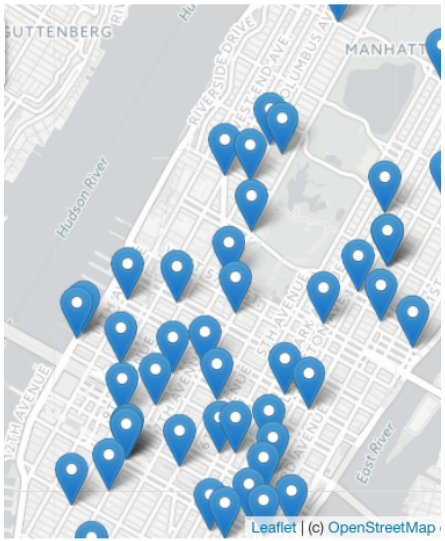


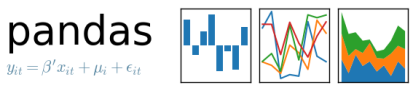


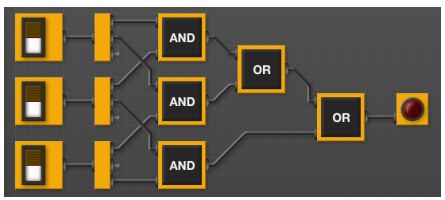


- **This course assumes no previous programming experience.**
- "... Emphasis on problem-solving and computational thinking through 'coding': computer programming for beginners..."
- Organized like a fugue, with variations on this theme:
  - Introduce coding constructs in Python,
  - Apply those ideas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

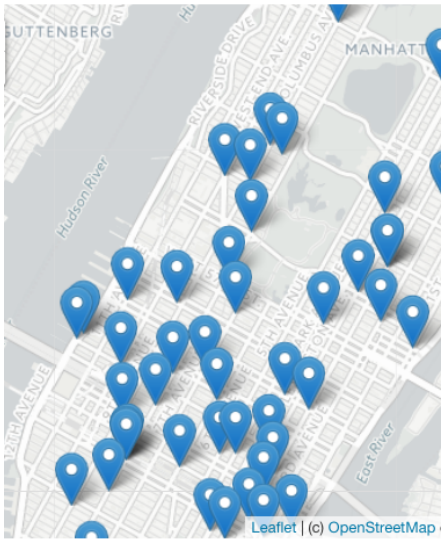


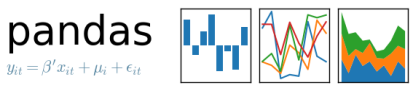


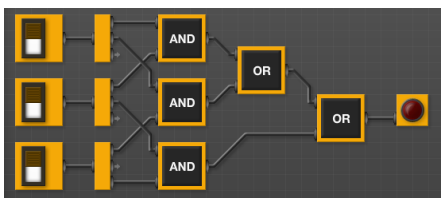


- **This course assumes no previous programming experience.**
- "... Emphasis on problem-solving and computational thinking through 'coding': computer programming for beginners..."
- Organized like a fugue, with variations on this theme:
  - Introduce coding constructs in Python,
  - Apply those ideas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

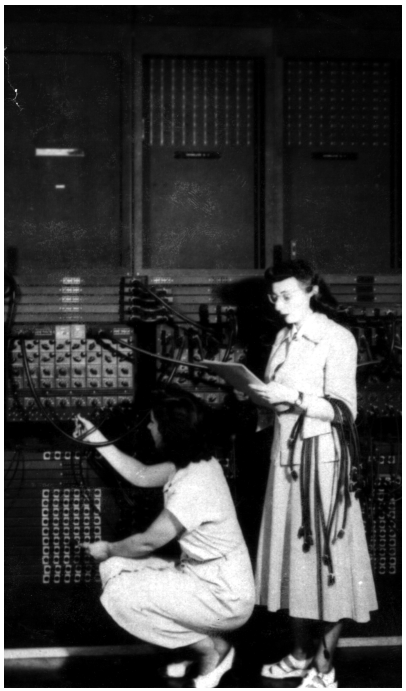






- **This course assumes no previous programming experience.**
- "... Emphasis on problem-solving and computational thinking through 'coding': computer programming for beginners..."
- Organized like a fugue, with variations on this theme:
  - Introduce coding constructs in Python,
  - Apply those ideas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



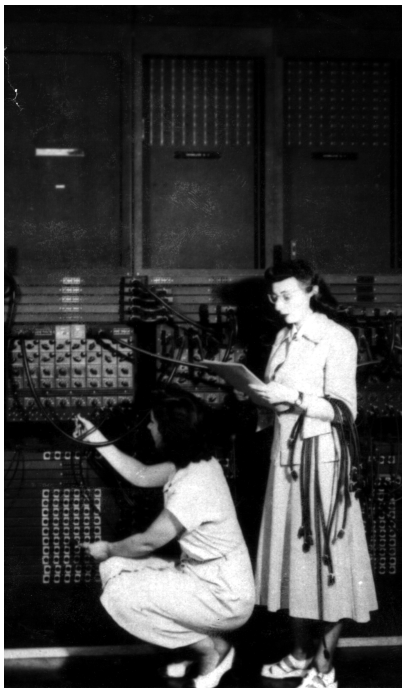
First "computers"

ENIAC, 1945.

Lecture:

- Tuesdays, 9:45-11:00am, on Zoom.

Class Structure



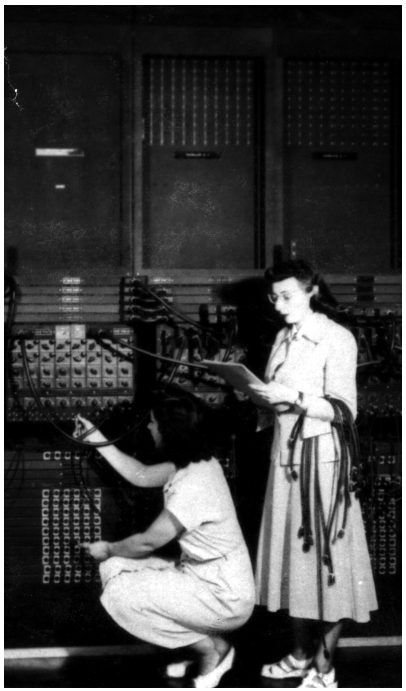
First "computers"

ENIAC, 1945.

Lecture:

- Tuesdays, 9:45-11:00am, on Zoom.
- Mix of explanation, challenges & active concept application.

Class Structure



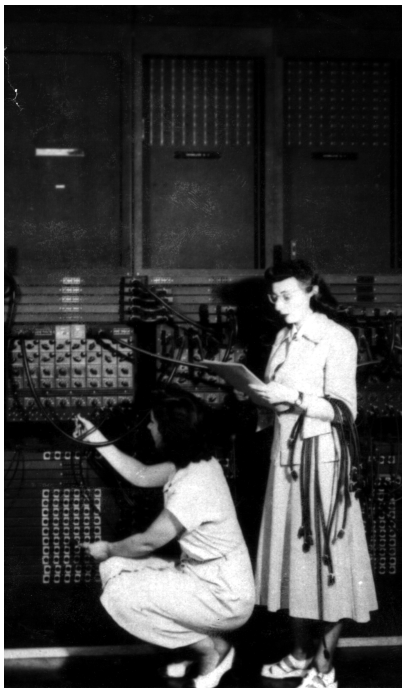
First "computers"

ENIAC, 1945.

Lecture:

- Tuesdays, 9:45-11:00am, on Zoom.
- Mix of explanation, challenges & active concept application.
- Lecture Preview: on Blackboard prior to each lecture.

Class Structure



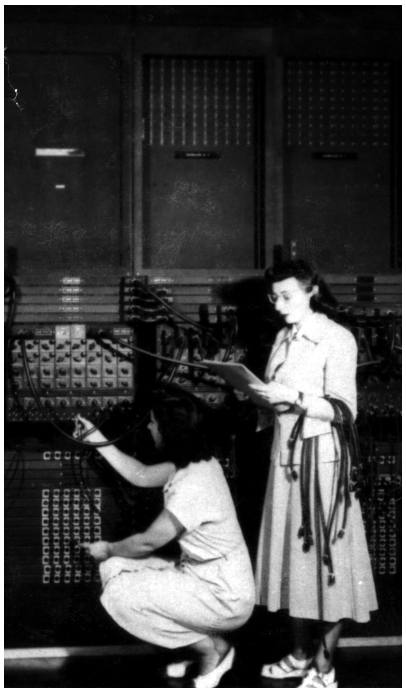
First "computers"

ENIAC, 1945.

Lecture:

- Tuesdays, 9:45-11:00am, on Zoom.
- Mix of explanation, challenges & active concept application.
- Lecture Preview: on Blackboard prior to each lecture.
- Lecture Quiz: on Gradescope during lecture.

Class Structure



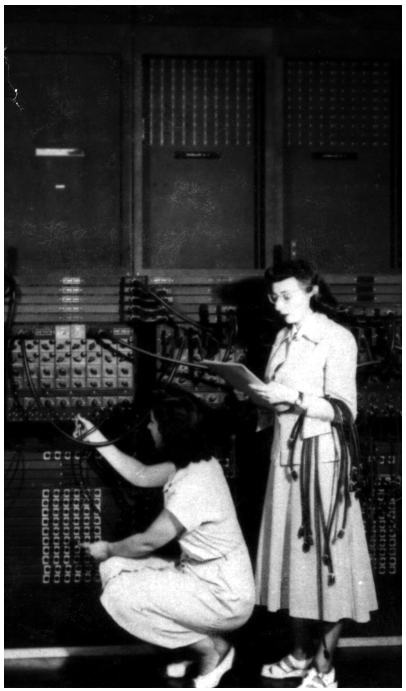
First "computers"

ENIAC, 1945.

Lecture:

- Tuesdays, 9:45-11:00am, on Zoom.
- Mix of explanation, challenges & active concept application.
- Lecture Preview: on Blackboard prior to each lecture.
- Lecture Quiz: on Gradescope during lecture.
- Ask questions in Q&A.

Class Structure



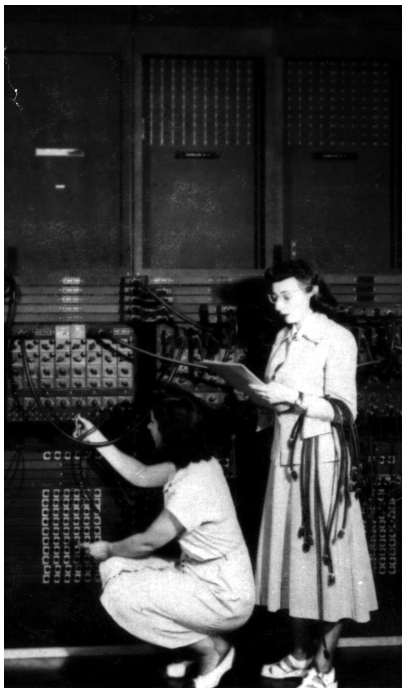
First "computers"

ENIAC, 1945.

Online Labs and Lab Quiz:

- **You must independently read through the weekly online Lab.**

Class Structure



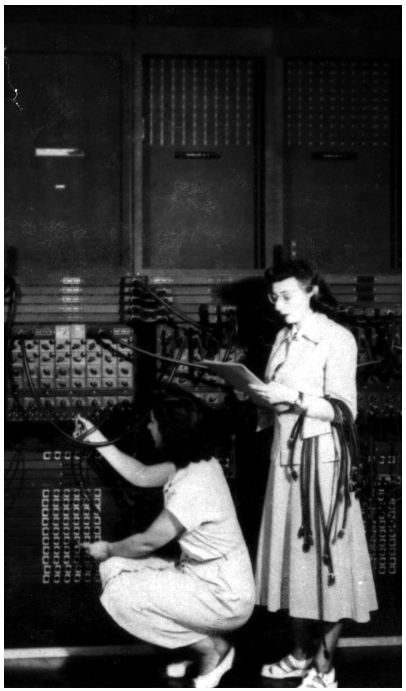
First "computers"

ENIAC, 1945.

Online Labs and Lab Quiz:

- **You must independently read through the weekly online Lab.**
- Set aside about 1 hour.

Class Structure



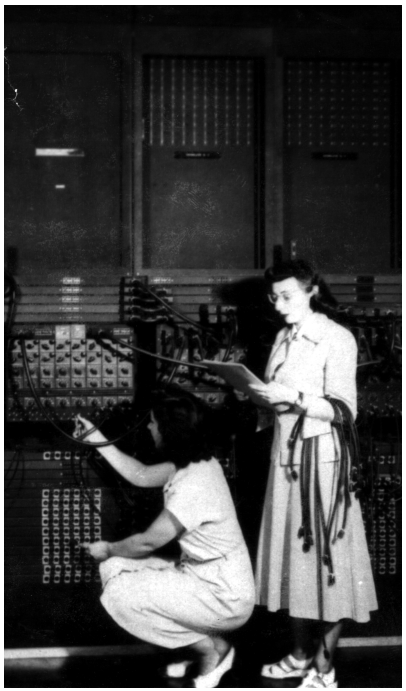
First "computers"

ENIAC, 1945.

Online Labs and Lab Quiz:

- **You must independently read through the weekly online Lab.**
- Set aside about 1 hour.
- Lab content directly supports weekly programming assignments.

Class Structure



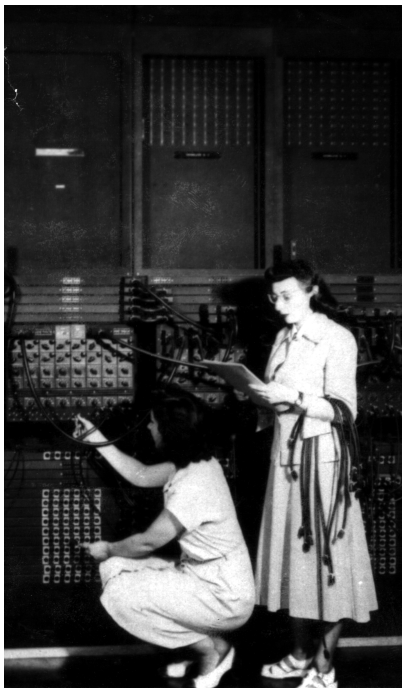
First "computers"

ENIAC, 1945.

Online Labs and Lab Quiz:

- **You must independently read through the weekly online Lab.**
- Set aside about 1 hour.
- Lab content directly supports weekly programming assignments.
- **Lab Quiz** on Gradescope to assess your understanding of the Labs (**Due on Wedesdays 6pm**)

Class Structure



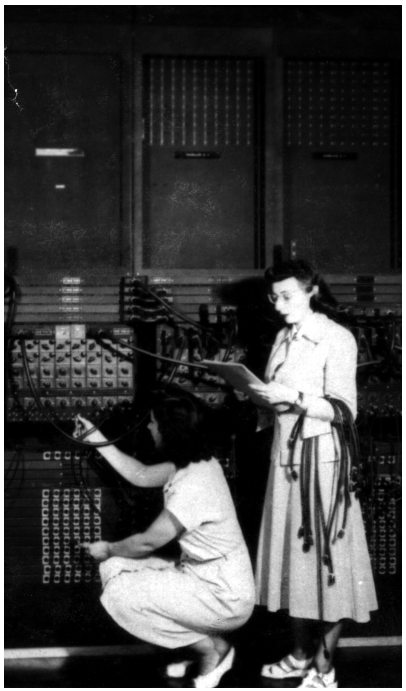
First "computers"

ENIAC, 1945.

Online Labs and Lab Quiz:

- **You must independently read through the weekly online Lab.**
- Set aside about 1 hour.
- Lab content directly supports weekly programming assignments.
- **Lab Quiz** on Gradescope to assess your understanding of the Labs (**Due on Wedesdays 6pm**)
- Labs found on course website (show)

Class Structure

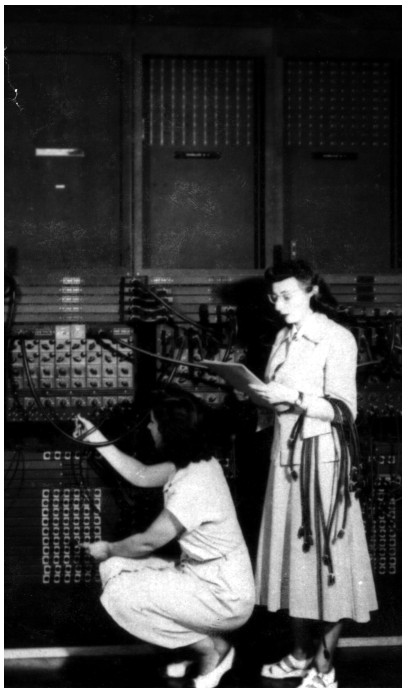


First "computers"

ENIAC, 1945.

Software Platforms:

Class Structure



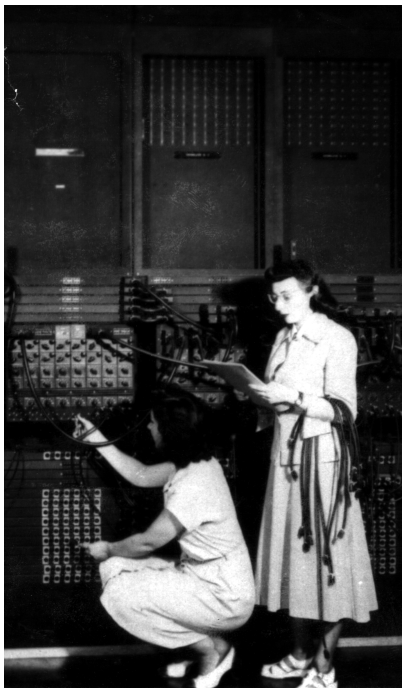
First "computers"

ENIAC, 1945.

Software Platforms:

- Blackboard
  - Important communication sent via Blackboard

Class Structure



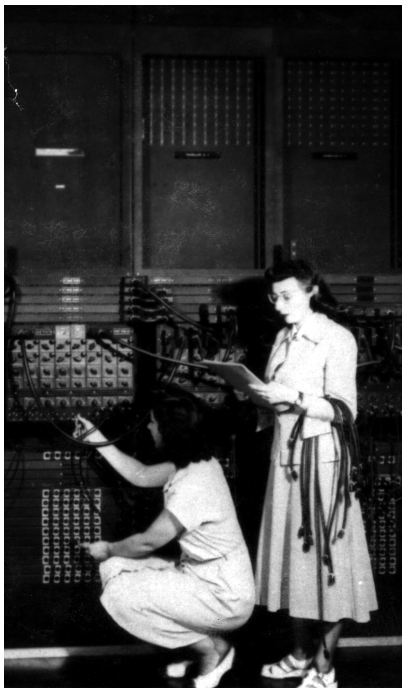
First "computers"

ENIAC, 1945.

Software Platforms:

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

Class Structure



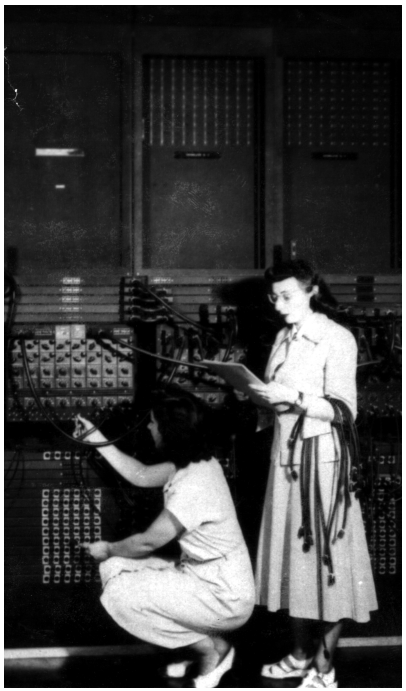
First "computers"

ENIAC, 1945.

Software Platforms:

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

Class Structure



First "computers"

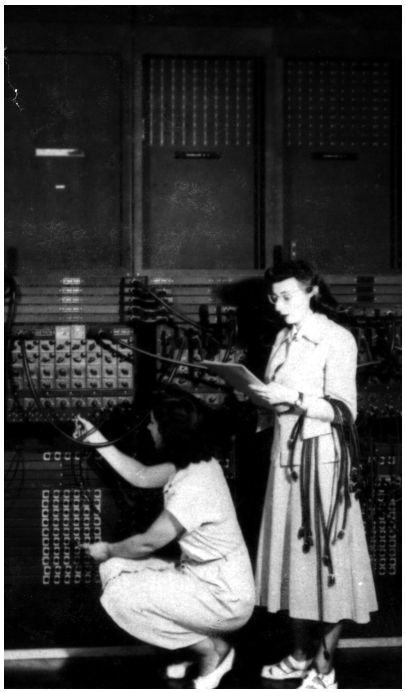
ENIAC, 1945.

Software Platforms:

- Blackboard
  - Important communication sent via Blackboard
  - Check your email account associated with Blackboard
  - Email studenthelpdesk@hunter.cuny.edu if you need to change it
- Gradescope
  - Email invite sent Monday.
  - Match to Blackboard email.

Class Structure

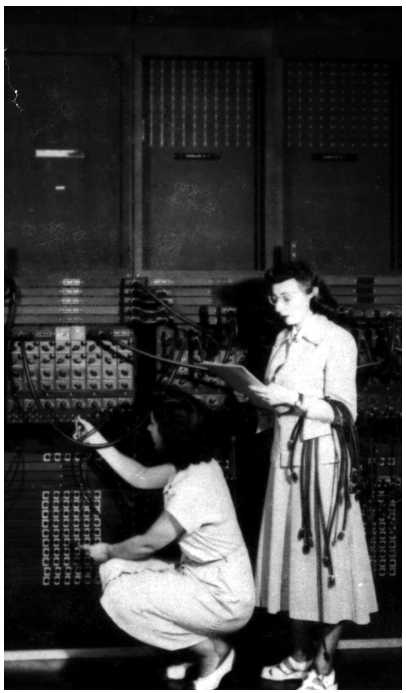
Help:



First "computers"

ENIAC, 1945.

Class Structure



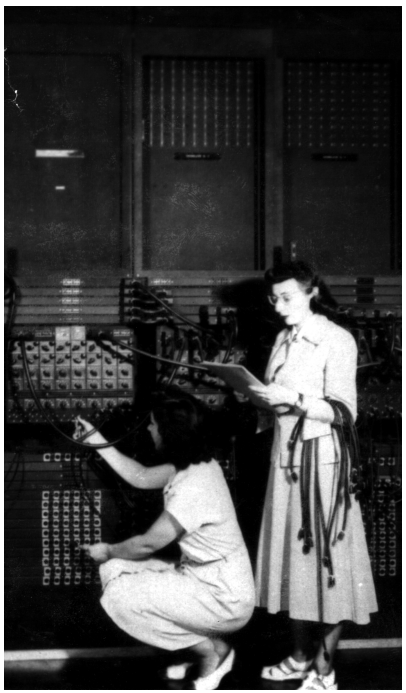
First "computers"

ENIAC, 1945.

Help:

- Peer-mentor Support (UTAs)
  - Drop-in Tutoring: UTA-lead group work to solve programming assignments

Class Structure



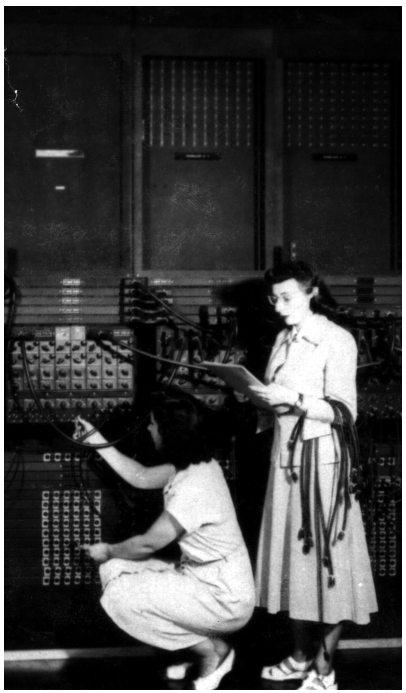
First "computers"

ENIAC, 1945.

Help:

- Peer-mentor Support (UTAs)
  - Drop-in Tutoring: UTA-lead group work to solve programming assignments
  - Drop-in Tutoring access link available on Blackboard under "Synchronous Meetings"

Class Structure



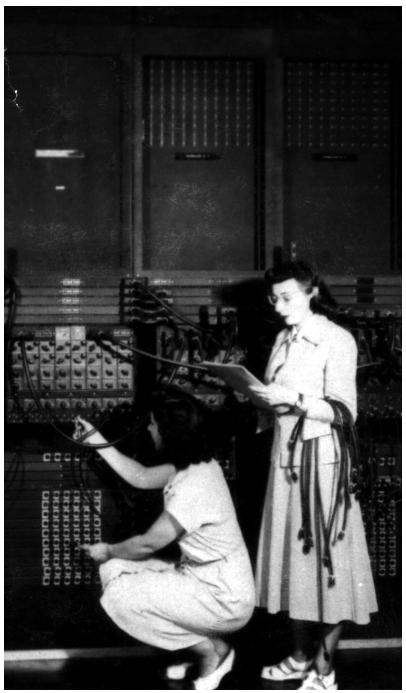
First "computers"

ENIAC, 1945.

Help:

- Peer-mentor Support (UTAs)
  - Drop-in Tutoring: UTA-lead group work to solve programming assignments
  - Drop-in Tutoring access link available on Blackboard under "Synchronous Meetings"
  - Discussion Board on Blackboard

Class Structure



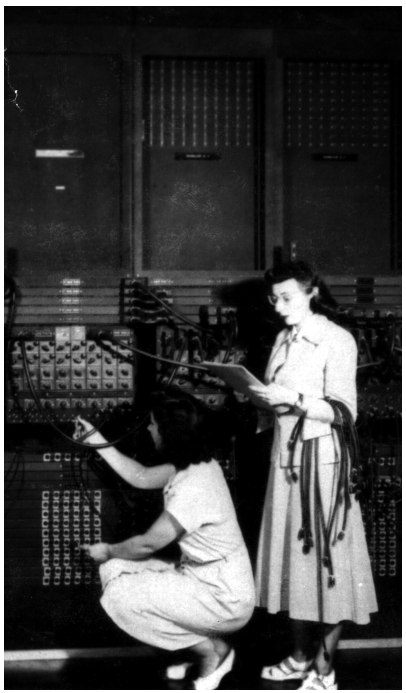
First "computers"

ENIAC, 1945.

Help:

- Peer-mentor Support (UTAs)
  - Drop-in Tutoring: UTA-lead group work to solve programming assignments
  - Drop-in Tutoring access link available on Blackboard under "Synchronous Meetings"
  - Discussion Board on Blackboard
  - Email: huntercsci127help@gmail.com

Class Structure



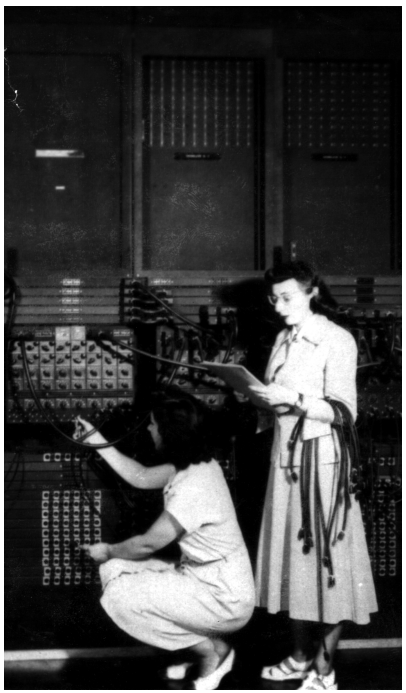
First "computers"

ENIAC, 1945.

Help:

- Peer-mentor Support (UTAs)
  - Drop-in Tutoring: UTA-lead group work to solve programming assignments
  - Drop-in Tutoring access link available on Blackboard under "Synchronous Meetings"
  - Discussion Board on Blackboard
  - Email: huntercsci127help@gmail.com
  - All help available Mo-Fr 11am-5pm when classes are in session

Class Structure



First "computers"

ENIAC, 1945.

Help:

- Peer-mentor Support (UTAs)
  - Drop-in Tutoring: UTA-lead group work to solve programming assignments
  - Drop-in Tutoring access link available on Blackboard under "Synchronous Meetings"
  - Discussion Board on Blackboard
  - Email: huntercsci127help@gmail.com
  - All help available Mo-Fr 11am-5pm when classes are in session
- Office Hours
  - Please email tligorio@hunter.cuny.edu to make an appointment

How to Succeed in this Course

- Come to Lecture

How to Succeed in this Course

- Come to Lecture
  - Do the lecture preview.

How to Succeed in this Course

- Come to Lecture
  - Do the lecture preview.
  - Pay attention during lecture.

How to Succeed in this Course

- Come to Lecture
  - Do the lecture preview.
  - Pay attention during lecture.
  - Actively participate in lecture work: try to solve problems/challenges

How to Succeed in this Course

- Come to Lecture
  - Do the lecture preview.
  - Pay attention during lecture.
  - Actively participate in lecture work: try to solve problems/challenges
- Read the Online Labs.

How to Succeed in this Course

- Come to Lecture
  - Do the lecture preview.
  - Pay attention during lecture.
  - Actively participate in lecture work: try to solve problems/challenges
- Read the Online Labs.
- Take the weekly Lab Quiz.

How to Succeed in this Course

- Come to Lecture
  - Do the lecture preview.
  - Pay attention during lecture.
  - Actively participate in lecture work: try to solve problems/challenges
- Read the Online Labs.
- Take the weekly Lab Quiz.
- Work ahead on Programming Assignments.

How to Succeed in this Course

- Come to Lecture
  - Do the lecture preview.
  - Pay attention during lecture.
  - Actively participate in lecture work: try to solve problems/challenges
- Read the Online Labs.
- Take the weekly Lab Quiz.
- Work ahead on Programming Assignments.
- Ask for help from our UTAs in Drop-in Tutoring or Discussion Board.

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 quizzes, lecture previews and lab quizzes with your final exam grade.
  Lecture quizzes, previews and lab quizzes only help your grade.

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 quizzes, lecture previews and lab quizzes with your final exam grade.
  Lecture quizzes, previews and lab quizzes only help your grade.
- Do I have to pass the final to pass the course?

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 quizzes, lecture previews and lab quizzes with your final exam grade.
  Lecture quizzes, previews and lab quizzes only help your grade.
- Do I have to pass the final to pass the course?
  Yes. *To demonstrate mastery, you must pass the final exam.*

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 quizzes, lecture previews and lab quizzes with your final exam grade.
  Lecture quizzes, previews and lab quizzes only help your grade.
- Do I have to pass the final to pass the course?
  Yes. *To demonstrate mastery, you must pass the final exam.*
  We will end most lectures with past final exam questions and review.

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 (particularly those new to programming) 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 (particularly those new to programming) 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 (particularly those new to programming) 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 (particularly those new to programming) 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 (particularly those new to programming) 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 (in the form of challenges during lecture)? 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 (particularly those new to programming) 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 (in the form of challenges during lecture)? 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.
  Actively applying concepts introduced in lecture 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 (particularly those new to programming) 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 (in the form of challenges during lecture)? 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.
  Actively applying concepts introduced in lecture increases student performance.
- I like working by myself. Why do I have to work in groups during drop-in tutoring?

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 (particularly those new to programming) 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 (in the form of challenges during lecture)? 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.
  Actively applying concepts introduced in lecture increases student performance.
- I like working by myself. Why do I have to work in groups during drop-in tutoring?
  Active group work and discussion 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 (particularly those new to programming) 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 (in the form of challenges during lecture)? 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.
  Actively applying concepts introduced in lecture increases student performance.
- I like working by myself. Why do I have to work in groups during drop-in tutoring?
  Active group work and discussion increases student performance.
  Also, it 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 master the concepts in this course?

Philosophy (Or Why We Do What We Do)

Help:

- What's the best way to master the concepts in this course?
  - *Most efficient way: do the programs*

Philosophy (Or Why We Do What We Do)

Help:

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

Philosophy (Or Why We Do What We Do)

Help:

- What's the best way to master the concepts in this course?
  - *Most efficient way: do the programs (more programs on old finals).*
  - *Aim to complete programs the week related ideas are covered in lab.*
  - *If you want to dig deeper there's reading and tutorials on webpage for each week.*

Philosophy (Or Why We Do What We Do)

Help:

- What's the best way to master the concepts in this course?
  - *Most efficient way: do the programs (more programs on old finals).*
  - *Aim to complete programs the week related ideas are covered in lab.*
  - *If you want to dig deeper there's reading and tutorials on webpage for each week.*
- If stuck:

Philosophy (Or Why We Do What We Do)

Help:

- What's the best way to master the concepts in this course?
  - *Most efficient way: do the programs (more programs on old finals).*
  - *Aim to complete programs the week related ideas are covered in lab.*
  - *If you want to dig deeper there's reading and tutorials on webpage for each week.*
- If stuck:
  - *Review lecture notes (pdf on webpage) and Online Lab*

Philosophy (Or Why We Do What We Do)

Help:

- What's the best way to master the concepts in this course?
  - *Most efficient way: do the programs (more programs on old finals).*
  - *Aim to complete programs the week related ideas are covered in lab.*
  - *If you want to dig deeper there's reading and tutorials on webpage for each week.*
- If stuck:
  - *Review lecture notes (pdf on webpage) and Online Lab*
  - *Still stuck? Ask for help from our UTAs:*

Philosophy (Or Why We Do What We Do)

Help:

- What's the best way to master the concepts in this course?
  - *Most efficient way: do the programs (more programs on old finals).*
  - *Aim to complete programs the week related ideas are covered in lab.*
  - *If you want to dig deeper there's reading and tutorials on webpage for each week.*
- If stuck:
  - *Review lecture notes (pdf on webpage) and Online Lab*
  - *Still stuck? Ask for help from our UTAs:*
    - ★ *Drop-in Tutoring: UTA-lead group work to solve programming assignments*

Philosophy (Or Why We Do What We Do)

Help:

- What's the best way to master the concepts in this course?
  - *Most efficient way: do the programs (more programs on old finals).*
  - *Aim to complete programs the week related ideas are covered in lab.*
  - *If you want to dig deeper there's reading and tutorials on webpage for each week.*
- If stuck:
  - *Review lecture notes (pdf on webpage) and Online Lab*
  - *Still stuck? Ask for help from our UTAs:*
    - ★ *Drop-in Tutoring: UTA-lead group work to solve programming assignments*
    - ★ *Drop-in Tutoring access link available on Blackboard under "Synchronous Meetings"*

Philosophy (Or Why We Do What We Do)

Help:

- What's the best way to master the concepts in this course?
  - *Most efficient way: do the programs (more programs on old finals).*
  - *Aim to complete programs the week related ideas are covered in lab.*
  - *If you want to dig deeper there's reading and tutorials on webpage for each week.*
- If stuck:
  - *Review lecture notes (pdf on webpage) and Online Lab*
  - *Still stuck? Ask for help from our UTAs:*
    - ★ *Drop-in Tutoring: UTA-lead group work to solve programming assignments*
    - ★ *Drop-in Tutoring access link available on Blackboard under "Synchronous Meetings"*
    - ★ *Post questions on Blackboard Discussion Board*

Philosophy (Or Why We Do What We Do)

Help:

- What's the best way to master the concepts in this course?
  - *Most efficient way: do the programs (more programs on old finals).*
  - *Aim to complete programs the week related ideas are covered in lab.*
  - *If you want to dig deeper there's reading and tutorials on webpage for each week.*
- If stuck:
  - *Review lecture notes (pdf on webpage) and Online Lab*
  - *Still stuck? Ask for help from our UTAs:*
    - ★ *Drop-in Tutoring: UTA-lead group work to solve programming assignments*
    - ★ *Drop-in Tutoring access link available on Blackboard under "Synchronous Meetings"*
    - ★ *Post questions on Blackboard Discussion Board*
    - ★ *Email questions to huntercsci127help@gmail.com*

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 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).

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!`
- 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,')
```

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,')
```

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.

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)

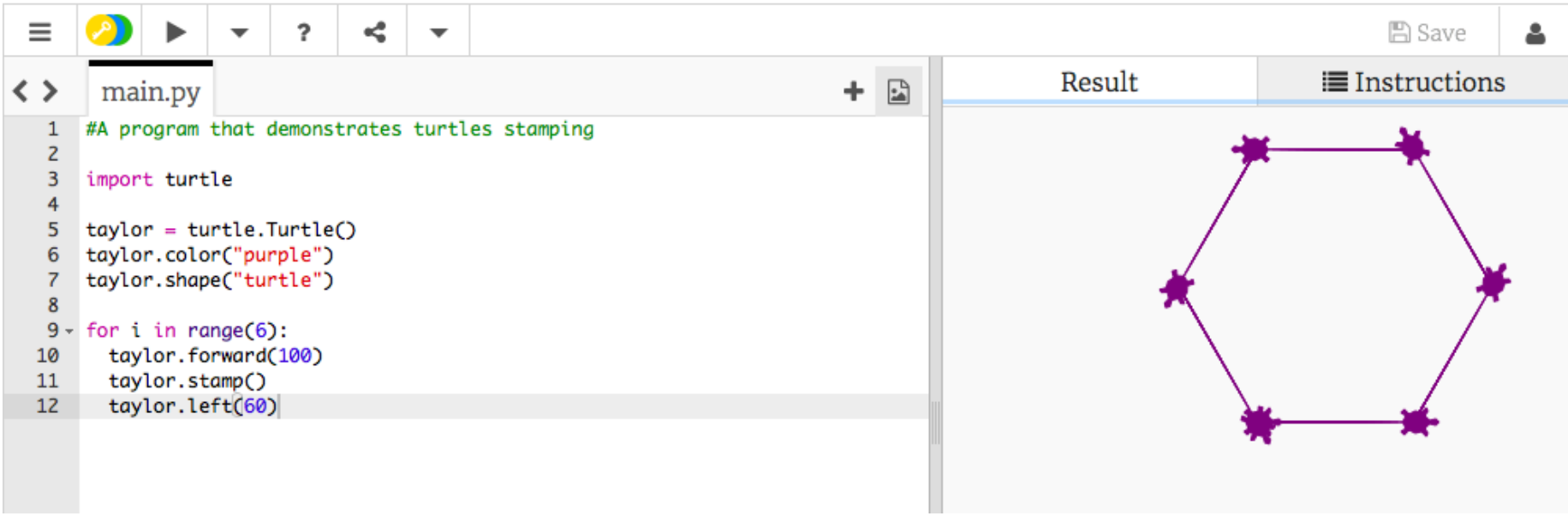
Turtles Introduction

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

Today's Topics

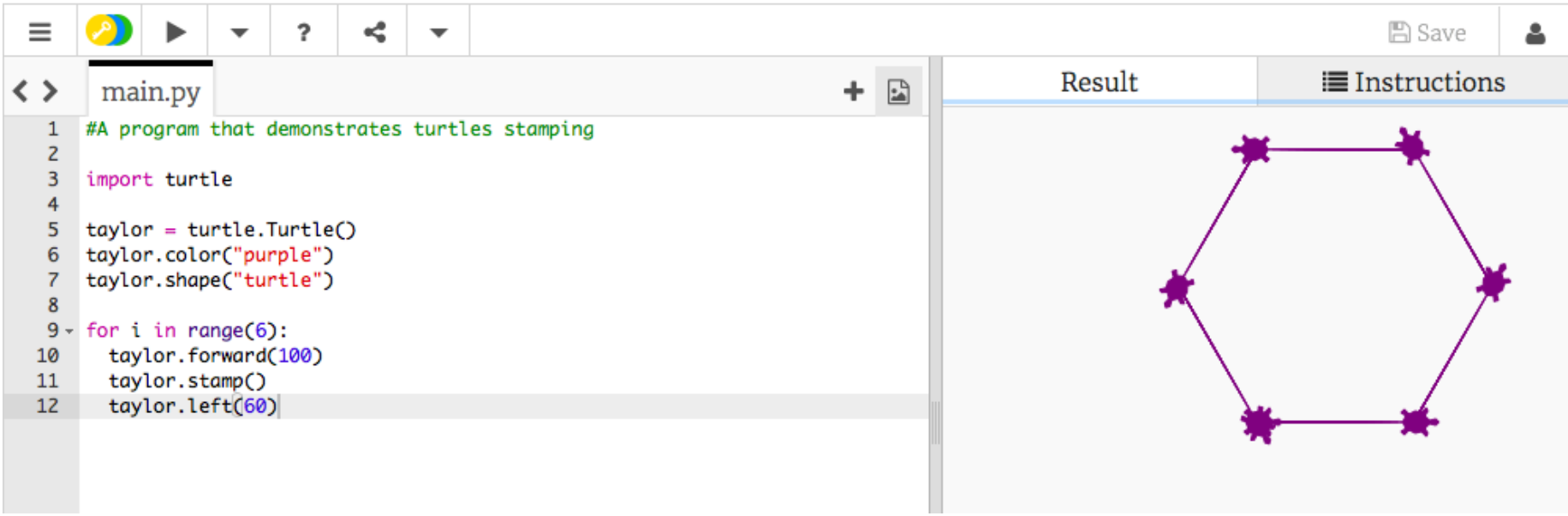
- Introduction to Python
- Turtle Graphics
- **Definite Loops (`for`-loops)**
- Algorithms

Turtles Introduction



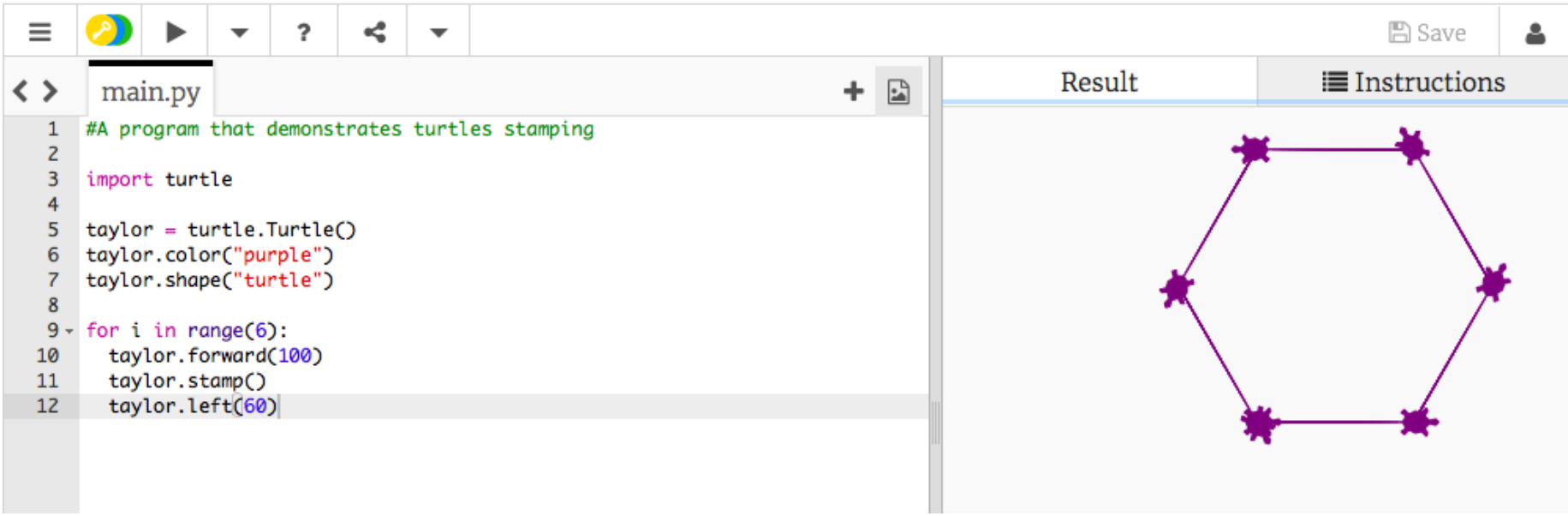
- Creates a turtle **variable**, called `taylor`.

Turtles Introduction



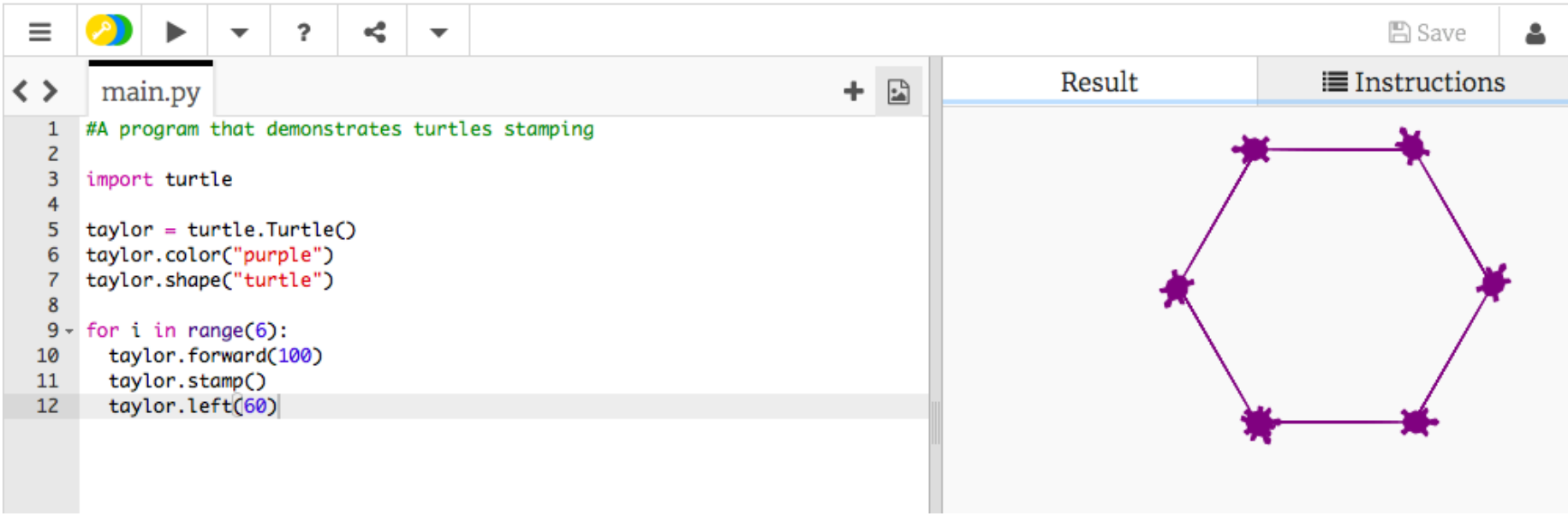
- Creates a turtle **variable**, called `taylor`.
- Changes the color (to purple) and shape (to turtle-shaped).

Turtles Introduction



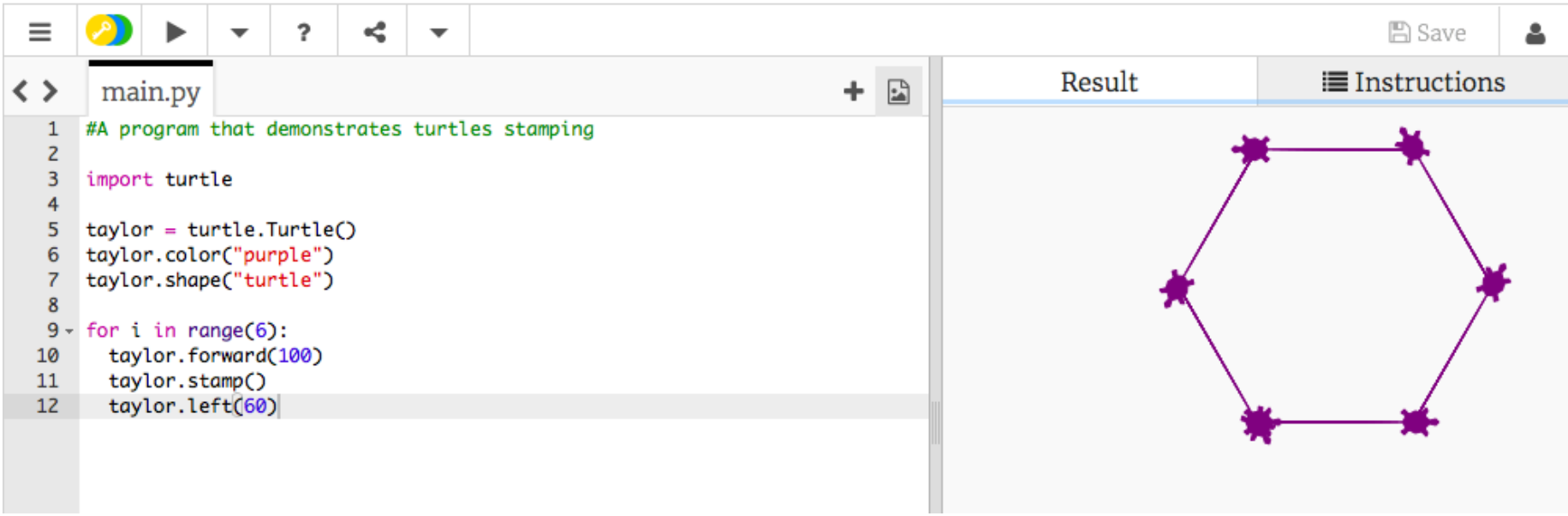
- Creates a turtle **variable**, called `taylor`.
- Changes the color (to purple) and shape (to turtle-shaped).
- Repeats 6 times:

Turtles Introduction



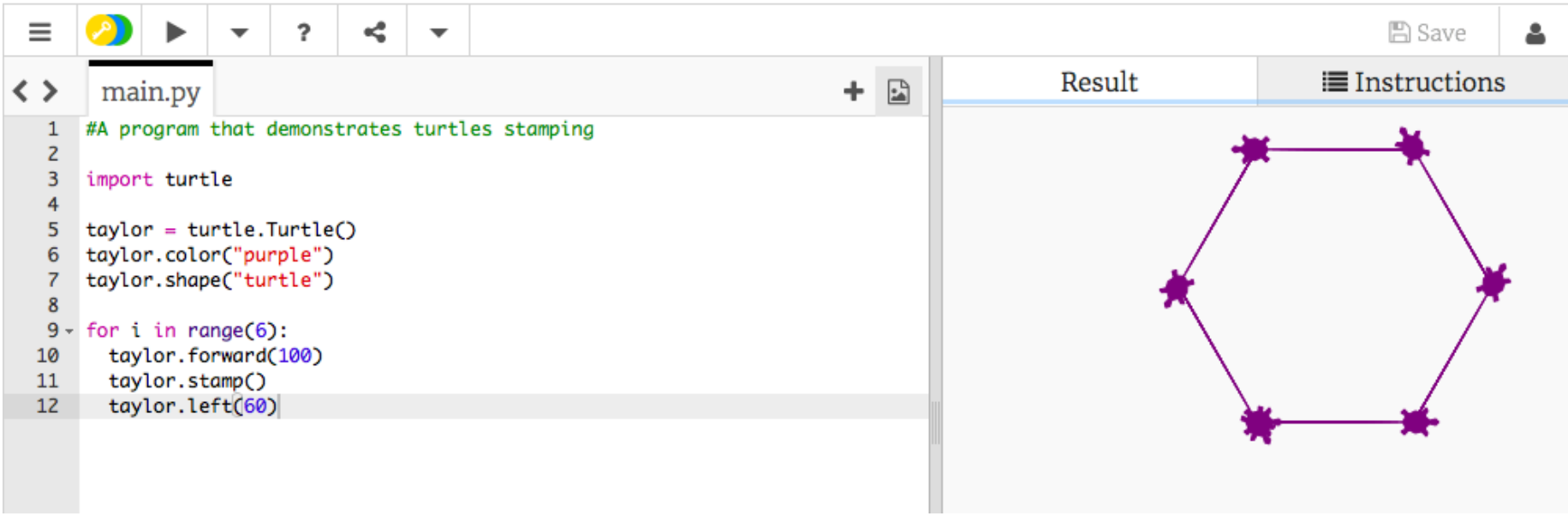
- Creates a turtle **variable**, called `taylor`.
- Changes the color (to purple) and shape (to turtle-shaped).
- Repeats 6 times:
  - Move forward; stamp; and turn left 60 degrees.

Turtles Introduction



- 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



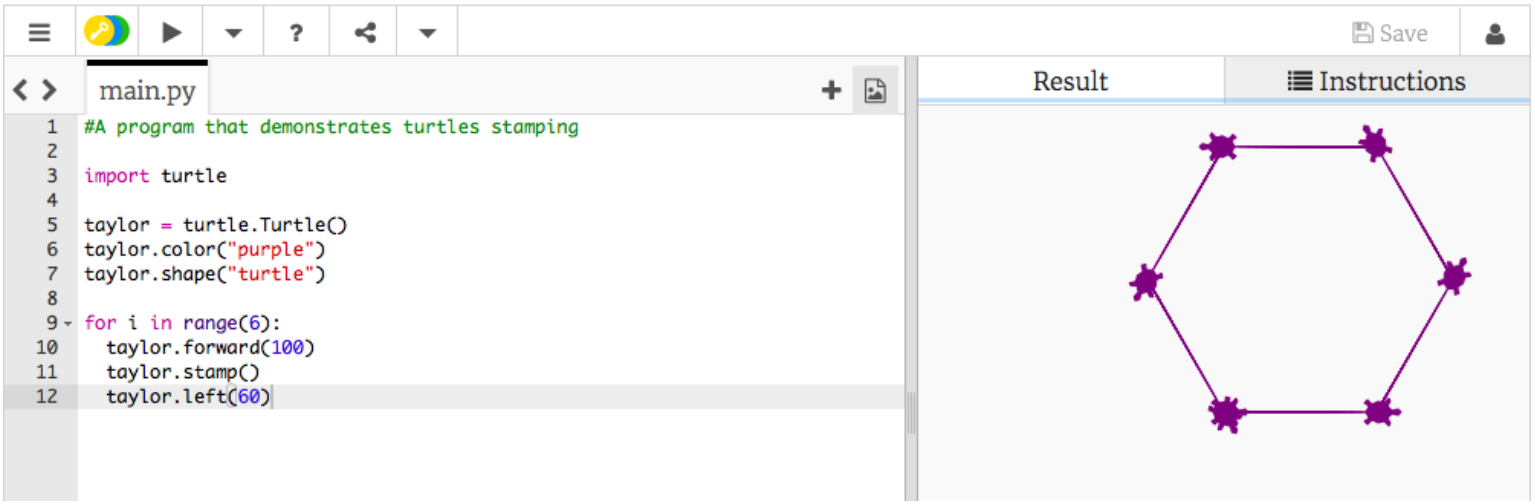
- Creates a turtle **variable**, called `taylor`.
- Changes the color (to purple) and shape (to turtle-shaped).
- Repeats 6 times:
  - Move forward; stamp; and turn left 60 degrees.
- Repeats any instructions **indented** in the "loop block"
- This is a **definite** loop because it repeats a fixed number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

Your Turn!!!

Try to solve this challenge:

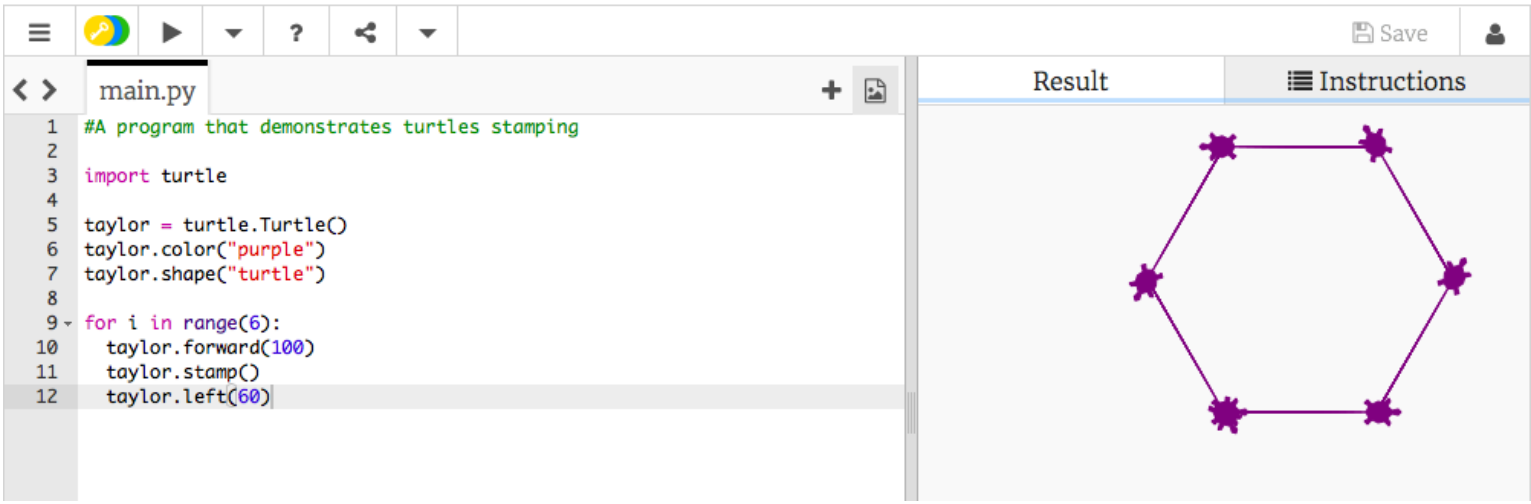
1. Write a program that will draw a 10-sided polygon.
2. Write a program that will repeat the line:
   `I'm lookin' for a mind at work!`
   three times.

Decagon Program



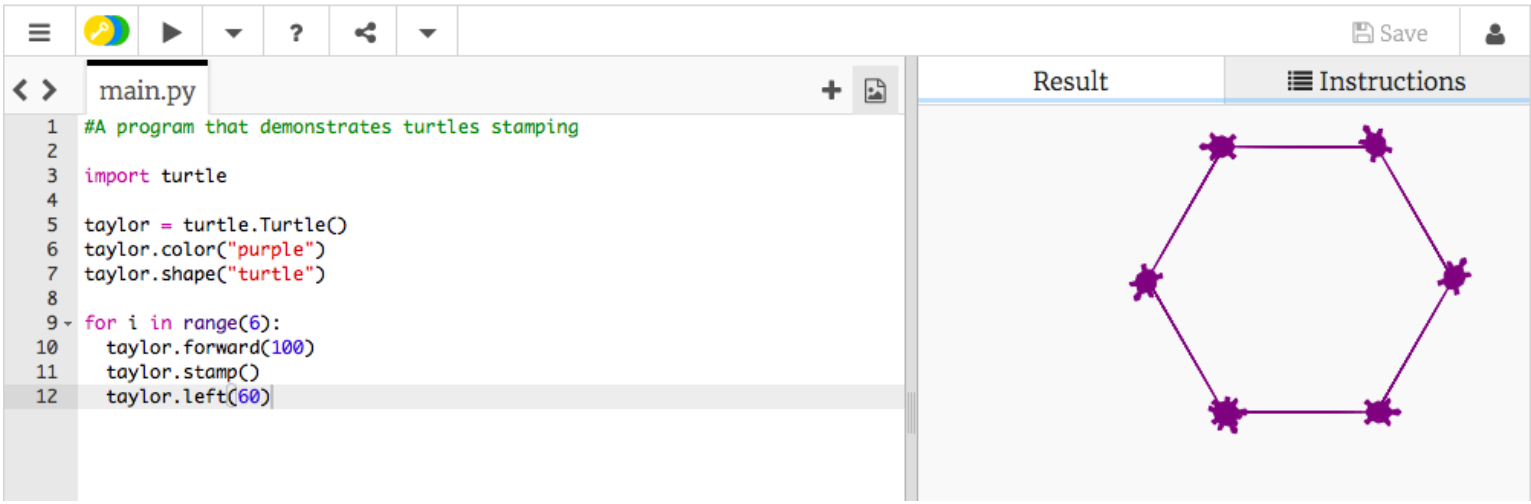
- Start with the hexagon program.

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!")
```

Lecture Quiz

Log-in to Gradescope

- Find Lecture 1 Quiz

Lecture Quiz

Log-in to Gradescope

- Find Lecture 1 Quiz
- Take the quiz

Lecture Quiz

Log-in to Gradescope

- Find Lecture 1 Quiz
- Take the quiz
- You have 3 minutes

Today's Topics

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

What is an Algorithm?

From our textbook:

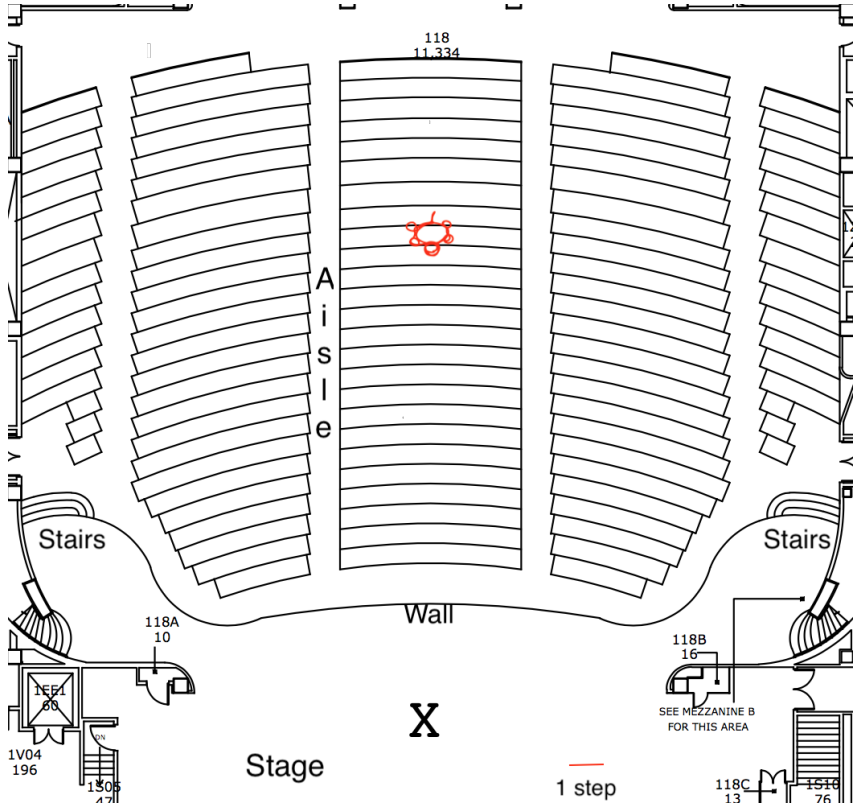
- An algorithm is a process or set of steps to be followed to solve a problem.

What is an Algorithm?

From our textbook:

- An algorithm is a process or set 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.

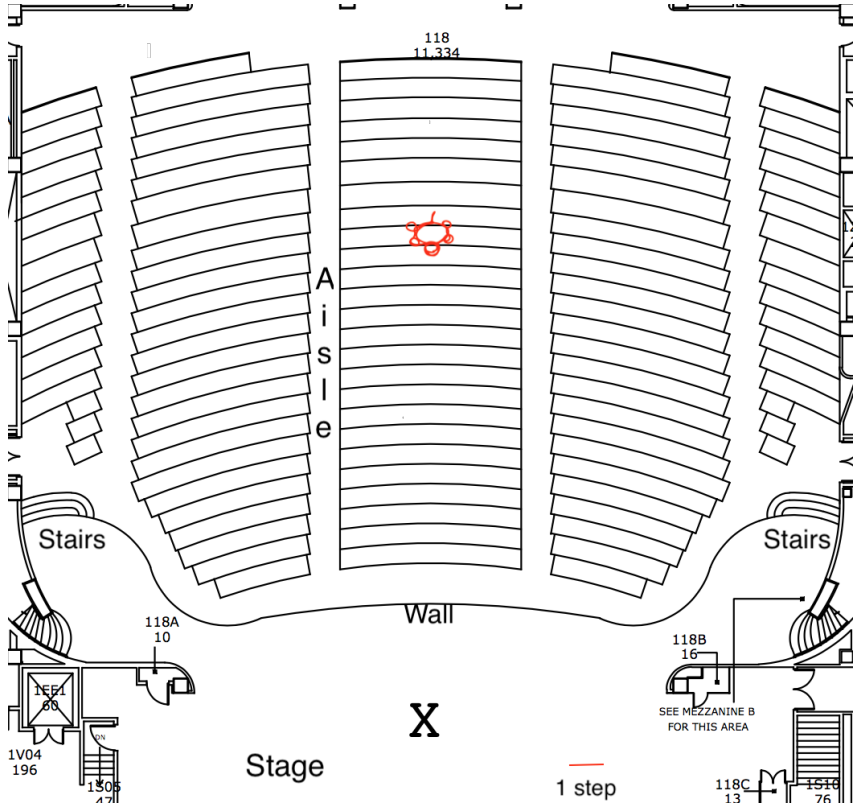
Your Turn!!!



Try to solve this challenge:

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

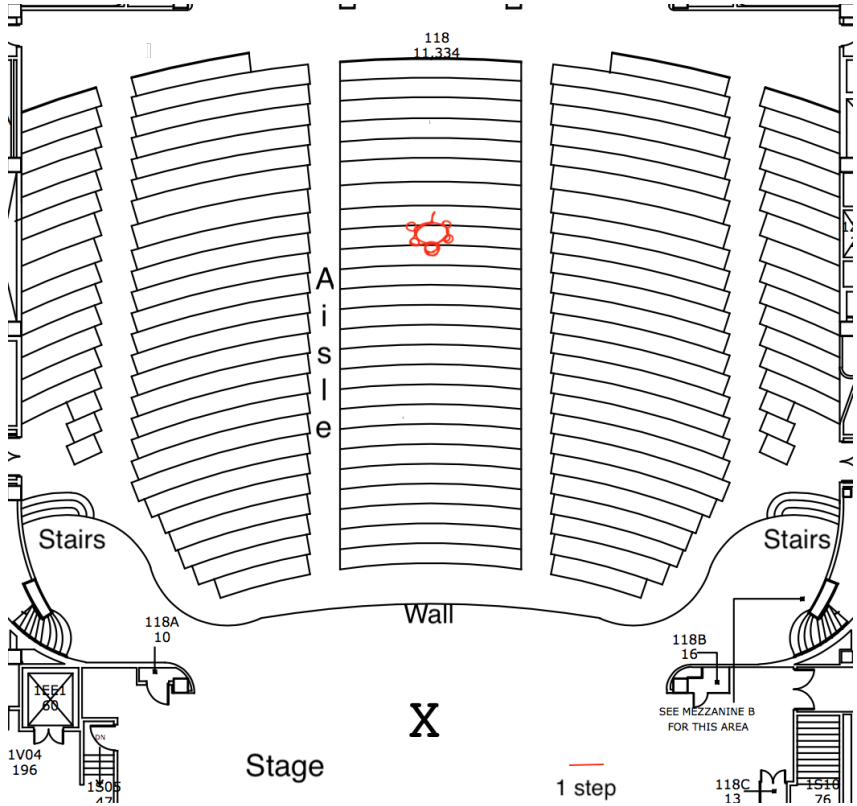
Your Turn!!!



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` on Stage.

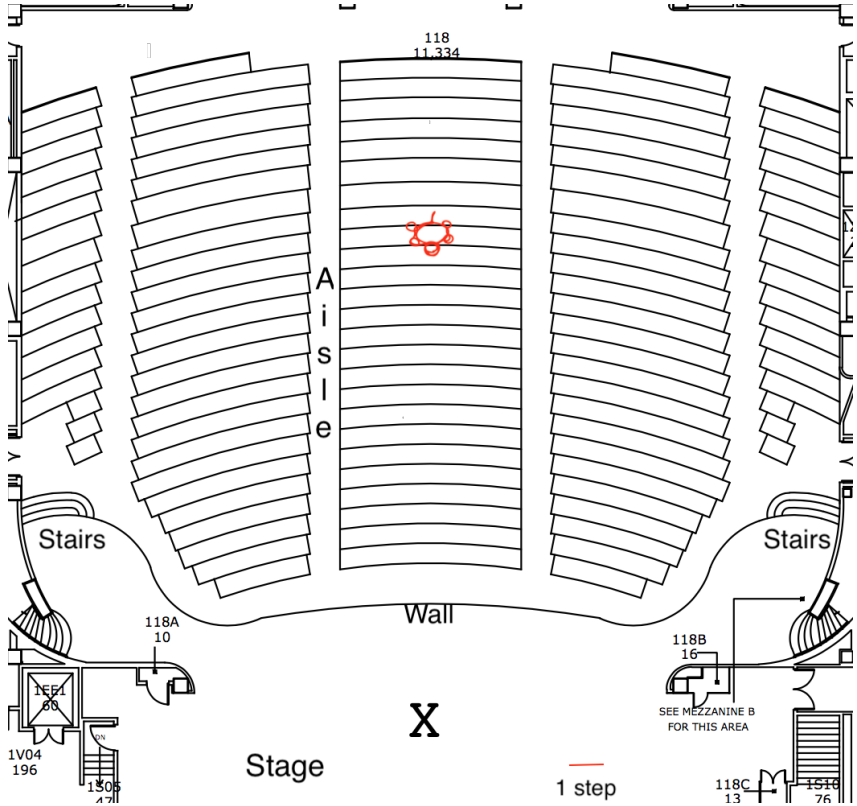
Your Turn!!!



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` on Stage.
3. Basic Rules:

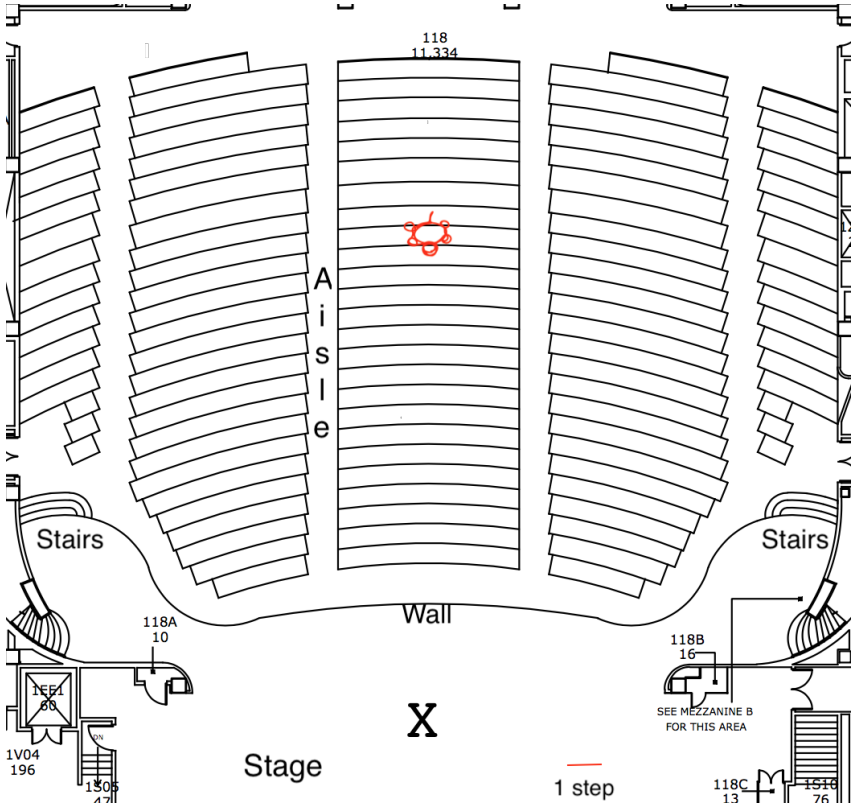
Your Turn!!!



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` on Stage.
3. Basic Rules:
   - Use `turtle` commands.

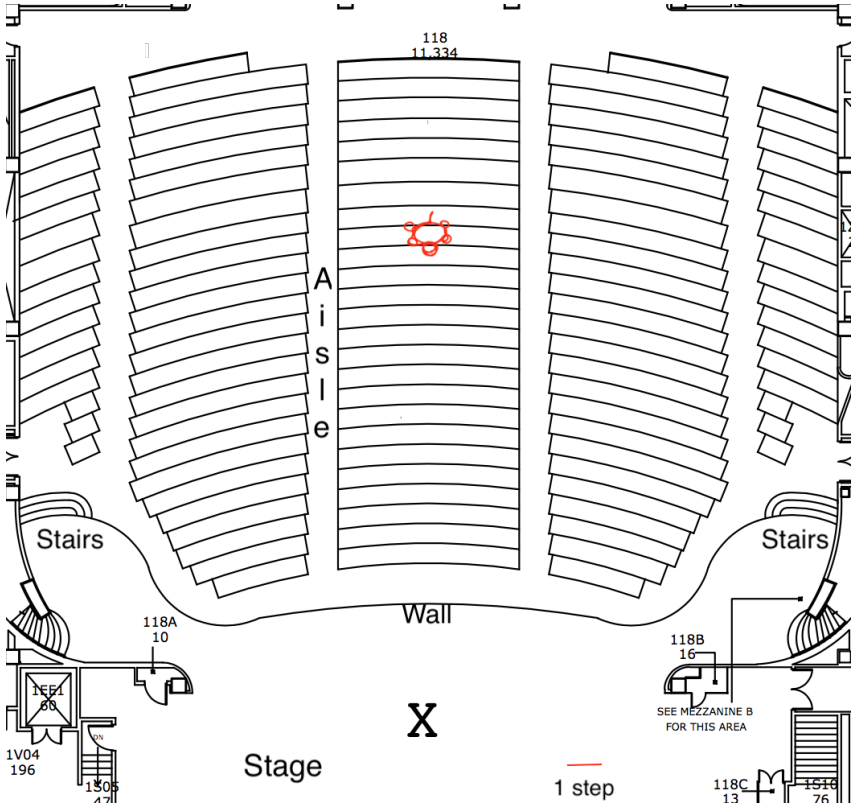
Your Turn!!!



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` on Stage.
3. Basic Rules:
   - Use `turtle` commands.
   - Do not run turtles into walls, chairs, obstacles, etc.

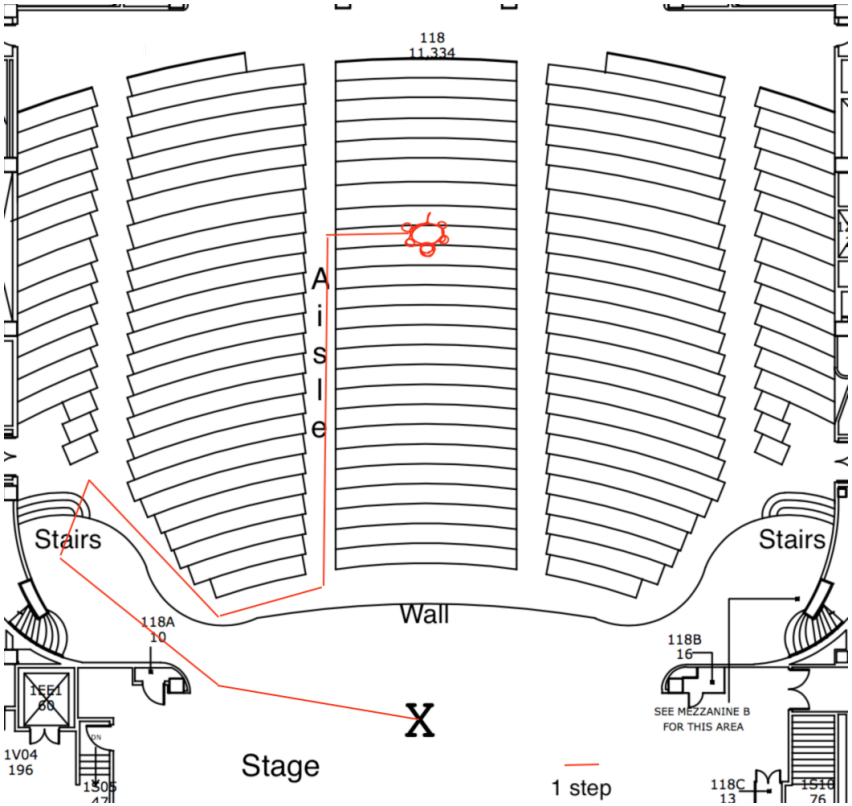
Your Turn!!!



Try to solve this challenge:

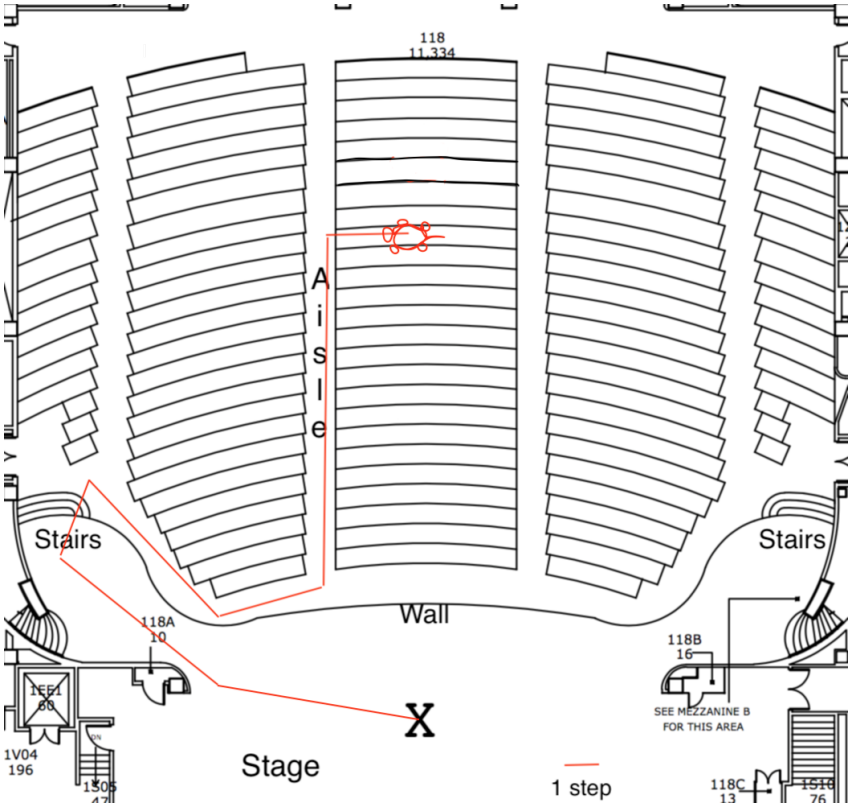
1. This is the floor plan of Assembly Hall at Hunter College.
2. Write an algorithm (step-by-step directions) to the red turtle to the `X` on Stage.
3. Basic Rules:
   - Use `turtle` commands.
   - Do not run turtles into walls, chairs, obstacles, etc.
   - Turtles cannot climb walls, must use stairs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

Your Turn!!!



One possible solution:

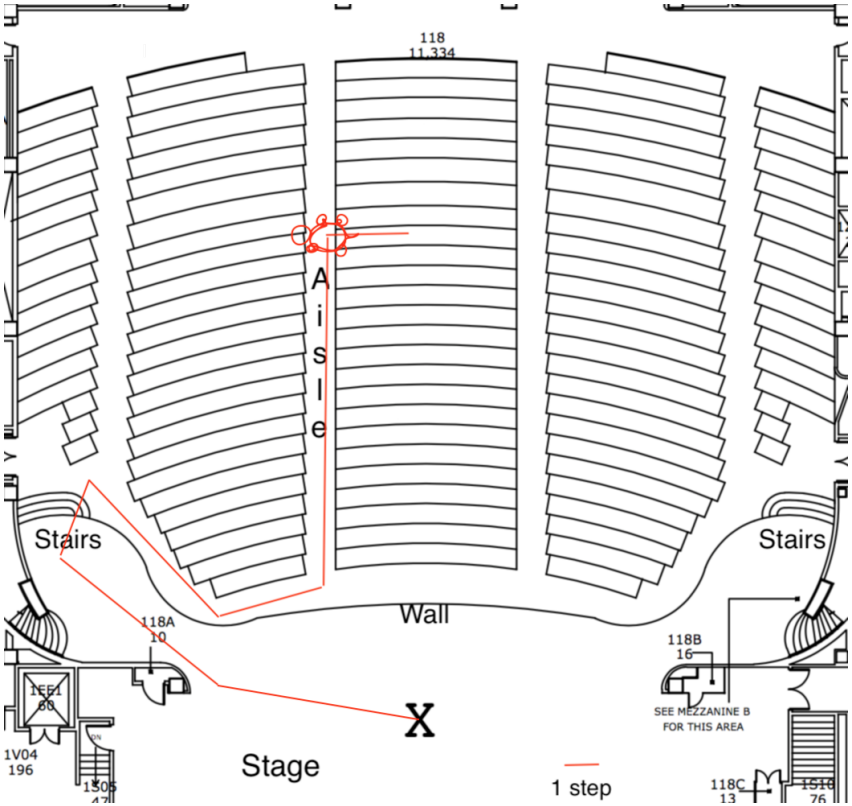
Your Turn!!!



One possible solution:

- Turn right 90 degrees.

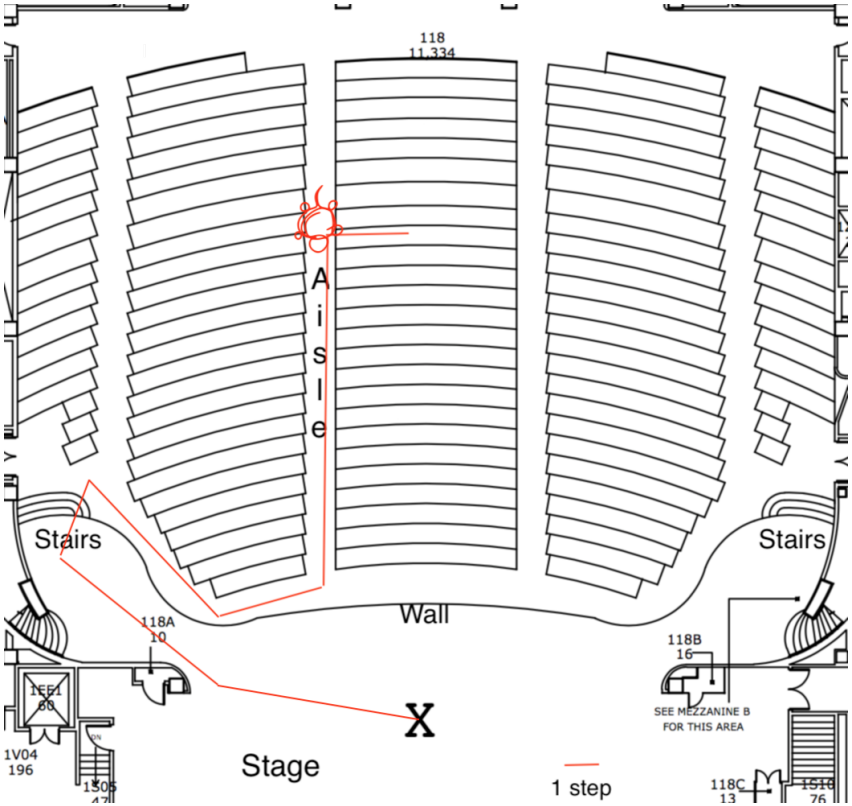
Your Turn!!!



- Turn right 90 degrees.
- Walk forward 3 steps.

One possible solution:

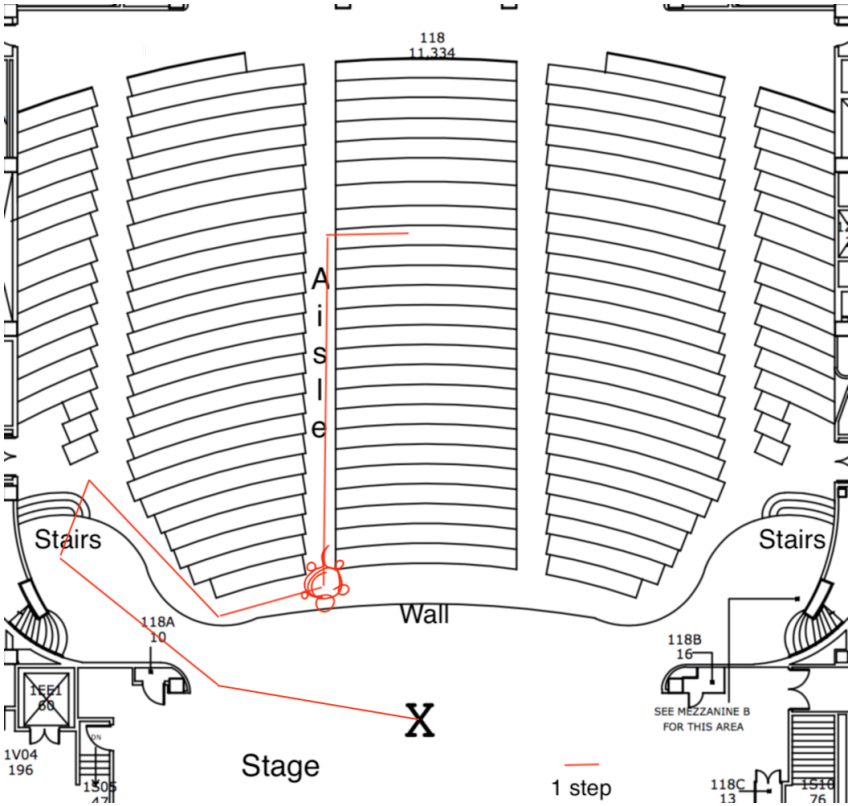
Your Turn!!!



- Turn right 90 degrees.
- Walk forward 3 steps.
- Turn left 90 degrees.

One possible solution:

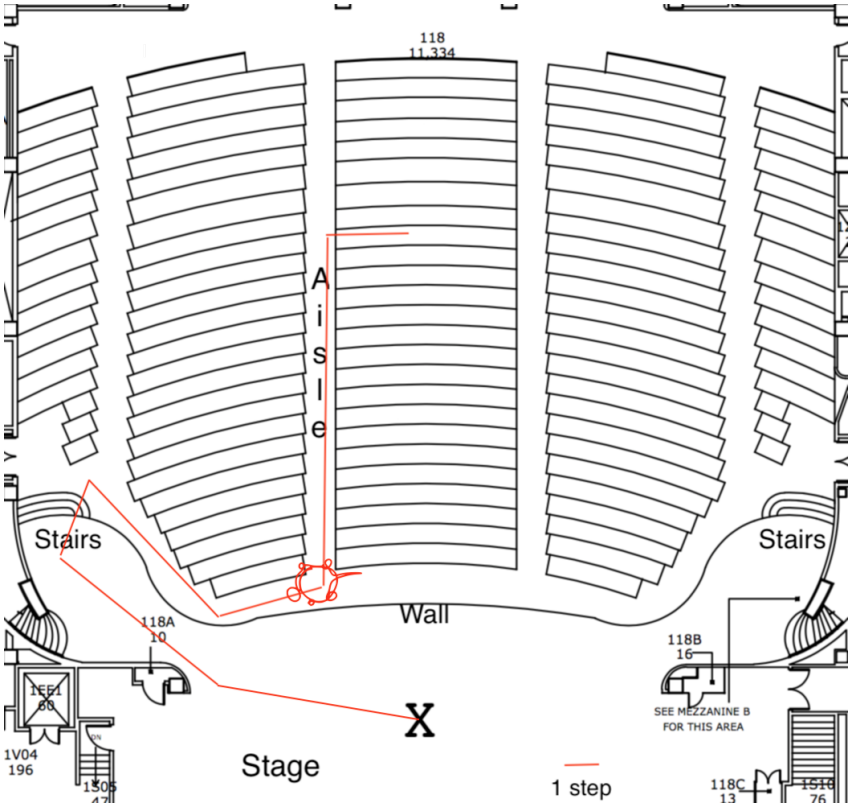
Your Turn!!!



- Turn right 90 degrees.
- Walk forward 3 steps.
- Turn left 90 degrees.
- Walk forward 10 steps.

One possible solution:

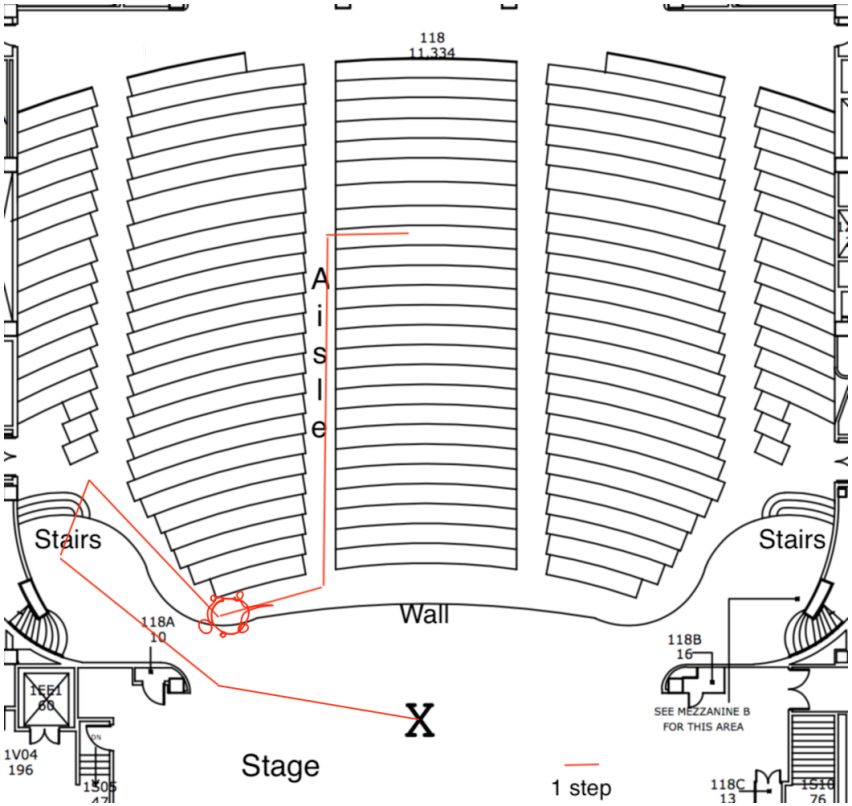
Your Turn!!!



- Turn right 90 degrees.
- Walk forward 3 steps.
- Turn left 90 degrees.
- Walk forward 10 steps.
- Turn right 65 degrees

One possible solution:

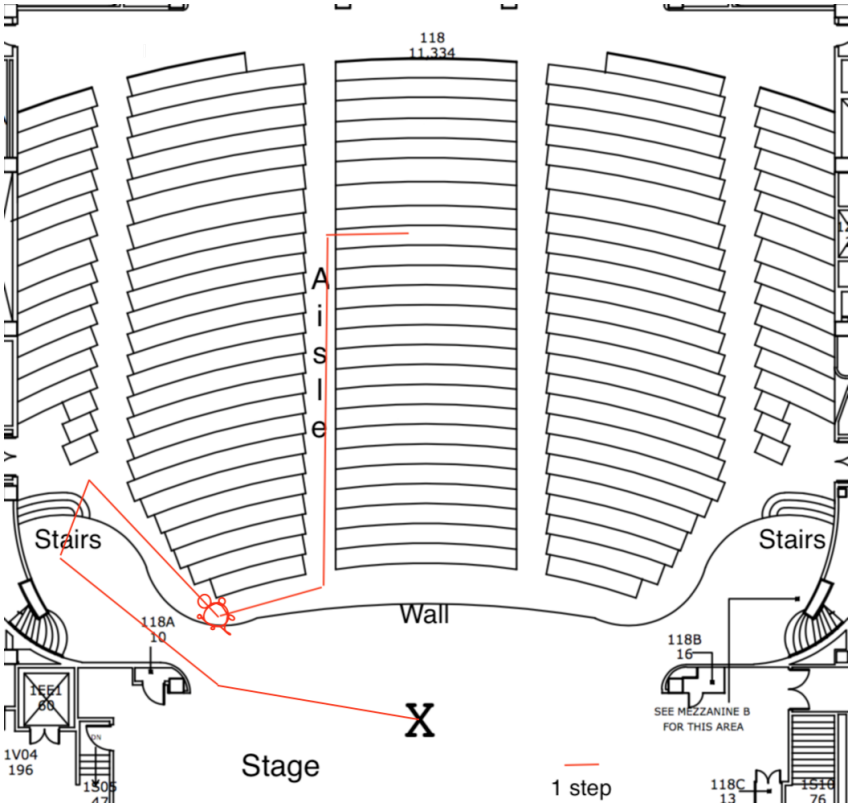
Your Turn!!!



One possible solution:

- Turn right 90 degrees.
- Walk forward 3 steps.
- Turn left 90 degrees.
- Walk forward 10 steps.
- Turn right 65 degrees.
- Walk forward 4 steps.

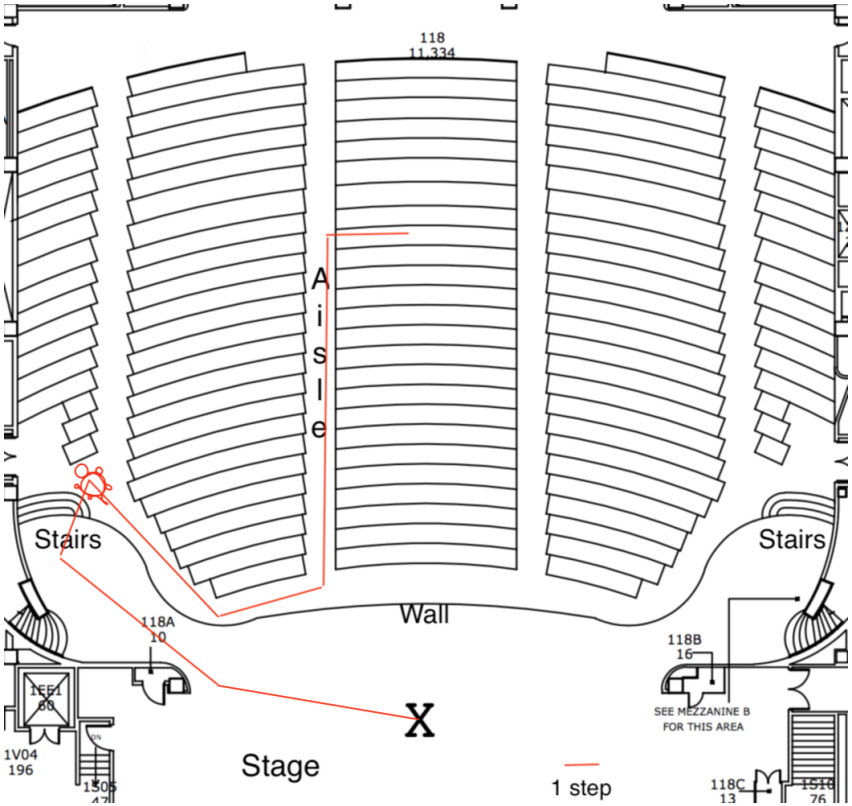
Your Turn!!!



- Turn right 90 degrees.
- Walk forward 3 steps.
- Turn left 90 degrees.
- Walk forward 10 steps.
- Turn right 65 degrees.
- Walk forward 4 steps.
- Turn right 45 degrees.

One possible solution:

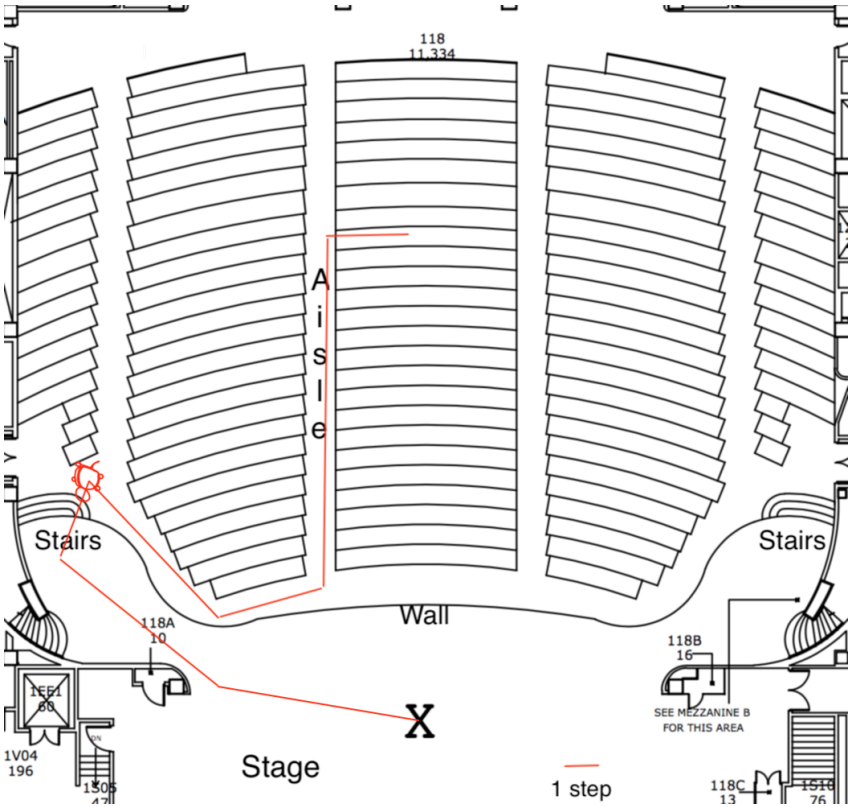
Your Turn!!!



- Turn right 90 degrees.
- Walk forward 3 steps.
- Turn left 90 degrees.
- Walk forward 10 steps.
- Turn right 65 degrees.
- Walk forward 4 steps.
- Turn right 45 degrees.
- Walk forward 6 steps.

One possible solution:

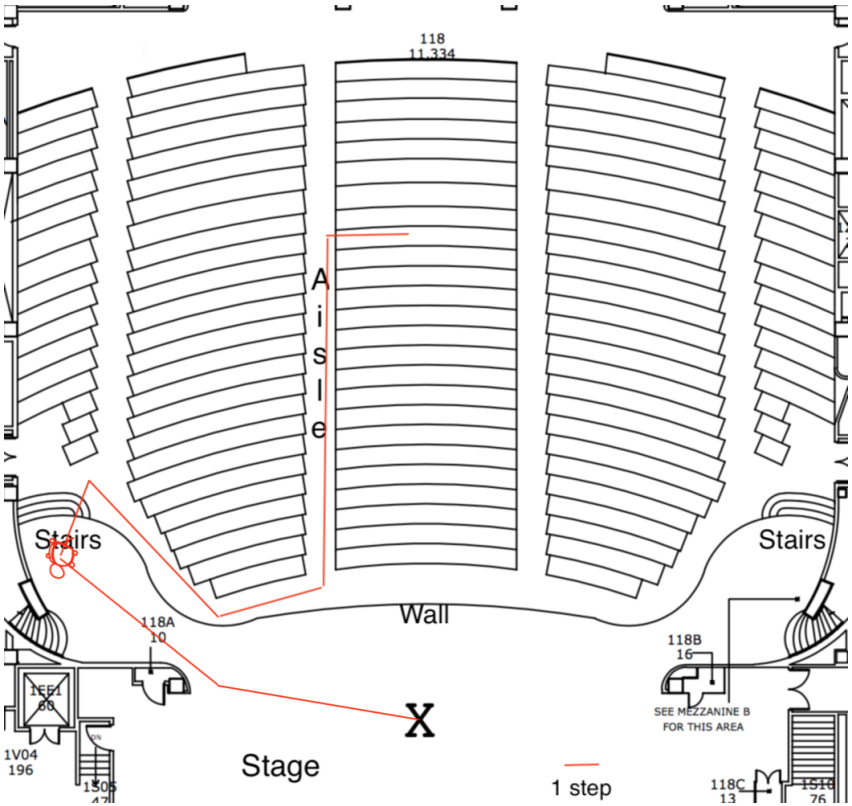
Your Turn!!!



One possible solution:

- Turn right 90 degrees.
- Walk forward 3 steps.
- Turn left 90 degrees.
- Walk forward 10 steps.
- Turn right 65 degrees.
- Walk forward 4 steps.
- Turn right 45 degrees.
- Walk forward 6 steps.
- Turn left 110 degrees.

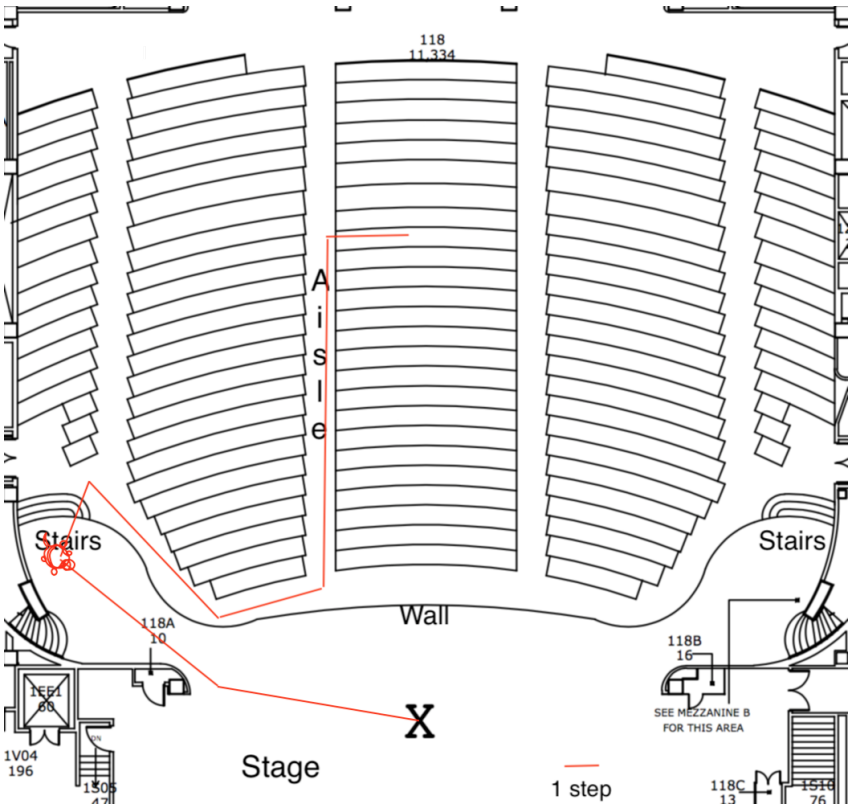
Your Turn!!!



One possible solution:

- Turn right 90 degrees.
- Walk forward 3 steps.
- Turn left 90 degrees.
- Walk forward 10 steps.
- Turn right 65 degrees.
- Walk forward 4 steps.
- Turn right 45 degrees.
- Walk forward 6 steps.
- Turn left 110 degrees.
- Walk forward 3 steps.

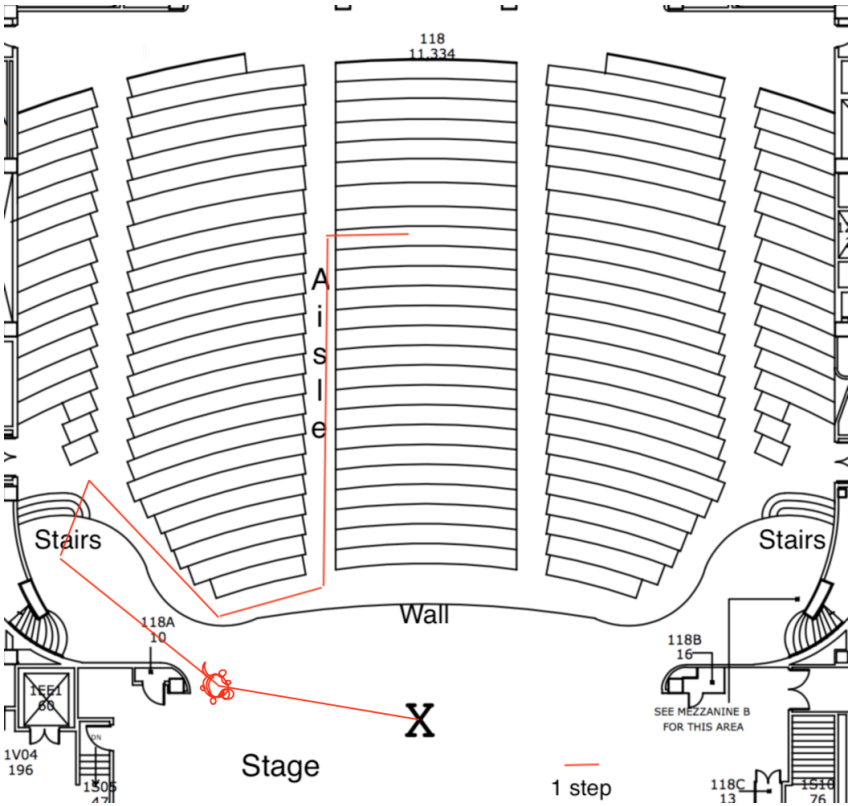
Your Turn!!!



One possible solution:

- Turn right 90 degrees.
- Walk forward 3 steps.
- Turn left 90 degrees.
- Walk forward 10 steps.
- Turn right 65 degrees.
- Walk forward 4 steps.
- Turn right 45 degrees.
- Walk forward 6 steps.
- Turn left 110 degrees.
- Walk forward 3 steps.
- Turn left 80 degrees.

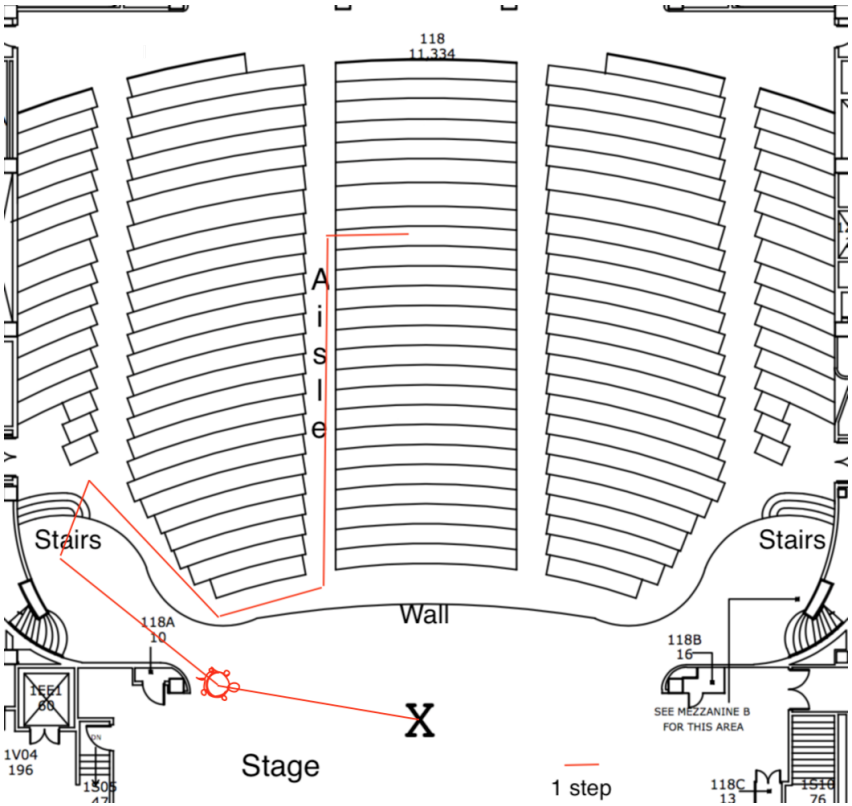
Your Turn!!!



One possible solution:

- Turn right 90 degrees.
- Walk forward 3 steps.
- Turn left 90 degrees.
- Walk forward 10 steps.
- Turn right 65 degrees.
- Walk forward 4 steps.
- Turn right 45 degrees.
- Walk forward 6 steps.
- Turn left 110 degrees.
- Walk forward 3 steps.
- Turn left 80 degrees.
- Walk forward 5 steps.

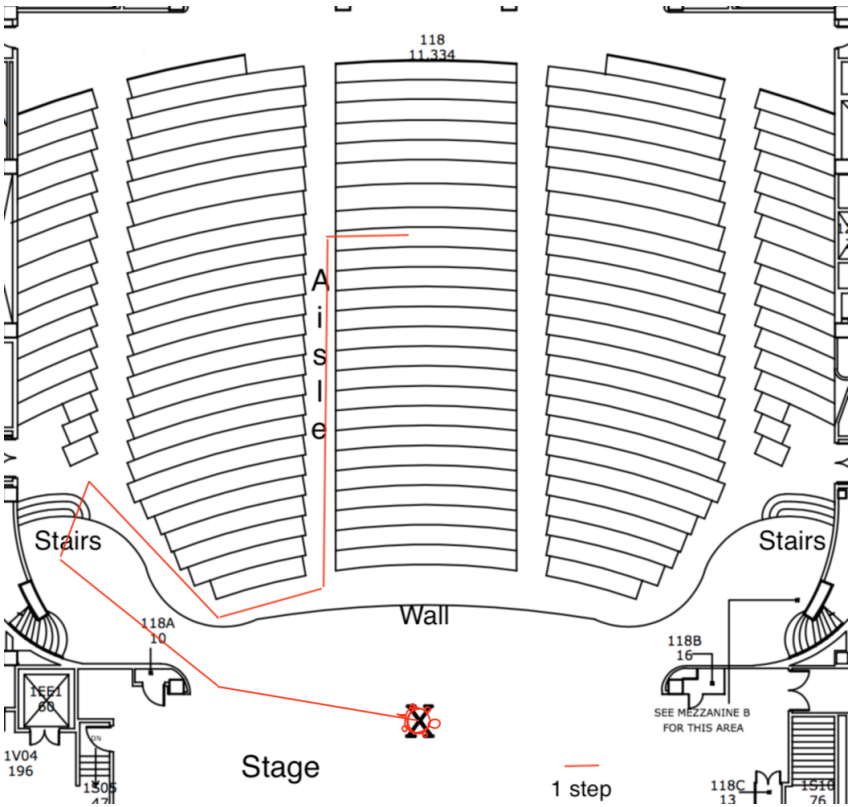
Your Turn!!!



One possible solution:

- Turn right 90 degrees.
- Walk forward 3 steps.
- Turn left 90 degrees.
- Walk forward 10 steps.
- Turn right 65 degrees.
- Walk forward 4 steps.
- Turn right 45 degrees.
- Walk forward 6 steps.
- Turn left 110 degrees.
- Walk forward 3 steps.
- Turn left 80 degrees.
- Walk forward 5 steps.
- Turn left 30 degrees.

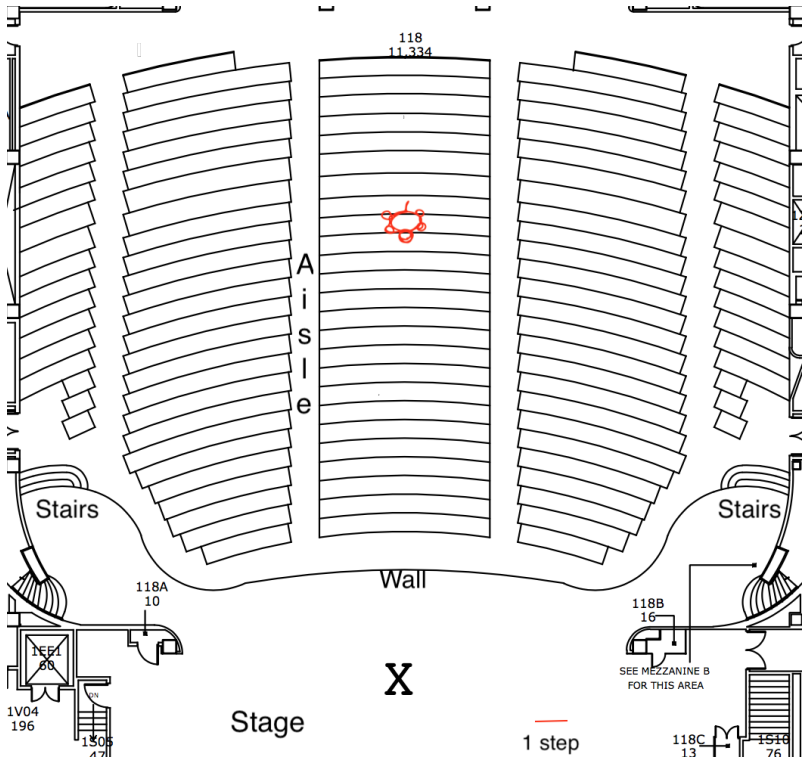
Your Turn!!!



One possible solution:

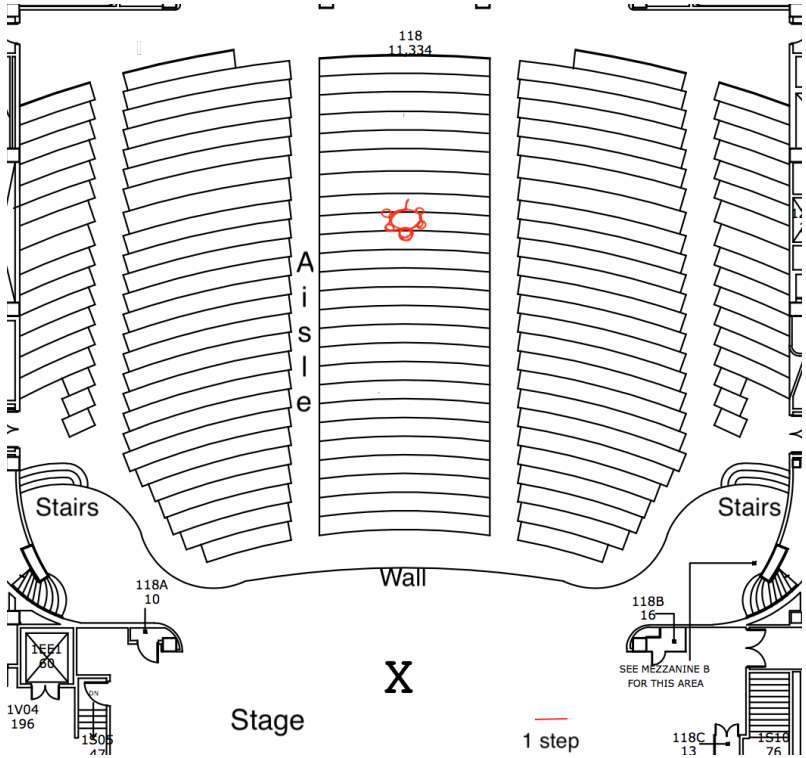
- Turn right 90 degrees.
- Walk forward 3 steps.
- Turn left 90 degrees.
- Walk forward 10 steps.
- Turn right 65 degrees.
- Walk forward 4 steps.
- Turn right 45 degrees.
- Walk forward 6 steps.
- Turn left 110 degrees.
- Walk forward 3 steps.
- Turn left 80 degrees.
- Walk forward 5 steps.
- Turn left 30 degrees.
- Walk forward 6 steps. Reached X!!!

Your Turn!!!



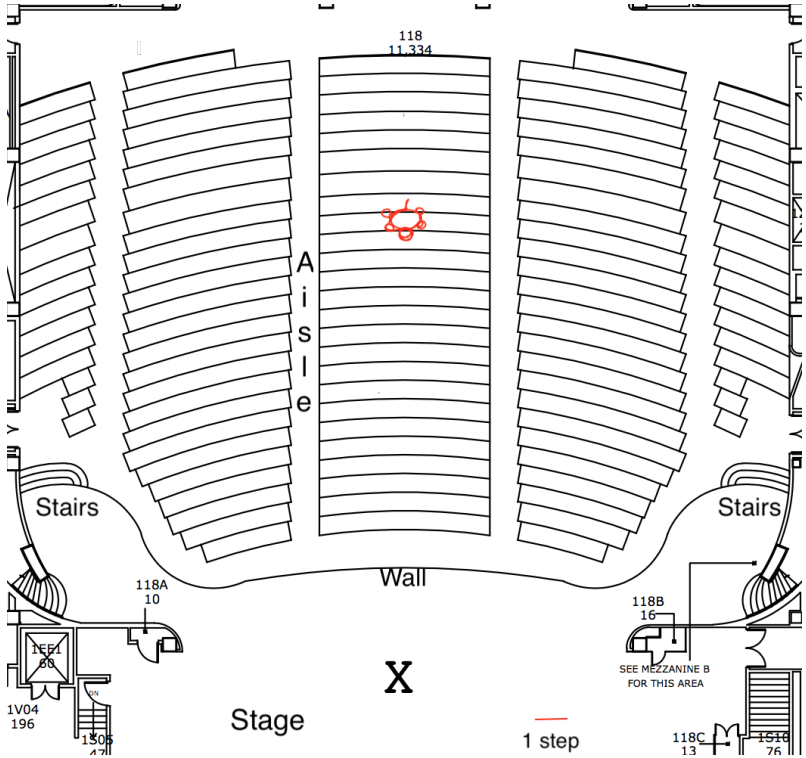
- For fun, post your algorithm on the "Turtle on Stage" forum in the Discussion Board on Blackboard

Your Turn!!!



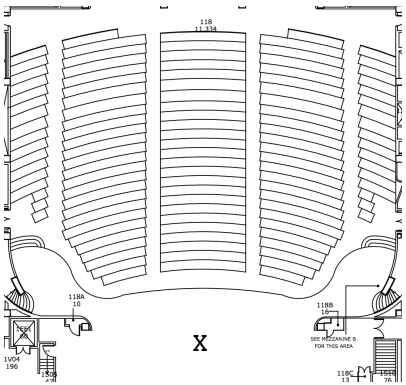
- For fun, post your algorithm on the "Turtle on Stage" forum in the Discussion Board on Blackboard
- "Test and Debug" other students' posted solutions and reply to their posts if you find a bug!

Your Turn!!!



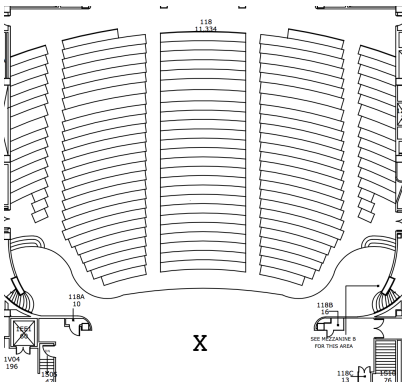
- For fun, post your algorithm on the "Turtle on Stage" forum in the Discussion Board on Blackboard
- "Test and Debug" other students' posted solutions and reply to their posts if you find a bug!
- Degrees the turtle turns are approximate, any good approximation is considered correct.

Recap



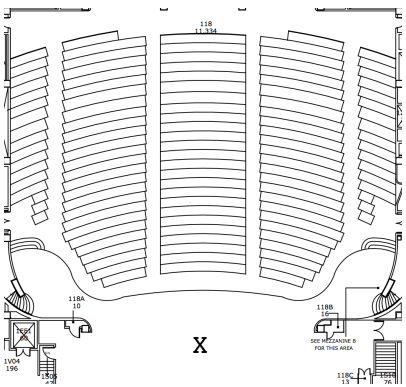
- Writing precise algorithms is difficult.

Recap



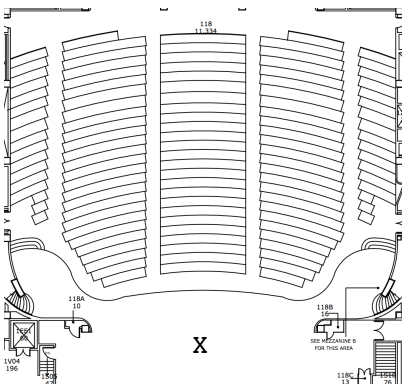
- Writing precise algorithms is difficult.
- In Python, we introduced:

Recap



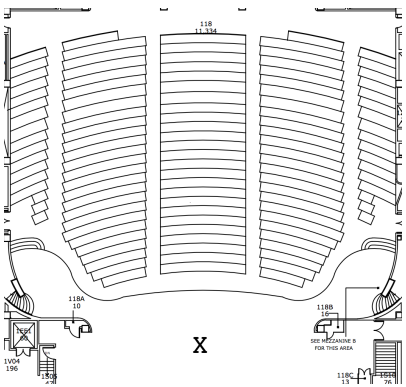
- Writing precise algorithms is difficult.
- In Python, we introduced:
  - strings, or sequences of characters,

Recap



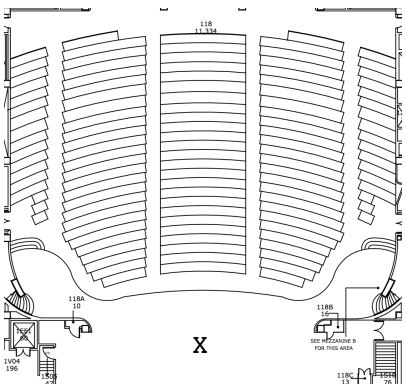
- Writing precise algorithms is difficult.
- In Python, we introduced:
  - strings, or sequences of characters,
  - `print()` statements,

Recap



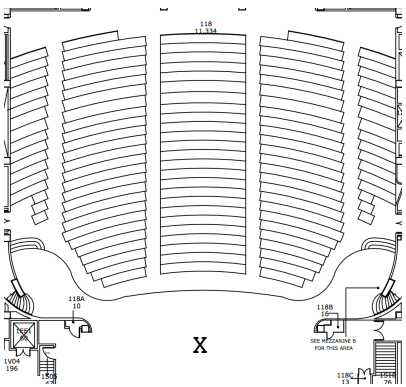
- Writing precise algorithms is difficult.
- In Python, we introduced:
  - strings, or sequences of characters,
  - `print()` statements,
  - `for`-loops with `range()` statements, &

Recap



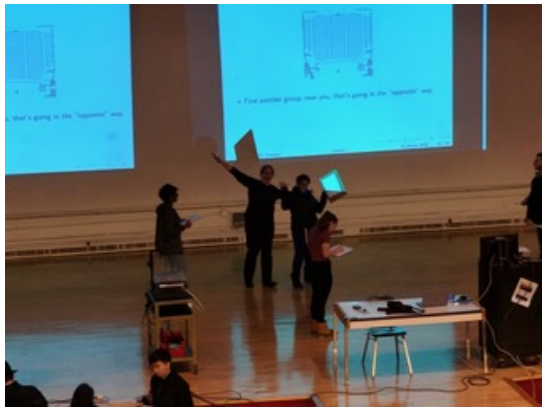
- Writing precise algorithms is difficult.
- In Python, we introduced:
  - strings, or sequences of characters,
  - `print()` statements,
  - `for`-loops with `range()` statements, &
  - variables containing turtles.

Recap



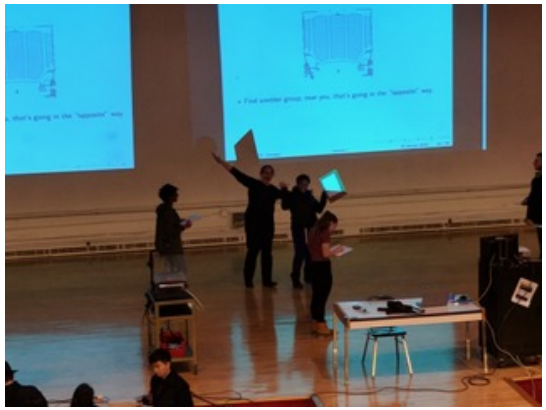
- Writing precise algorithms is difficult.
- In Python, we introduced:
  - strings, or sequences of characters,
  - `print()` statements,
  - `for`-loops with `range()` statements, &
  - variables containing turtles.

Practice Quiz & Final Questions



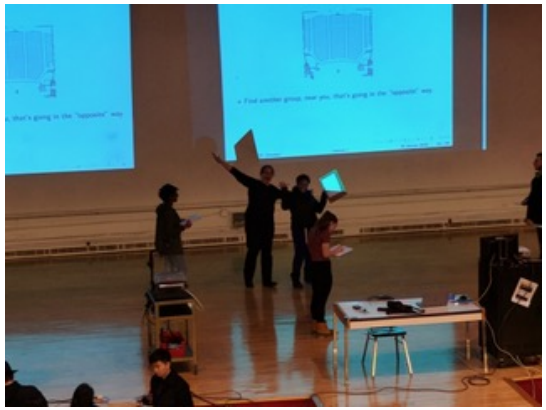
- Since you must pass the final exam to pass the course, we end every lecture with final exam review.

Practice Quiz & Final Questions



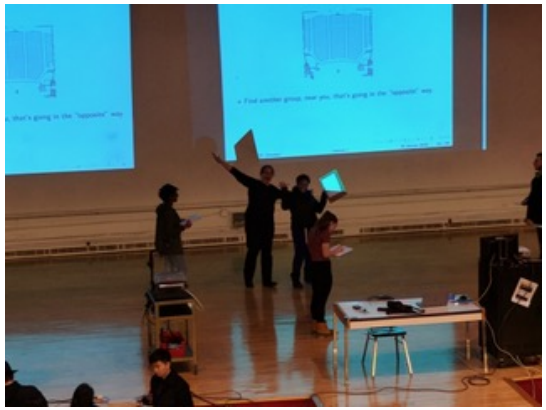
- Since you must pass the final exam to pass the course, we end every lecture with final exam review.
- Pull out something to write on (not to be turned in).

Practice Quiz & Final Questions



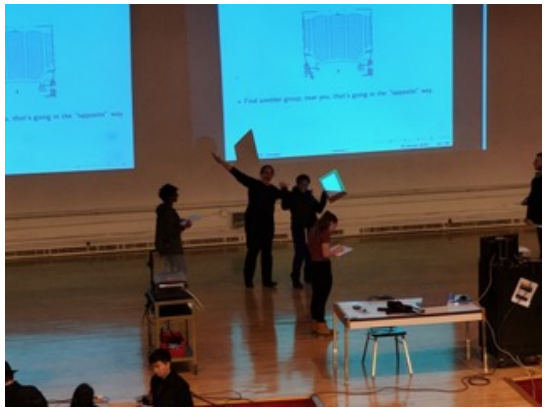
- Since you must pass the final exam to pass the course, we end every lecture with final exam review.
- Pull out something to write on (not to be turned in).
- Lightning rounds:

Practice Quiz & Final Questions



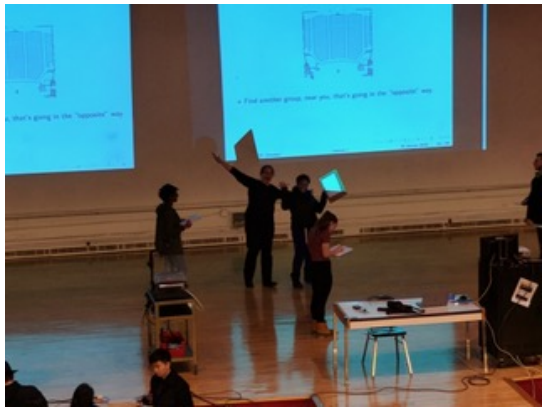
- Since you must pass the final exam to pass the course, we end every lecture with final exam review.
- Pull out something to write on (not to be turned in).
- Lightning rounds:
  - write as much you can for 60 seconds;

Practice Quiz & Final Questions



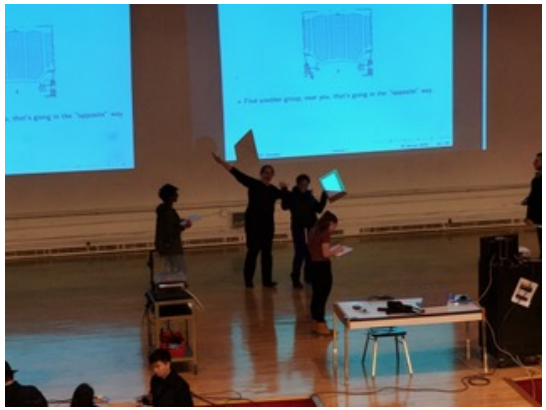
- Since you must pass the final exam to pass the course, we end every lecture with final exam review.
- Pull out something to write on (not to be turned in).
- Lightning rounds:
  - write as much you can for 60 seconds;
  - followed by answer; and

Practice Quiz & Final Questions



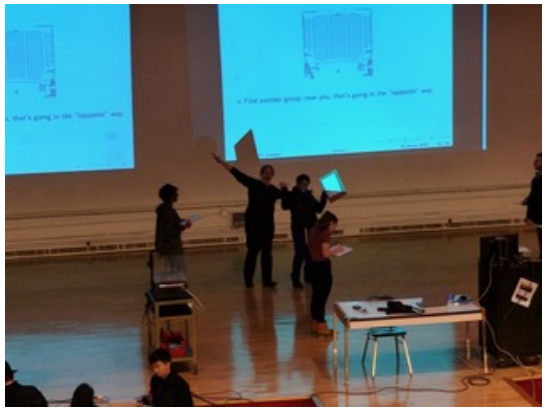
- Since you must pass the final exam to pass the course, we end every lecture with final exam review.
- Pull out something to write on (not to be turned in).
- Lightning rounds:
  - write as much you can for 60 seconds;
  - followed by answer; and
  - repeat.

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.

See you next week!

Before next lecture, don't forget to:

- Work on this week's Online Lab
- Take the Lab Quiz on Gradescope by 6pm on Wednesday
- Submit this week's 5 programming assignments