Welcome to Python!

March 29, 2022

Parth Sarin (they/he)

Originally from: College Station, TX
Program: Math, Policy, Education, Religion
Hobbies: Baking, running, long walks, dance, picking fruits off trees on campus

Watching

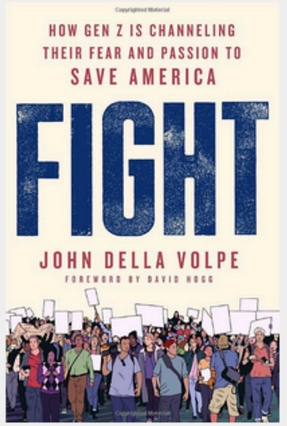
Abbott Elementary

Avatar: The Last Airbender

Listening To

Shine
(by Koffee)

Reading



Fight
By: John Della Volpe

Tara Jones (she/her)

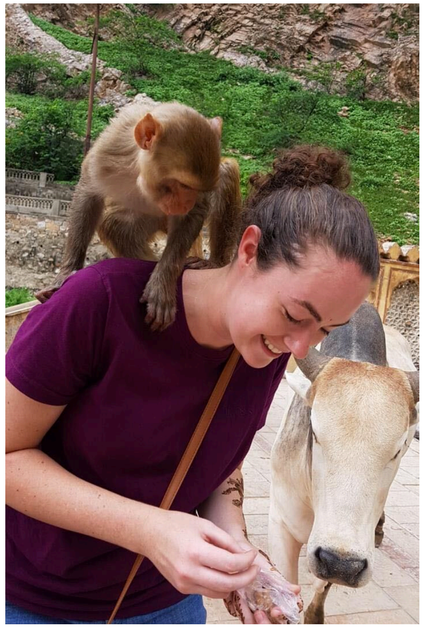
Originally from: Monterey, CA
Program: Computer Science B.S. and M.S.
Hobbies: Teaching!! Riding horses, Barry's Bootcamp, Cooking vodka pasta, Super Smash Bros

Watching

Listening To

Hanging out with

My cousin's cute python



Elizabeth Fitzgerald
(she/her)

Theo Culhane
(he/him)

Shounak Ray
(he/him)

Arpit Ranasaria
(he/him)

Sign up for sections on Canvas by next Monday

(when you have a group)

Case Study: Hello World

```java
// Java
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello world!");
    }
}
```

```
$ javac HelloWorld.java
$ java HelloWorld
Hello world!
```

Case Study: Hello World

```cpp
// C++
#include <iostream>
using namespace std;

int main() {
    cout << "Hello world!" << endl;
    return 0;
}
```

```
$ g++ helloWorld.cpp
$ ./a.out
Hello world!
```

Case Study: Hello World

```
print("Hello world!")
```

```
$ python helloworld.py
Hello world!
```

Case Study: Int Size

```cpp
// C++
#include <iostream>
using namespace std;

int main() {
    cout << sizeof(41) << endl;
    return 0;
}
```

```
$ g++ intSize.cpp
$ ./a.out
4
```

Case Study: Int Size

```
print((41).__sizeof__())
```

```
$ python intsize.py
28
```

>>> import this

The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.

>>> import this

In the face of ambiguity, refuse the temptation to guess.

There should be one-- and preferably only one --obvious way to do it.

Although that way may not be obvious at first unless you're Dutch.

Now is better than never.

Although never is often better than *right* now.

If the implementation is hard to explain, it's a bad idea.

If the implementation is easy to explain, it may be a good idea.

Namespaces are one honking great idea -- let's do more of those!

Learning Goals

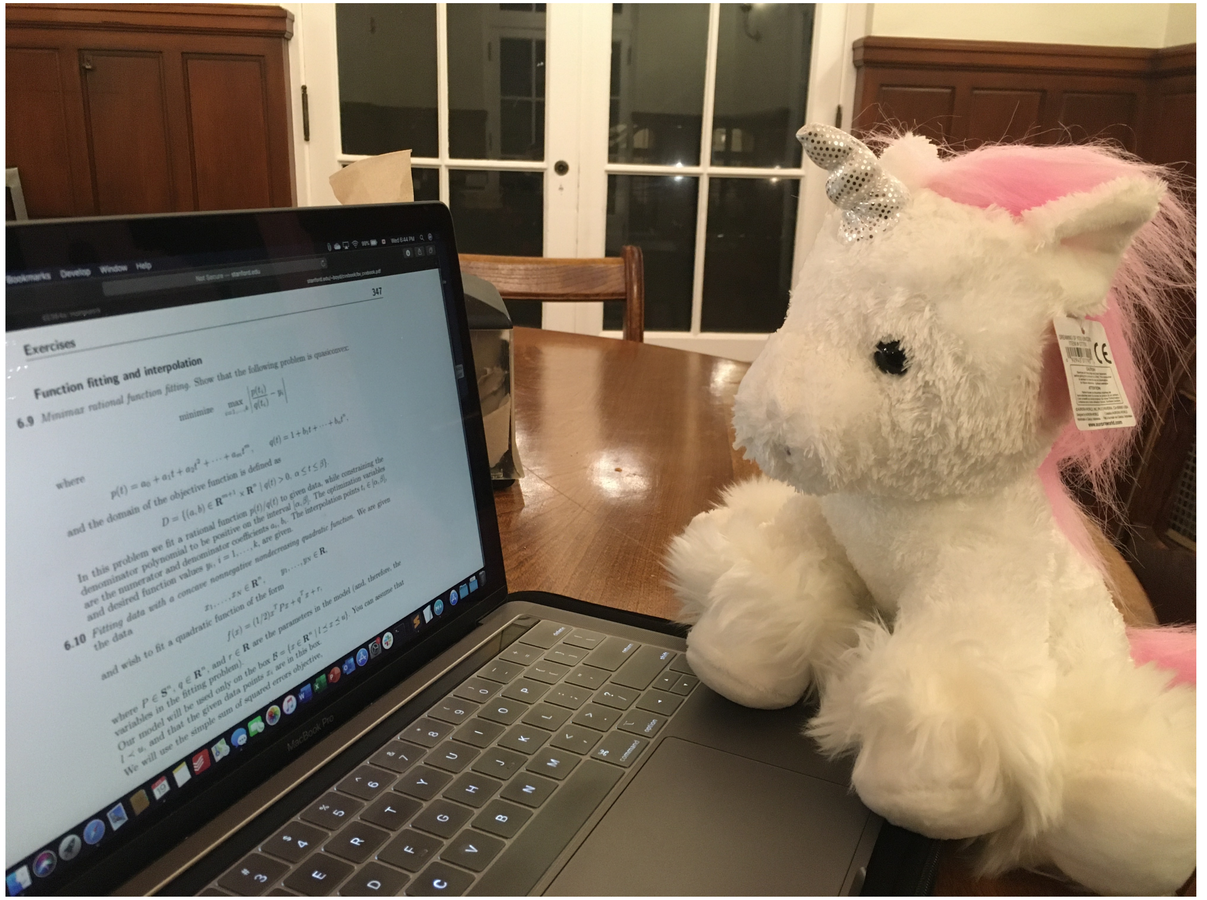
After CS 41, students will be able to...

1. Compare and contrast Python's language design with other languages they've seen before.
2. Determine whether or not Python is an appropriate language to write a program in.
3. Implement a significant project in Python.
4. Translate code written in other languages into Python, especially code that doesn't rely on third-party libraries.
5. Select the appropriate data structure for a program written in Python. Recognize when Python does not have the appropriate built-in data structure for a problem.
6. Design and implement custom Python objects (classes) for Python programs to augment Python's object functionalities.
7. Interact with external applications, like web APIs, using Python to send requests and process responses.
8. Apply standard libraries and third-party packages to domain-relevant problems.

Logistics

Ed	Discussion, Ed workspaces, link on the website
Canvas	Section sign-up, link on the website
Groups	Sign up link (also in the Assignments tab of the website)
Assignments	3 assignments + a final project A0 is individual, the rest are in groups
Grading	No grades on any assignments, feedback only Overall course grade is based on completion
Labs	Class on Tuesday, Lab on Thursday
Office Hours	Links on the website
Waitlist?	Come see us after class!

Unicornelius (any/all)



The Road Ahead

The Road Ahead

Week 1: Welcome & Basics

Week 2: Data Structures

Week 3: Object-Oriented Python

Week 4: Functions

Week 5: Python & the Web, APIs

The Road Ahead

Week 6: Python & the Web, Server-side

Week 7: Functional Programming

Week 8: Standard Libraries

Week 9: TBD

Week 10: Final Project Presentations

🐍 Basics

~~Python Basics~~

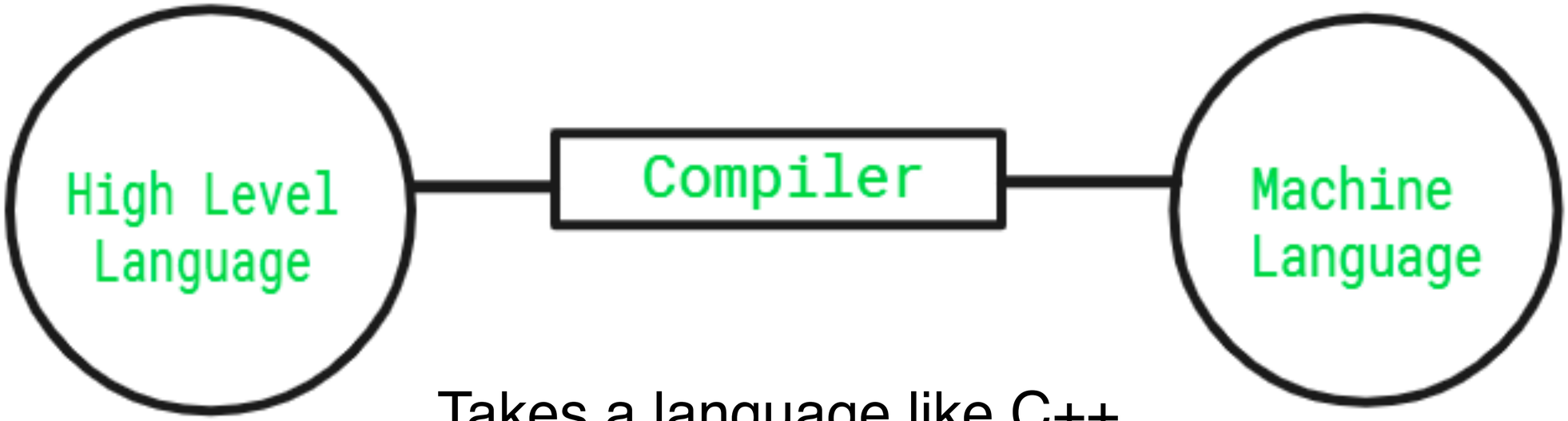
3/29/2022

Today’s Agenda

- Complied vs. Interpreted
- Variables & Types
- Numbers & Booleans
- Strings
- Lists
- Control Flow

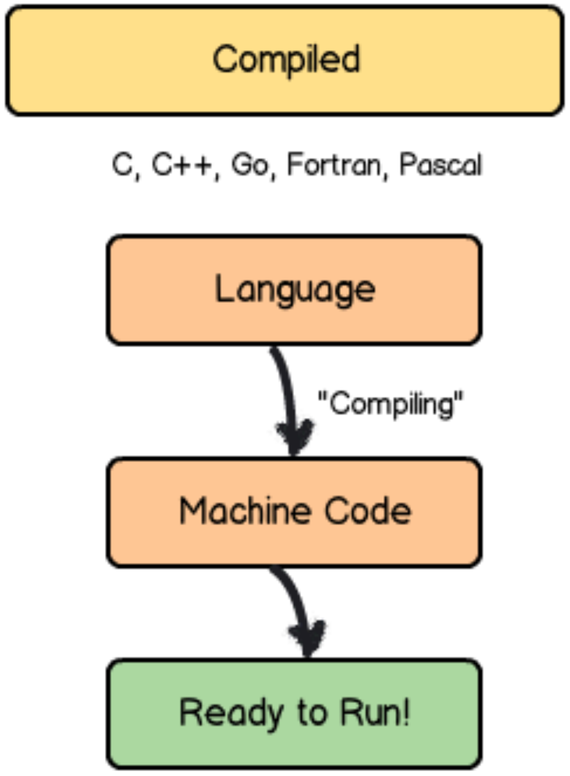
Compilers

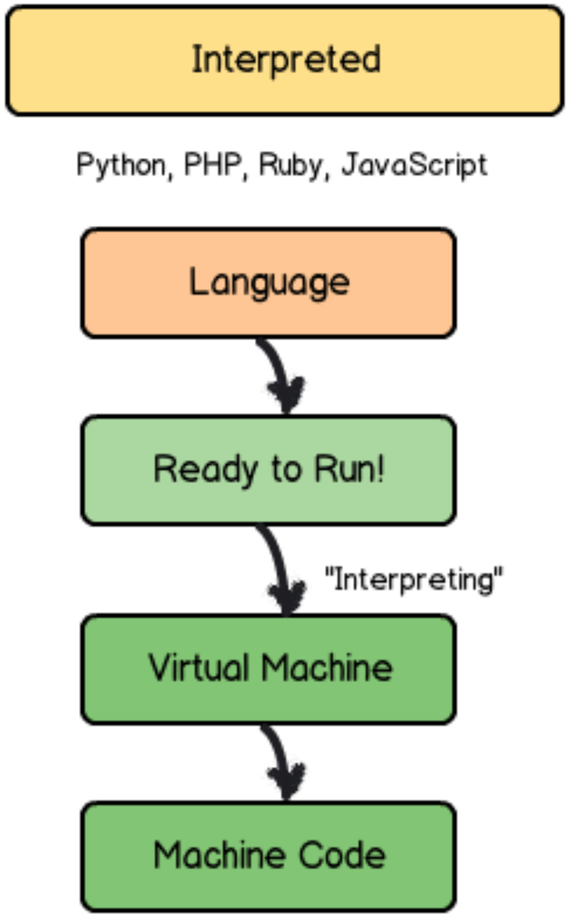
Compilers

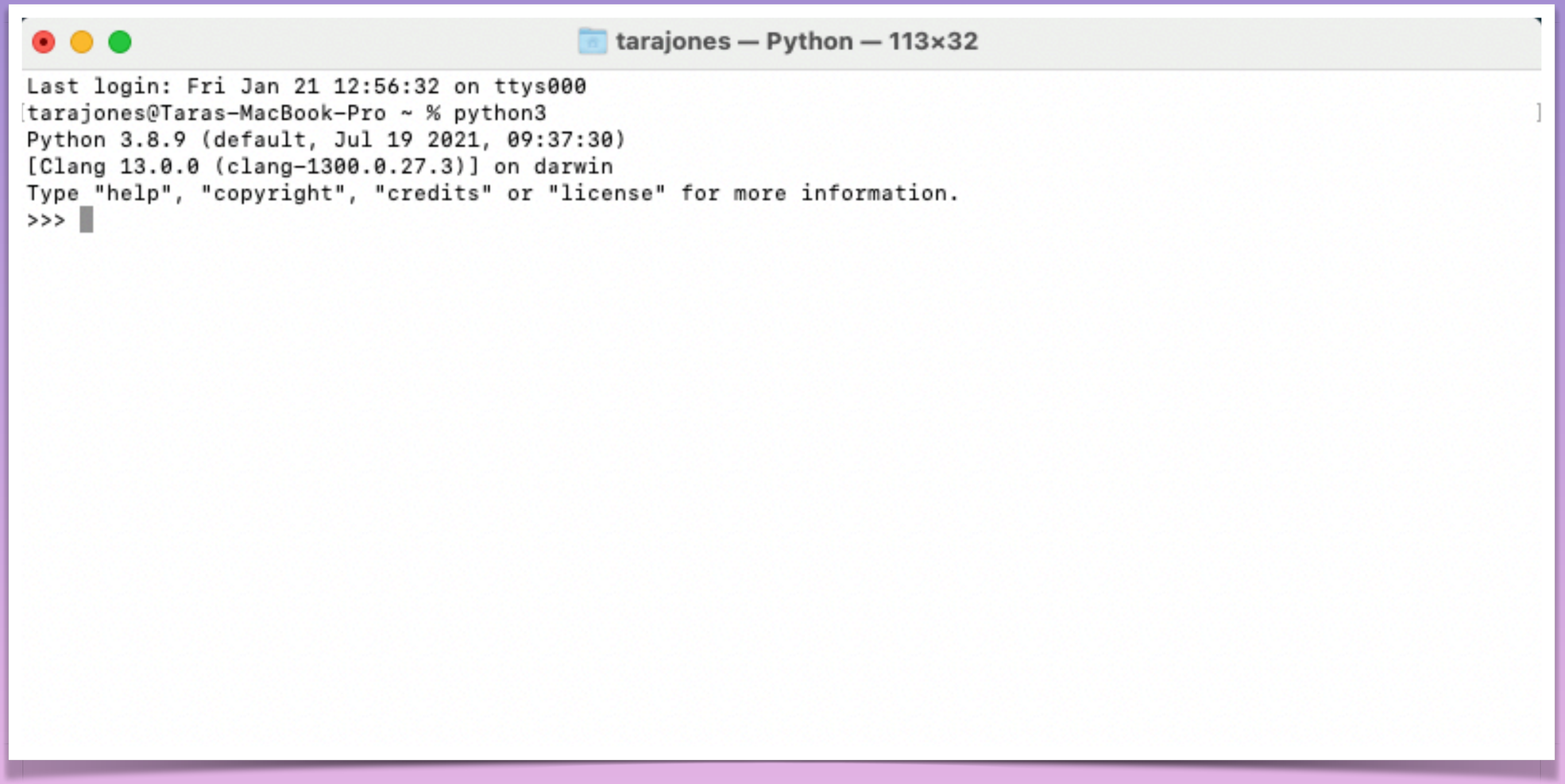


Takes a language like C++ and allows it to run on your computer’s CPU

Python is an Interpreted Language







The fact that python is interpreted gives us this great tool

Variables in Python

Something to get used to is no explicit distinction of types by the user

```
a = 5
name = "Tara Jones"
```

types are inferred by the interpreter

Now this does not mean that there are no types in Python, they are just hidden from the user and show up functionally

- Variable type can be checked by checking `type(var)`

Variable’s value and their types can change throughout the program

Numbers and Bools

Numbers

```
#int
a = 5

#float
b = 5.0
```

Numeric Operation	Operation Syntax	Assignment Syntax
Addition	`x + 5`	`x += 5`
Subtraction	`x - 5`	`x -= 5`
Multiplication	`x * 5`	`x *= 5`
Division	`x / 5`	`x /= 5`
Integer Division	`x // 5`	`x //= 5`
Modulo Operator	`x % 5`	`x %= 5`
Exponentiation	`x ** 5`	`x **= 5`

Booleans

```
cs_41 = True

is_it_friday_yet = False
```

Boolean Operation	Operation Syntax
Not	`not a`
And	`a and b`
Or	`a or b`
Equals (Not Equals)	`a == b (a != b)`
Greater (Or Equal)	`a > b (a >= b)`
Less (Or Equal)	`a < b (a <= b)`
Chained Expressions	`a > b > c`

Strings

```
name = "Tara Jones"
```

```
name = "Tara Jones"
       0123456789
```

- Can parse a character by writing name of string, hard brackets, and the index:

```
name[1]
```

- Can parse a substring by writing hard brackets, and the start of the substring and the exclusive end of the substring separated by a colon:

```
name[0:4]
```

```
name = "Tara Jones"
       0123456789
```

- Can do some fun stuff with negative parsing

```
name[-1]
```

- This also works with sub string parsing

```
name[-5:]
```

```
name = "Tara Jones"
       0123456789
```

IMPORTANT to note that strings in python are immutable

- This means that they do not support changing parts of the string once it has been created
- But, we can fully reset a variable to a different string and add strings together
- Len function is helpful here!

Lists

- Collection of Objects
- Very similar functionality to strings
- Allow for multiple types inside
- Can check if something is “in” list
- List parsing

```
my_list = []
my_list.append("Tara")
my_list.remove("Tara")
my_list.append(6)
```

Input and Casting

```
name = input("What is your name: ")
```

```
phone_num = int(input("What is your #?: ")
```

Questions?

Control Flow

if, else

```
num = int(input("Enter a number: "))
if num % 2 == 0:
    print("even")
else:
    print("odd")
```

if, elif, else

```
animal = input("Enter an animal: ")

if animal == "dog":
    print("I love dogs!!")
elif animal == "horse":
    print("Neigh!!")
else:
    print("Cool choice!!")
```

Truthiness and Falseness

Python objects have a quality of being “truthy” or “falsy”

- **Falsy:**
  - Variables with values of 0 or None, empty data structures
- **Truthy:**
  - Just the opposite! Variables with values other than 0 or None, data structures and strings that are not empty

Try Except

```
name = input("What's your name?: ")

try: #dangerous code
    ID = name + 2022
except TypeError: #give an exception for a type error
    print("Oops! You tried to add two different types")
```

- For trying dangerous code

Loops

```
for i in range(100):
    print(i)
```

```
my_list = [1,2,3,4]
for i in range(len(my_list)):
    print(my_list[i])
```

```
for elem in my_list:
    print(elem)
```

Continue

```
#make every number odd
my_list = [1,2,3,4,5]

for elem in my_list:
    if elem % 2 == 0:
        elem+=1
    else:
        continue
```

While Loops

```
while true:
    number = int(input("Give a number: "))
    if number > 100:
        break
```

Functions

```
def my_func(a,b):
    return a + b
```

```
def my_func:
    print("I can take in no params and not explicitly return")
```

Function Facts

- Parameters do not have set return type
- Nested functions can exist
- Return statement is option and will naturally return an object `None`

Function Demo

is_prime

- Number greater than 1 that is only divisible by 1 and itself!

Thank you!