CS Unit Kick-off

HELLO!

- My name's Mari
- Software Engineer for 6 years
- Currently a Sr. iOS Engineer in FB Dating
- Previously at Apple, Coursera
- I love EDM, boxing & photography

AGENDA

- CS Unit Expectations
- Python Basics
- *Attendance code will be posted in the Zoom chat during break*
- *Ask questions on the Slack thread*
- *I don't expect you to be coding with me during lecture*
- *I will post slides in advance and code snippets afterwards*

CS Unit Expectations

CS UNIT EXPECTATIONS

- **Goal**: Teach you problem solving skills, algorithms and data structures needed for technical interviews
- We’ll be using *Python* to learn this material

WHY PYTHON?

- Python is a very common language used in the industry
    - Data analysis, server/backend frameworks (e.g. Django), scripts, etc.
- It’s also a very intuitive, readable and concise language
- Do you need to use Python for technical interviews? No!

ASSESSMENTS

- Module Project/Exercises
    - Highly recommended to self-assess your understanding
- Sprint Challenge
    - 4th Sprint Challenge: >= 60% score Required to pass the unit
- General Coding Assessment (GCA)
    - Required for graduation, no minimum score required
- All these tests are done via *CodeSignal*
- Please use Python for these assessments
- **Make sure to register using your lambda school email!**

CODESIGNAL

SUPPORT

- Our awesome SL/TLs
- Our TA, Steve Lanier will be hosting study hall lectures
- I will be hosting AMAs throughout the unit

TIPS FOR SUCCESS

- CS is not easy. **Do not fall behind**.
- Utilize our support staff. We want you to succeed!
- Look at the warmup before lecture
  - Lectures are **not** a substitute for material in Canvas
- Do additional problems outside of what we give you
  - Leetcode is arguably the best place you can practice
- Still don’t understand? Look at other resources (Youtube, etc.)

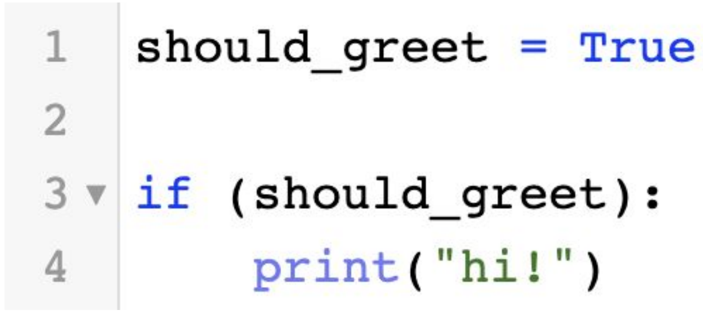
Python Basics

PYTHON BASICS

- Python is arguably one of the easiest languages to learn!
- The following slides give a brief overview of the common things to know

BOOLEANS

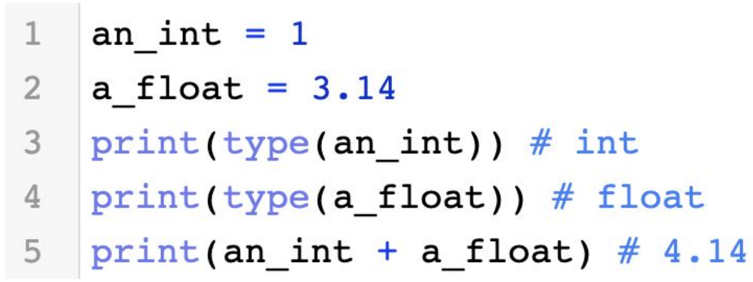
- Booleans can only have two values: True or False
- Usually used to control the flow of the program
- Commonly used operations:
    - == for equality
    - **not** keyword for negation
    - bool() to cast other types to a Boolean
- Other data types can evaluate to booleans:
    - e.g. “abc”, 123, [“a”, “b”] all evaluate to True
    - None, 0, “”, [], () all evaluate to False
- Booleans are immutable, so they behave similar to pass-by-value when passed inside a function



```python
should_greet = True

if (should_greet):
    print("hi!")
```

NUMBERS

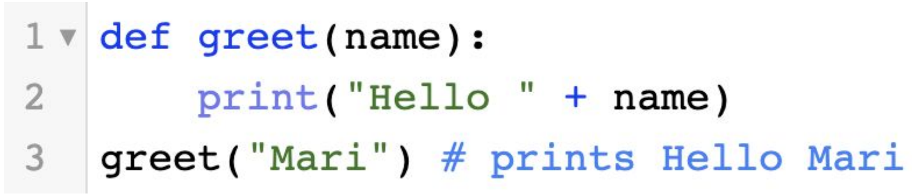
- The two most common number data types: **int**, **float**
- **int** is a positive/negative whole number (1, 2, 3, etc.)
- **float** is a positive/negative number containing a decimal
  - 1.0, 2.1, 3.14, etc.
- You can use math operations on these data types
- You can cast other data types to int or float
- Numbers are immutable, so if you pass them in a function, they behave similar to pass-by-value



```python
an_int = 1
a_float = 3.14
print(type(an_int)) # int
print(type(a_float)) # float
print(an_int + a_float) # 4.14
```

FUNCTIONS

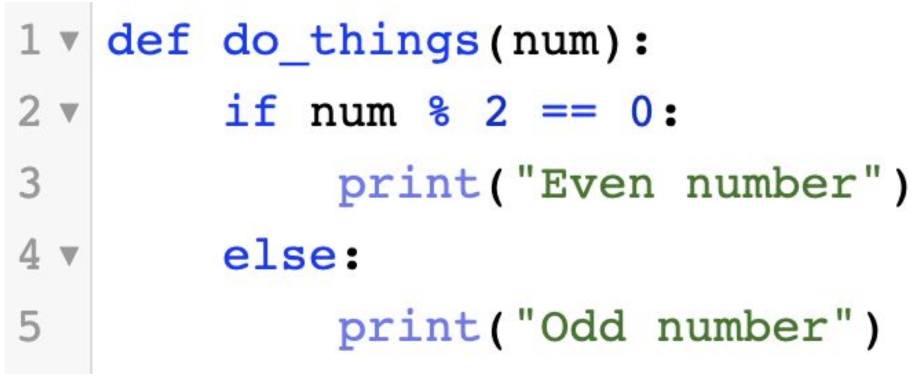
- Defined using **def** keyword
- The behavior of arguments passed in a function depends on the data type
- Python cares about whitespaces to signify end-of-line and code blocks
- Fun-fact:
    - Functions can contain inner functions



```
def greet(name):
    print("Hello " + name)
greet("Mari") # prints Hello Mari
```

CONTROL FLOW

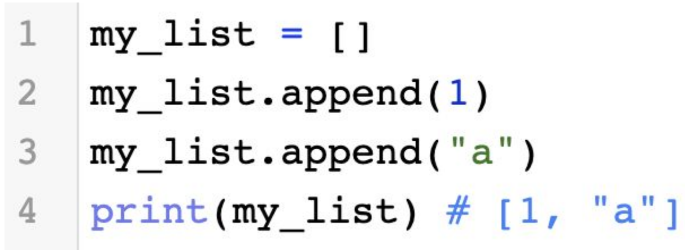
- You can use the following to control the flow of execution:
  - if/elif/else blocks
  - and/or operators
- Note: Python requires code to be indented/have whitespace in order to be executed in the block



```python
def do_things(num):
    if num % 2 == 0:
        print("Even number")
    else:
        print("Odd number")
```

LISTS

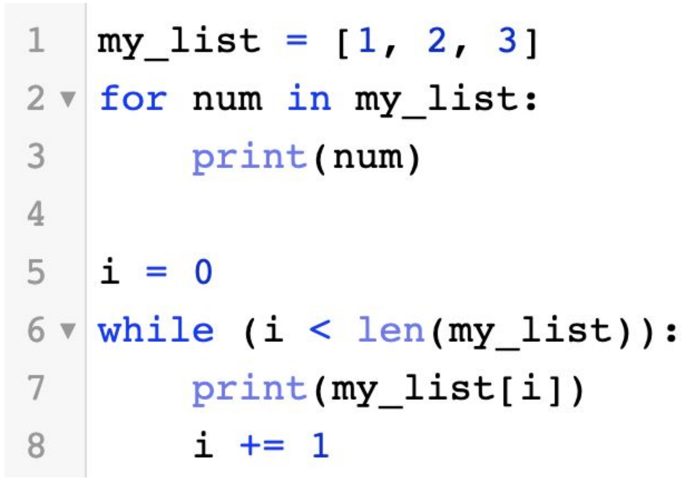
- Implemented using arrays in Python
- Used to store sequential data
- Loosely-typed, can store heterogeneous data types
- Lists are mutable, so passing them in a function behaves similar to pass-by-reference
- Fast access via indexing and appending/removing at the end of the list
- You can get sublists using slicing
- See common operations here



```python
my_list = []
my_list.append(1)
my_list.append("a")
print(my_list) # [1, "a"]
```

ITERATION

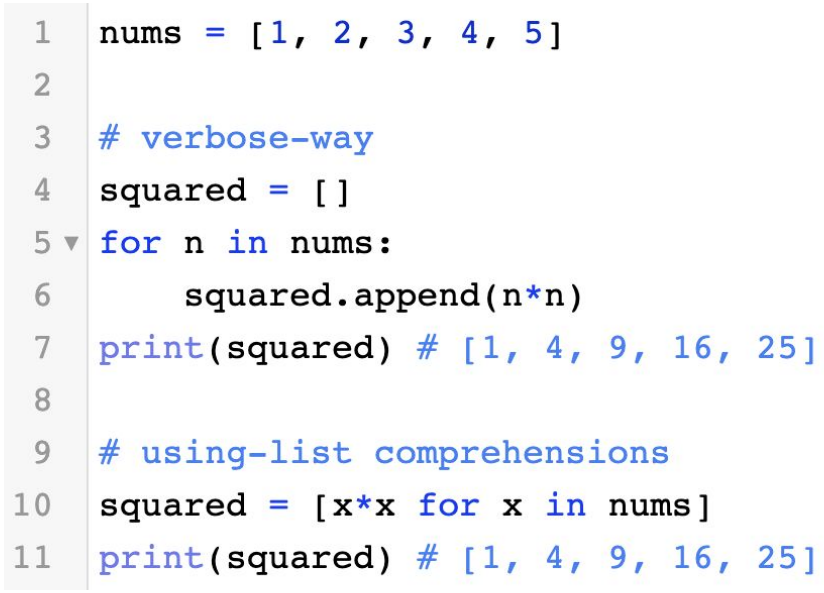
- Two ways to iterate through a collection:
  - **for loop -** iterate over a specified range
  - **while loop -** iterate while a certain condition evaluates to true



```python
my_list = [1, 2, 3]
for num in my_list:
    print(num)

i = 0
while (i < len(my_list)):
    print(my_list[i])
    i += 1
```

LIST COMPREHENSIONS

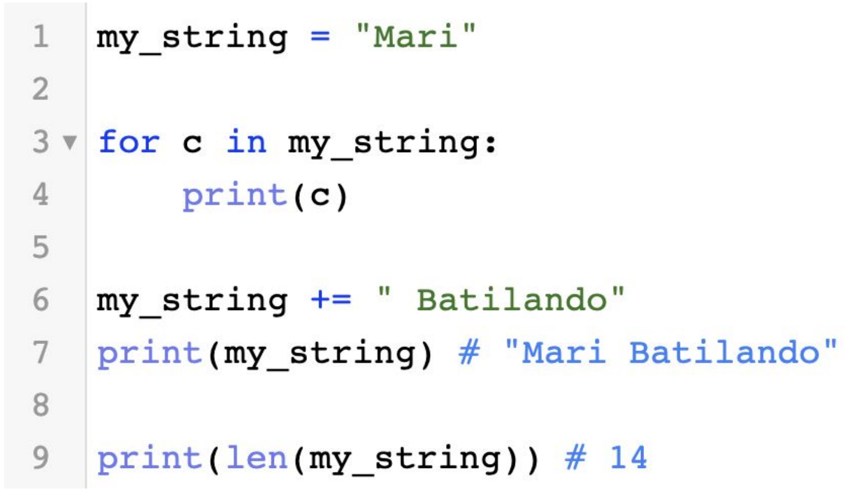
- An easy and concise way to create lists
- Read more about it here



```python
nums = [1, 2, 3, 4, 5]

# verbose-way
squared = []
for n in nums:
    squared.append(n*n)
print(squared) # [1, 4, 9, 16, 25]

# using-list comprehensions
squared = [x*x for x in nums]
print(squared) # [1, 4, 9, 16, 25]
```

STRINGS

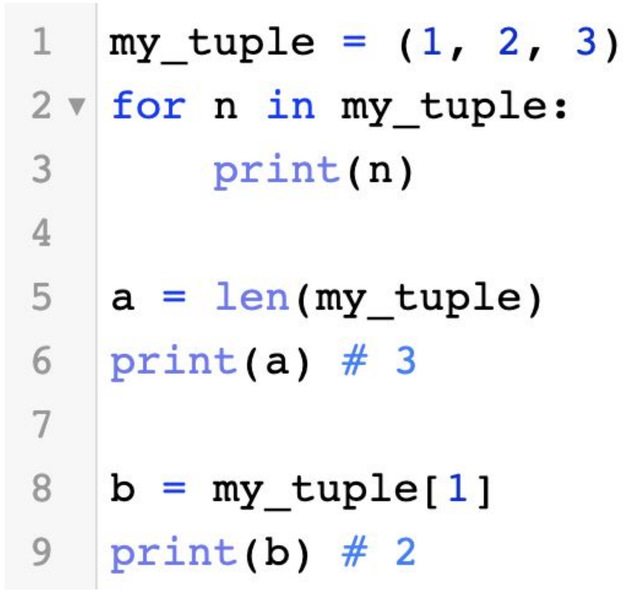
- Similar to a list of characters
- Can do many operations a list can: indexing, iteration
- You can use format strings for string interpolation
- Strings are immutable, so passing it in a function behaves similar to pass-by-value
- See documentation here



```python
my_string = "Mari"

for c in my_string:
    print(c)

my_string += " Batilando"
print(my_string) # "Mari Batilando"

print(len(my_string)) # 14
```

TUPLES

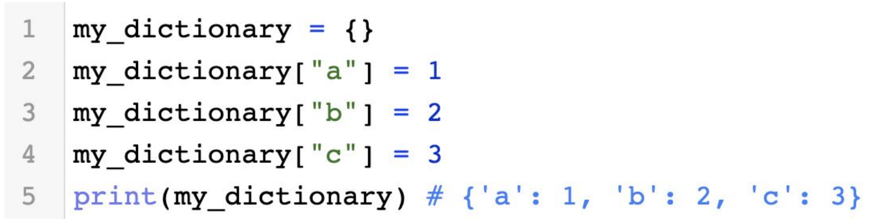
- Basically immutable lists
- Behavior similar to lists, but no mutable operations (e.g. append, remove, etc.)
- Passing tuples in a function behaves similar to pass-by-value
- See documentation here



```python
my_tuple = (1, 2, 3)
for n in my_tuple:
    print(n)

a = len(my_tuple)
print(a) # 3

b = my_tuple[1]
print(b) # 2
```

DICTIONARIES

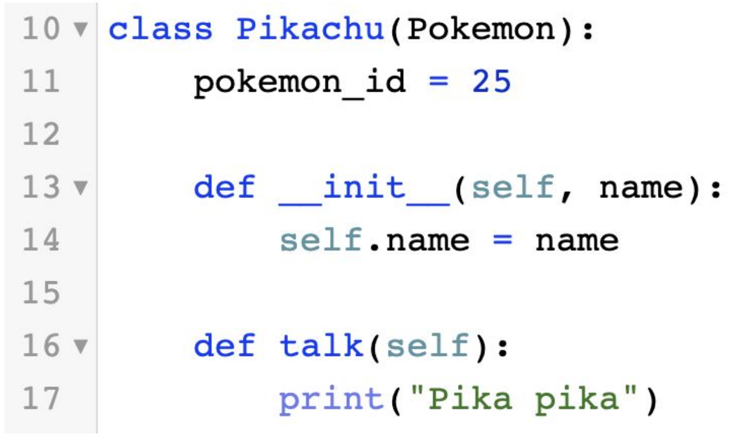
- Stores key-value pairs
- Strings and Numbers are usually keys, though other data types can also be keys
- Value can be any data type
- Dictionaries can contain heterogeneous key-value pairs
- Dictionaries are mutable, so passing it in a function behaves similar to pass-by-reference
- Dictionaries are collections, so you can iterate through them using a for-loop
- See documentation here



```python
my_dictionary = {}
my_dictionary["a"] = 1
my_dictionary["b"] = 2
my_dictionary["c"] = 3
print(my_dictionary) # {'a': 1, 'b': 2, 'c': 3}
```

CLASSES

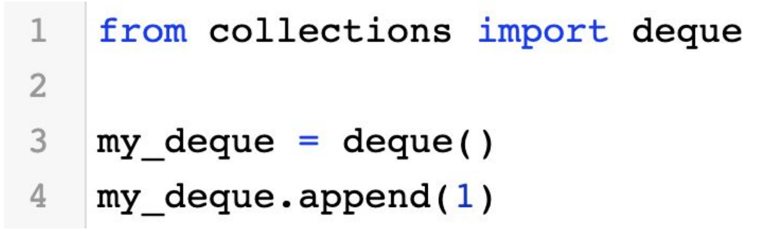
- Python allows creation of new types via classes
- Classes support instance and class variables/methods and they also can have inheritance
- Fun-fact: Python doesn't support private variables/methods
- See documentation here



```python
class Pikachu(Pokemon):
    pokemon_id = 25

    def __init__(self, name):
        self.name = name

    def talk(self):
        print("Pika pika")
```

IMPORTING LIBRARIES

- Use the **import** keyword to import other Python libraries/modules
- Combine it with **from** keyword to import a specific class/variable



```python
from collections import deque

my_deque = deque()
my_deque.append(1)
```

Coding Exercises