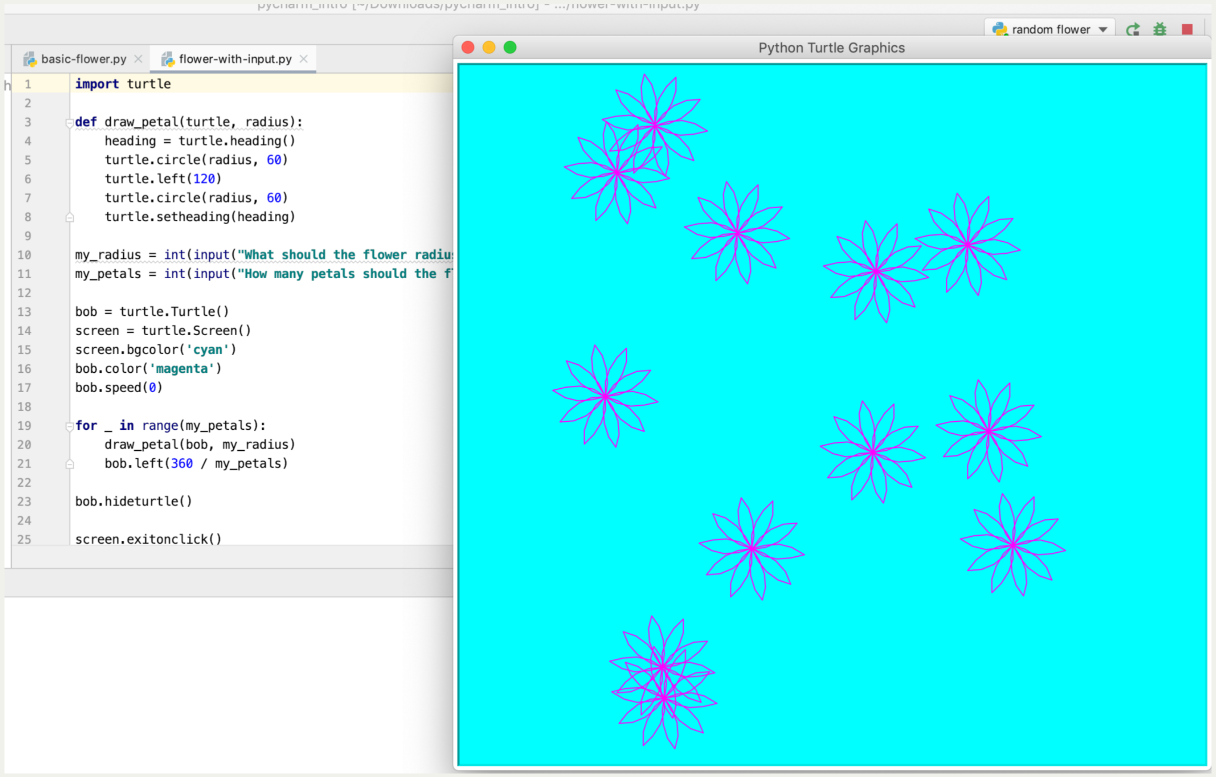
Lecture 1: Introduction

CS5001 / CS5003:
Intensive Foundations
of Computer Science



```
import turtle

def draw_petal(turtle, radius):
    heading = turtle.heading()
    turtle.circle(radius, 60)
    turtle.left(120)
    turtle.circle(radius, 60)
    turtle.setheading(heading)

my_radius = int(input("What should the flower radiu
my_petals = int(input("How many petals should the f

bob = turtle.Turtle()
screen = turtle.Screen()
screen.bgcolor('cyan')
bob.color('magenta')
bob.speed(0)

for _ in range(my_petals):
    draw_petal(bob, my_radius)
    bob.left(360 / my_petals)

bob.hideturtle()

screen.exitonclick()
```

PDF of this presentation

Lecture 1: Introduction

Today:

- About CS5001
- Course Staff
- What is computer science?
- What is programming?
- Why Python?
- Our first programs!
- Setting up the tools on your computer
    - The PyCharm IDE
    - The REPL
- Introduction to Programming in Python
    - Variables
    - Assignment
    - Arithmetic Operations
- Some simple (but interesting!) programs

Lecture 1: About CS5001

- This is a programming course where we will learn the Python programming language but more importantly, we will learn how to *solve problems with Python*
- We will learn Python and problem solving through a mix of lectures (me talking and demonstrating, and you doing, as well, and labs (you doing while Mark facillitates). We will also have weekly assignments, which will be a mix of short and longer programming assignments.
    - You will learn the basics and some nuances from me and Mark, but you will learn by *doing*, and this means thata you must put in the time to do the programming assignments.
- We will have two exams (a midterm and a final), where you will need to demonstrate that you understand Python programs (by reading them, and explaining the output, or fixing bugs, etc.), and you will need to demonstrate that you can write Python code.
- When you finish this course, you will know how to program (in Python -- but, learning other languages will not be too hard), and you will know how to solve problems using Python.

Lecture 1: About CS5001: Logistics

- Course website:
  - https://course.ccs.neu.edu/cs5001f19-sf/
- Piazza: (question / answer website):
  - http://piazza.com/northeastern/fall2019/cs5001sf
- Assignments:
  - One per week, generally due Tuesday before class
- Lecture:
  - Tuesdays, 7pm-10pm
- Lab:
  - Thursdays, 6pm-9pm
- Office hours:
  - There will be both online and in person office hours, depending on the person. See the course website for a calendar.
- Exams:
  - One in-class midterm (Tue. October 22nd)
  - A final exam (Tue. December

Lecture 1: Course Staff

- Chris Gregg, email: cgregg@northeastern.edu
  - BS Electrical Engineering, Johns Hopkins University
  - MS Education, Harvard University
  - Ph.D Computer Engineering, University of Virginia
  - U.S. Navy Cryptologist (7 years active duty, 16 years reserves)
  - High school physics teacher (7 years), Boston / Santa Cruz
  - Tufts University Lecturer, computer science department (2 years)
  - Stanford University Lecturere, computer science department (current)
  - Facebook software engineer (mostly summer)

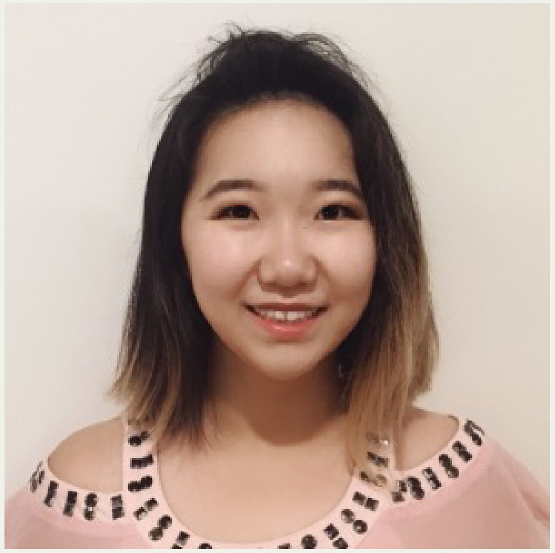
Lecture 1: Course Staff: CS5003 Lab Instructor

- Mark Miller
  - Ph.D., MIT EECS 1979 specializing in AI & Edu
    - Contributed to Logo, 1st programming language designed specifically for children
    - Worked with Papert, Minsky, Goldstein, Winston, Sussman, Abelson, Hewitt
    - Also worked at BBN (think tank in Cambridge where Logo was invented), with John Seely Brown, Jaime Carbonell, Allan Collins
  - Worked at Texas Instruments and Apple
    - Launched TI's AI corporate research
    - Lab Director, Learning and Tools at Apple
  - Founded Computer*Thought, VC-backed AI startup for programming language instruction
  - Founded Learningtech.org, a 501(c)(3)
    - Helps schools with EdTech & CS Edu

Lecture 1: Course Staff: TAs

- Yiya He

- Joyce Liu



Lecture 1: What is Computer Science?

- Computer science is about solving problems, primarily with the use of computers.
- Computer science is *not* programming, although programming is one of the tools computer scientists use to solve problems
- Examples of problems that computer science can solve:
  - What is the best route to take when traveling from San Francisco to Boston? (or, navigate humans to Mars)
  - Given a list of names, sort them alphabetically, as fast as possible
  - Determine which advertisement to insert into a user's Internet browsing experience
  - Solve a Sudoku puzzle
  - Analyze radio telescope data returned by a space telescope
  - Pilot a self-driving car (or rocket)
  - Provide secure encryption for online purchases
  - Break encryption from *the enemy*
  - Manage an online storefront, including the database that knows which items are available
- Computer science *uses* computers to solve problems, but computers themselves are not the focus of computer science.

Lecture 1: What is Programming?

- Programming is telling a dumb computer, in simple terms, how to accomplish a computational task.
  - Let's play a game...the peanut butter and jelly sandwich game
- Programming involves describing what you want a computer to do in an exact way, with no ambiguity.
- We will spend much of this class learning how to write unambiguous programs, and it can be tricky!
- Programming can be frustrating, but it can also be extremely rewarding, when you finally make your programs work the way you want them to work.

Lecture 1: What is Programming?

- Much of the time you spend programming will be fixing *bugs*, which are mistakes that you've created in your programs. There are three primary types of bugs:
    - Syntax errors
        - *Syntax* is the structure of your program, or the rules that you must follow to write a correct program. A *syntax error* happens when you don't follow those rules. Your program won't run if there are syntax errors anywhere in the program.
    - Runtime errors
        - The *runtime* is when your program is running. Syntactically correct programs can have *runtime errors* if they try to do something that is unexpected, like dividing by 0, or trying to print something to the keyboard.
    - Semantic errors
        - Possibly the most difficult bug to find is a *semantic error*, where your program will run, but it won't do what you wanted it to do. After you get used to Python syntax, you will spend most of your time debugging semantic errors
- We will learn a lot about programming in this class, and we will practice doing a lot of programming. You will likely find it both frustrating and exhilarating!

Lecture 1: What is Python (and why are we using Python)?

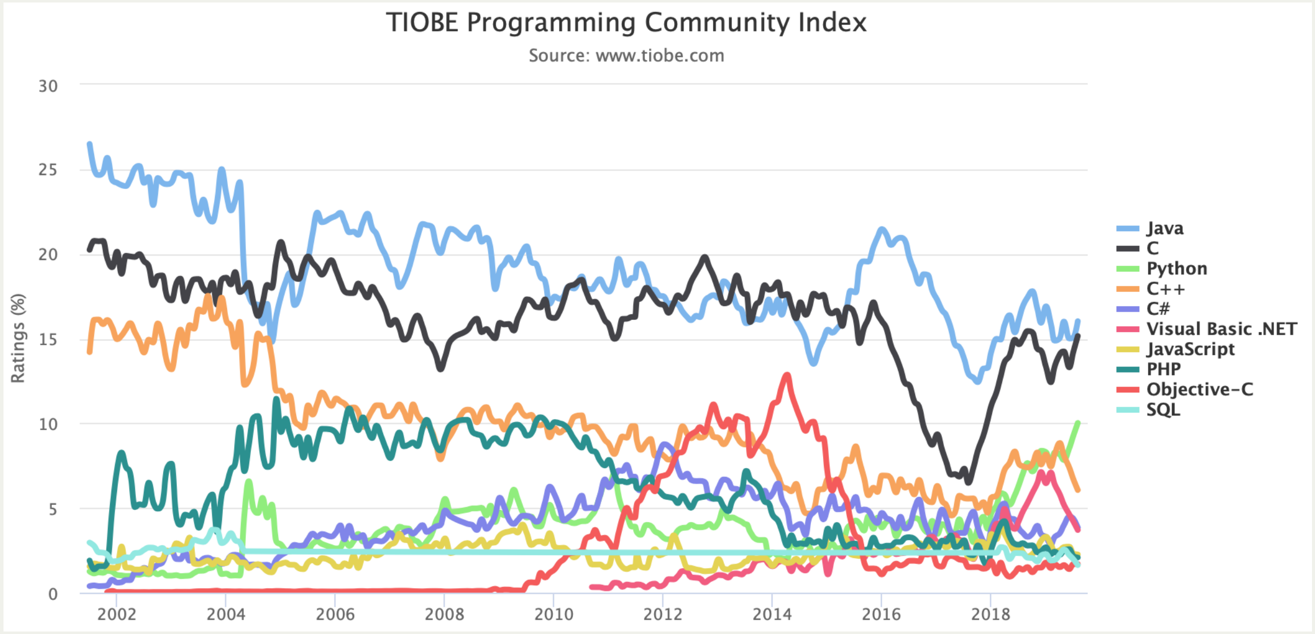
- Python is a *high-level* programming language, designed to be easy to learn, and useful for both small and large programs.
  - *low-level* languages, also known as *machine languages* are more difficult to write in, and are closer to what the underlying computer actually understands. A high-level language, like Python, is easier to write in, and the same program works on different types of computers. This is not true for low-level languages
- Python programs need to be formatted carefully -- the PyCharm environment you will most often use will help format your programs for you, but you need to understand the rules (which we will learn, of course)
- Here is an example of a python program (you do not need to understand it yet!):

```python
'''This program asks for a person's name and age, and then prints their age for every year until 2030'''

CURRENT_YEAR = 2019
END_YEAR = 2030

name = input("What is your name? ")
age = int(input("What is your age? "))

print("\nHello, {}!\n".format(name))
print("Year Age")

for year in range(CURRENT_YEAR, END_YEAR + 1):
    print("{} {}".format(year, age + year - CURRENT_YEAR))

print("Goodbye!")
```

- Let's test it! https://repl.it/languages/python3

Lecture 1: Why Python?

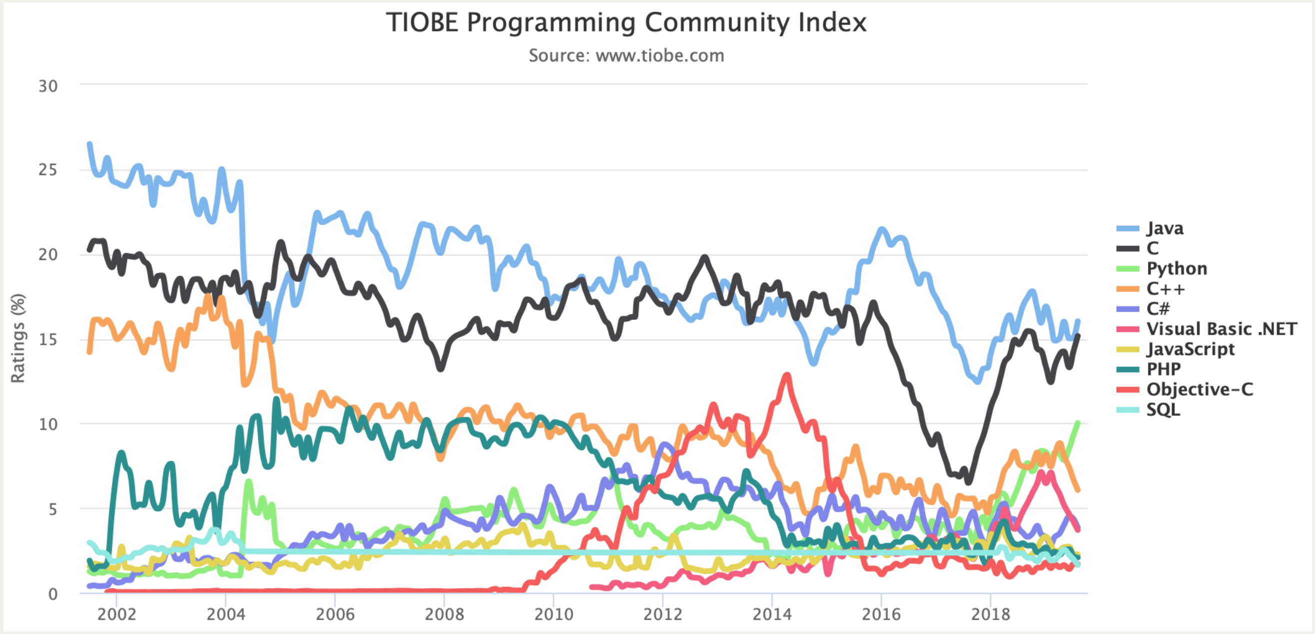
- This is a list of the top programming languages over time. The light green line is the Python language.
- Notice that it has been becoming more popular over the years. It was actually invented in 1991, and started gaining popularity in the early 2000s
- It isn't quite at the top, but it is a great language to learn first, and it is a language that will likely continue to become more important.
- Some other languages near the top of the list:
    - Java: This is an important language because it is another high-level language that is geared towards running on many different types of computers easily
    - C and C++: These two languages are relatively old (1972 and 1985), but most code written for embedded systems is written in C, and so are the backbone of most operating systems (Linux, Windows, MacOS). You will learn these languages in follow-on courses.



- Javascript (1995): This is the *language of the web* (and has nothing to do with Java). All websites have some Javascript running them.
- Does anyone know the most widely used language in programming?

Lecture 1: Why Python?

- This is a list of the top programming languages over time. The light green line is the Python language.
- Notice that it has been becoming more popular over the years. It was actually invented in 1991, and started gaining popularity in the early 2000s
- It isn't quite at the top, but it is a great language to learn first, and it is a language that will likely continue to become more important.
- Some other languages near the top of the list:
  - Java: This is an important language because it is another high-level language that is geared towards running on many different types of computers easily
  - C and C++: These two languages are relatively old (1972 and 1985), but most code written for embedded systems is written in C, and so are the backbone of most operating systems (Linux, Windows, MacOS). You will learn these languages in follow-on courses.



- Javascript (1995): This is the *language of the web* (and has nothing to do with Java). All websites have some Javascript running them.
- Does anyone know the most widely used language in programming?

Profanity!

Lecture 1: Our First Programs!

- Here is our first simple Python program (and the first program you'll write in most languages):

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

- This is such a simple program that you can probably guess what it is going to do. We can test it. This program uses the **`print`** *function*, which means that whatever we put inside the parentheses gets printed to the screen.

- So, let's move on to our second program, which is only a bit more interesting:

```
# Ask for a user's name, and print it
name = input("What is your name? ")
print(name)
```

- Okay, that was a bit more intesting, in that we had input *and* output.
- In this case, we used a *variable*, which is simply a name that refers to a *value*. In this case, the value was a *string*, which is a series of characters, in this case, the user's name.

Lecture 1: Our First Programs!

- What if we want to print "**`Hello`**," and then the name? How would we do that?

```
# Ask for a user's name, and print it
name = input("What is your name? ")
???
print(name)
```

- But, what if we want to print "**`Hello, Name!`**", all on one line?

  - Now we have to get into *formatting* with the **`print`** function.
  - Formatting with the **`print`** function looks like this:

```
print("Hello, {}!".format(name))
```

    - Wherever we want to embed something into our print string, we put two curly braces, **`{}`**.

  - On the next slide, we will see a more advanced program, with a more advanced format string.

Lecture 1: A more advanced format string

```
# Ask for a user's name and age, and print them
name = input("What is your name? ")
age = input("What is your age? ")
print("Hello, {}, you look great for being {} years old!".format(name, age))
```

- The above program has a new variable, called `age`, which we use to store the user's age.
- The format statement now has two sets of curly braces, and in the format parentheses, we put both variables, separated by a comma. The first variable gets substituted into the first set of curly braces, and the second variable gets substituted into the second set of curly braces.
- By the way: everything needs to be syntactically correct. We must have opening and closing parentheses where needed, and we must have the quotation marks in the right places. If we leave anything out, we will get a syntax error -- let's test that!
- We can create a semantic error, too -- what if we switched the variable order?

```
# Ask for a user's name and age, and print them
name = input("What is your name? ")
age = input("What is your age? ")
print("Hello, {}, you look great for being {} years old!".format(age, name))
```

Lecture 1: A more advanced format string

```
# Ask for a user's name and age, and print them
name = input("What is your name? ")
age = input("What is your age? ")
print("Hello, {}, you look great for being {} years old!".format(name, age))
```

- The above program has a new variable, called **`age`**, which we use to store the user's age.
- The format statement now has two sets of curly braces, and in the format parentheses, we put both variables, separated by a comma. The first variable gets substituted into the first set of curly braces, and the second variable gets substituted into the second set of curly braces.
- By the way: everything needs to be syntactically correct. We must have opening and closing parentheses where needed, and we must have the quotation marks in the right places. If we leave anything out, we will get a syntax error -- let's test that!
- We can create a semantic error, too -- what if we switched the variable order?

```
# Ask for a user's name and age, and print them
name = input("What is your name? ")
age = input("What is your age? ")
print("Hello, {}, you look great for being {} years old!".format(age, name))
```

Lab 1: Getting your tools set up, practicing some code, and submitting your code

- Now that we have seen a *little* bit of programming, it is time for you to try some programs with a partner.
- Go to the course website (https://course.ccs.neu.edu/cs5001f19-sf), click on the Calendar link, and then click on the link for Lab 1.
  - We will first set up our tools (Python and then the PyCharm *Integrated Development Environment*)
  - Next, you will work with a partner to write some simple programs
  - Finally, you will submit your programs to https://handins.ccs.neu.edu/courses
- Your first assignment, which is due next Tuesday before class (7pm, not 6pm), will be turned in the same way.