Intermediate Python

Unit 1: Class Introduction

Let's start with introductions

- Tell us your name,
- An interest or a hobby,
- What you've been up to since the last class,
- And what you're hoping to get out of this class.

What are we learning?

This is a continuation of the Introduction to Programming with Python class, and we'll continue where we left off in that class and finish learning about basic concepts in modern programming and dive deeper into web programming.

Who is this class for?

It's for anyone that is interested in going beyond the basics of programming.

Who is this class for?

- For those that want to become software engineers
- For those that want to use programming in your current field/job
- For those that are just curious

Class schedule

- We meet Monday and Thursday
- Mondays we meet at the Refugee Center
- Thursdays we meet on Zoom
- Class starts at 6pm and ends at 7:30pm
- This course lasts 15 weeks
- Last day of class is June 29th

Let's talk about the tools we'll use in the class.

Zoom

Thursdays classes will be held over Zoom.

Zoom

Zoom meetings will be recorded and made available in Canvas for you to watch whenever.

Zoom etiquette

When we meet using Zoom, please turn your **cameras on** and **mute yourself** after joining.

Slack

We'll use Slack to communicate.

Slack

Install the Slack app on your computer and on your phone.

Slack

Use your real name in your Slack profile so it's easy to find you.

Canvas

All of the content related to this class will be in Canvas.

Slack and Canvas

You should have received an invitation to Slack and to Canvas. Create your accounts if you haven't and let me know if you have any issues.

Visual Studio Code

Visual Studio Code, or VS Code, is a text editor. A text editor is a tool used to write code.

Syllabus

This course is divided into sections, each covering a new concept.

Syllabus: sections

1. Class Introduction and review
2. Dictionaries
3. Introduction to Flask
4. Classes
5. File IO
6. Error handling
7. Modules

Syllabus: homework

- We'll have homework assignments every week
- I'll specify when the assignment is due
- Homework can be found in Canvas
- Should be submitted through Canvas

How can you succeed in this class?

- Do your homework
- Write the code yourself, no copy/paste
- Run all of the code that you write
- Take notes
- Use Slack
- Ask questions
- Work with your classmates

How can you fail in this class?

- You don't write code
- You don't take notes
- You don't ask questions

Review

Review topics

- Input and output
- Operators
- Conditionals
- Lists
- Loops
- Functions

Input and output

What is it?

Input is what allows users to interact with our programs.

What is it?

Input allows information from the outside world to be used in our programs.

How do we do it?

We use the `input` function to get what the user typed into their computers.

Remember that...

Remember that the `input` function always returns a `string`, even if the user typed a number.

Converting strings

But we can use `int` and `float` to convert strings to integer and real numbers.

What is it?

Output is how we are able to show data to the user.

How do we do it?

We use the `print` function to print a string to the terminal screen.

Input and output, an example

```python
user_input = input("What is your age? ")
user_age = int(user_input)

print("Next year you will be " + str(user_age + 1))
```

Operators

Operators

- Arithmetic operators
- Equality operators
- Comparison operators
- Boolean operators

Operators

Different types of operators let us do different things.

Arithmetic operators

Arithmetic operators allow us to perform basic arithmetic.

Arithmetic operators

Arithmetic operators return a number (an integer or float).

Arithmetic operators, an example

```
race_time = 60.34
penalty_time = 10

total_time = race_time + penalty_time
```

List of arithmetic operators

- `*`, multiplication
- `%`, modulo
- `/`, floating point number division
    - `10 / 3` results in `3.3333333333333335`
- `//`, integer number division
    - `10 // 3` results in `3`
- `+`, addition
- `-`, subtraction

Equality operators

Equality operators allow us to compare whether two values are equal to each other (or not equal to each other).

Equality operators

Equality operators return booleans (`True` or `False`)

List of equality operators

- `==`, equal to
- `!=`, not equal to

Comparison operators

Comparison operators help us determine when a value is greater than or less than another value.

Comparison operators

Comparison operators return booleans (`True` or `False`)

List of comparison operators

- `>`, greater than
- `>=`, greater than or equal to
- `<`, less than
- `<=`, less than or equal to

Boolean operators

Boolean operators allow us to create expressions that operate on multiple boolean values with a single result.

List of boolean operators

- `and`, both sides must be `True`
- `or`, one of the sides must be `True`
- `not`, the result is negated

Conditionals

Conditionals

Conditionals allow parts of our code to run only when certain conditions are met.

For example...

For example, when we detect that a user has won the game, we can print a message to the screen and stop the program.

`if` statements

Python has the `if` statement.

`if` statements

`if` statements need a condition and an indented block of code which runs when the condition is `True`.

Conditionals, an example

```python
# ...

if user_won_the_game:
    print("Congrats, you win!")
    stop_game = True

# ...
```

The condition

The condition should be a boolean expression.

The condition should be a boolean expression

This means you can use any single boolean value, or an expression with operators that result in a boolean value.

`elif` and `else` clauses

`if` statements can be extended with `elif` and `else` clauses.

`elif`

`elif` clauses also need a condition and an indented block of code to run when the condition is `True`.

`elif`

`elif` clauses only run when the previous `if` and `elif` conditions are all false.

`else`

`else` clauses do not need a condition, but they do need an indented block of code to run.

`else`

`else` clauses are the "default" option and only run when the previous `if` and `elif` conditions are all false.

The structure of `if` statements

`if` statements must start with the `if` keyword.

The structure of `if` statements

`elif` and `else` are optional.

The structure of `if` statements

There can be many `elif` clauses in an `if` statement.

The structure of `if` statements

There can only be one `else` and it must go at the end.

Lists

Lists

Lists are values.

Lists

Lists are values that contain multiple values within them.

Lists

Lists are containers of values.

A list of strings

```python
driver_list = [
    "Max Verstappen",
    "Charles Leclerc",
    "Lewis Hamilton",
    "George Russell",
    "Lando Norris"
]
```

A list of integers

```
score_list = [
    454,
    308,
    305,
    275,
    246
]
```

A list of lists

```python
driver_and_score_list = [
    ["Max Verstappen", 454],
    ["Charles Leclerc", 308],
    ["Sergio Perez", 305],
    ["George Russell", 275],
    ["Carlos Sainz", 246]
]
```

Accessing items from a list

Use the square brackets to access items from a list.

Accessing items from a list, an example

```python
driver_list = [
    "Max Verstappen",
    "Charles Leclerc",
    "Lewis Hamilton",
    "George Russell",
    "Lando Norris"
]

driver_list[0] # "Max Verstappen"
driver_list[1] # "Charles Leclerc"
driver_list[2] # "Lewis Hamilton"
```

Remember that...

Remember that the first item in a list is in position 0.

The index

The position of an item is referred to as its "index" in a list.

Adding and removing items from a list

The `push` and `pop` methods are used to add and remove items.

`push` and `pop`

`push` adds an item to the end of the list.

`push` and `pop`

`pop` removes and returns the item from the end of the list.

Hint: randomly picking an item from a list

```python
import random

choices = ["rock", "paper", "scissors"]
computers_choice = random.choice(choices)

print("The computer chose " + computers_choice)
```

Loops

Loops

Loops allow parts of our code to run multiple times.

Loops

Loops allow parts of our code to run zero or more times.

Loops in Python

`while` and `for`.

Loops in Python

- `while` loops run for as long as the condition is `True`.
- `for` loops run for each item in the list.

for loop, an example

```
driver_and_score_list = [
    ["Max Verstappen", 454],
    ["Charles Leclerc", 308],
    ["Sergio Perez", 305],
    ["George Russell", 275],
    ["Carlos Sainz", 246],
]

for driver_info in driver_and_score_list:
    print(driver_info[0] + " scored " + str(driver_info[1]) + " points")
```

`break` and `continue`

`break` and `continue` let us skip a single iteration or prevent a loop from running again.

`break`

`break` stops a loop, even if its condition is still true or there are more items to iterate through.

`break`

`break` is useful when we want to stop a loop early, or stop a loop that is running for ever (like `while True: ...`)

`continue`

`continue` stops the current iteration and "jumps" to the top of the loop. The loop continues if it can.

Functions

What are they?

Functions are units of code that we can re-use in our program.

Code that we can re-use

This means that instead of writing the same code multiple times, we can write it once inside of a function and re-use by calling the function.

Anatomy of a function

Functions have a **name**.

Function names

A function's name is important since we'll need to know it to make it run.

Anatomy of a function

Functions can have zero or more **parameters**.

Function parameters

A function's parameters are its "inputs".

Function parameters

```python
def show_greeting(name):
    print("Hello " + name + ", welcome to class")

show_greeting("Ryan")               # prints "Hello Ryan, welcome to class"
show_greeting("Olivia")             # prints "Hello Olivia, welcome to class"
```

Anatomy of a function

Functions can **return** zero of more values.

Function return value

A function "returns" a value with the `return` keyword.

Function return value

A function can have more than one `return` statement in its body, but it will stop running as soon as it hits the first `return`.

Function return value

```python
def show_greeting(name):
    # Previously: print("Hello " + name + ", welcome to class")
    return "Hello " + name + ", welcome to class"
```

Function return value

```python
def show_greeting(name):
    # Previously: print("Hello " + name + ", welcome to class")
    return "Hello " + name + ", welcome to class"

greeting1 = show_greeting("Ryan")

print(greeting1)                    # prints "Hello Ryan, welcome to class"
print(show_greeting("Olivia"))  # prints "Hello Olivia, welcome to class"
```