ICS 2203 Internet Application Programming

Introduction to Python

Out line

- Introduction
- Python
- Variables
- Formatting Strings
- Conditions
- Sequences
  - Strings
  - Lists
  - Tuples
  - Sets
  - Dictionaries
  - Loops
- Functions
- Modules
- Object-oriented programming
- Functional programming
- Decorators
- Lambda Functions
- Exceptions

Introduction -Python

- very powerful and widely-used language
- **Interpretable** language: C or Java -compiled
- Allow's developers to quickly build fairly complicated web applications.
- In this course, we'll be using Python 3.
- **Hello, world program.** in Python, it would look like this:

```
print("Hello, world!") #example 1: hello.py
```

break down

- a print **function** built in to the python language, takes an **argument** in parentheses, and displays that argument on the command line.

- To write and run this program on your computers,
  - type this line into your text editor of choice, and then save the file as something.py.
  - head over to your terminal, navigate to the directory containing your file, and type python something.py.

variables

- Python variable types are inferred:- each variable does have a type, but not explicitly stated when creating the variable.
- assigning a value to a variable in Python, the syntax looks like this:

```
a = 28
b = 1.5
c = "Hello!“
d = True
e = None
```

common variable types

- **Int**: An integer
- **float**: A decimal number
- **str**: A string, or sequence of characters
- **bool**: A value that is either True or False
- **NoneType**: A special value (None) indicating the absence of a value.

Example 2: name.py

```
name = input("Name: ")
print("Hello, " + name)
```

- Takes input from the user and say hello to that user.
- Uses built in function called input- it displays a prompt to user, and returns whatever the user provides as input.

- NOTE:
  - **first line**: the variable name is assigned to whatever the input function returns.
  - **second line**: the + operator is used to combine, or concatenate, two strings. In python, the + operator can be used to add numbers or concatenate strings and lists.

Formatting Strings

- **formatted strings**, or f-strings for short. Provide easier ways to work with strings
- For example, instead of using `"Hello, " + name` write `f"Hello, {name}"` for the same result.
  - *The f before the string indicates we are using formatted strings.*
- We can even plug a function into this string if we want, and turn our program above into the single line:

```
print(f"Hello, {input("Name: ")}")
```

Conditions

- Example: change output depending on the number a user types in:

```
num = input("Number: ")
if num > 0:
    print("Number is positive")
elif num < 0:
    print("Number is negative")
else:
    print("Number is 0")
```

How it works:

- conditionals in python contain:
  - a keyword (if, elif, or else) and then (except in the else case)
  - a boolean expression, or an expression that evaluates to either True or False.
  - Then, all of the code to run if a certain expression is true is indented directly below the statement.

  NOTE: Indentation is required as part of the Python syntax.

Exception

- what happens when an error occurs while we're running python code.
- Gives error messages - interpreting these errors, is a very valuable skill to have.
- For the above:
  - It ran's into a TypeError, which generally means Python expected a certain variable to be of one type, but found it to be of another type. In this case, the exception says it is wrong to use the > symbol to compare a str and int, and then above we can see that this comparison occurs in line 2.
  - Input function always returns a string.-
    - **type casting:** int(input("Number:"))

Sequences

- One of the most powerful parts of the Python language is its ability to work with sequences of data in addition to individual variables.
- There are several types of sequences that are similar in some ways, but different in others.
- Terms used with sequences: mutable/immutable and ordered/unordered.
  - **Mutable** : once a sequence has been defined, we can change individual elements of that sequence,
  - **Ordered** : the order of the objects matters.

Strings

- Ordered: Yes
- Mutable: No

- A string is a sequence of characters. This means we can access individual elements within the string! For example:

```
name = "Harry"
print(name[0])
print(name[1])
```

Lists

- Ordered: Yes
- Mutable: Yes
- A Python list allows you to store any variable types.
- create a list using square brackets and commas, as shown below.
- Similarly to strings, we can print an entire list, or some individual elements. We can also add elements to a list using append, and sort a list using sort

Example list: list.py

```
# This is a Python comment
names = ["Harry", "Ron", "Hermione"]
# Print the entire list:
print(names)
# Print the second element of the list:
print(names[1])
# Add a new name to the list:
names.append("Draco")
# Sort the list:
names.sort()
# Print the new list:
print(names)
```

Tuples

- Ordered: Yes
- Mutable: No

- Generally used when you need to store just two or three values together, such as the x and y values for a point.
- In Python code, we use parentheses:

```
point = (12.5, 10.6)
```

Sets

- Ordered: No
- Mutable: N/A

- Sets are different from lists and tuples in that:
  - they are unordered.
  - while you can have two or more of the same elements within a list/tuple, a set will only store each value once.
- define an empty set using the set function.
- use add and remove to add and remove elements from that set,
- the len function to find the set's size.
  - Note that the len function works on all sequences in python.

Example : myset.py

```
# Create an empty set:
s = set()
# Add some elements:
s.add(1)
s.add(2
s.add(3)
s.add(4)
s.add(3)
s.add(1)
```

```
# Remove 2 from the set
s.remove(2)
# Print the set:

print(s)
# Find the size of the set:
 print(f"The set has {len(s)} elements.")

 """ This is a python multi-line comment:
Output: {1, 3, 4}
The set has 3 elements. """
```

Dictionaries

- **Ordered**: No
- **Mutable**: Yes
- Python Dictionaries or dicts,
- A set of key-value pairs, where each key has a corresponding value,
- In Python, we use curly brackets to contain a dictionary, and colons to indicate keys and values.

Example

```
# Define a dictionary
houses = {"Harry": “Mansion", “Wamboi": “Bungalow"}
 # Print out Harry's house
print(houses["Harry"])
 # Adding values to a dictionary:
 houses[“Lydia"] = “Mansion“
 # Print out Lydia's House:
 print(houses[“Lydia"])
""" Output:
Mansion
 Mansion """
```

Loops

- In Python, they come in two main forms: for loops and while loops.
- For loops are used to iterate over a sequence of elements, performing some block of code (indented below) for each element in a sequence. For example, the following code will print out the numbers from 0 to 5:

```
for i in [0, 1, 2, 3, 4, 5]:
    print(i)
```

- using the python range function, allows us to easily get a sequence of numbers. The following code gives the exact same result as our code from above:

```
for i in range(6):
    print(i)
```

For loop in a list

- For loop can work for any sequence! For example, if we wish to print each name in a list, we could write the code below:

```
# Create a list:
names = ["Harry", "Ron", "Hermione"]


# Print each name:
for name in names:
    print(name)
```

loop through each character in a single name!

```
name = "Harry“
 for char in name:
 print(char)
```

Functions- definitions

- Example: function that takes in a number and squares it:

```
def square(x):
    return x * x #square function
```

Notice:

- the def keyword to indicate a function definition.
- the function take's in a single input called x
- the return keyword indicate's what the function's output should be.

Calling functions

- Example:

```
for i in range(10):
    print(f"The square of {i} is {square(i)}") #calling
```

Modules

- For large projects, it become’s useful to be able to write functions in one file and run them in another.
- Example:
  - consider creating one file called functions.py with the function square() as above:
  - another file called square.py with the calling code:
- To run this code make the files be aware of each other. (by default, Python files don’t know about each other).
- How: explicitly import the square function from the functions module we just wrote.

Importing a function

```
from functions import square
 for i in range(10):
        print(f"The square of {i} is {square(i)}")
```

Importing an entire functions module

```
from functions
 for i in range(10):
        print(f"The square of {i} is {function.square(i)}")
```

Built-in modules

- There are many built-in Python modules we can import such as math or csv that give us access to even more functions.
- Additionally, we can download even more Modules to access even more functionality!

- We'll spend a lot of time using the Django Module, which we'll discuss in the next lecture.

Object-Oriented Programming

a programming paradigm, or a way of thinking about programming, that is centered around objects that can store information and perform actions.

Classes

- User defined types.
- A Python Class is essentially a template for a new type of object that can store information and perform actions.
- Here's a class that defines a two-dimensional point:

```
class Point():
    # A method defining how to create a point:
    def __init__(self, x, y):
        self.x = x
        self.y = y
```

Note: the keyword self used to represent the object we are currently working with. self should be the first argument for any method within a Python class.

Creating objects

```
p = Point(2, 8) // initiates an object of type point
print(p.x)
 print(p.y)
```

Class that represents an airline flight

```
class Flight()
 # Method to create new flight with given capacity
def __init__(self, capacity):
            self.capacity = capacity
             self.passengers = []
# Method to add a passenger to the flight:
def add_passenger(self, name):
             if not self.open_seats():
                        return False
            self.passengers.append(name)
            return True
 # Method to return number of open seats
  def open_seats(self):
            return self.capacity - len(self.passengers)
```

Instantiating objects for the above class.

```
# Create a new flight with o=up to 4 passengers
 flight = Flight(4)
 # Create a list of people
people = ["Harry", "Ron", “Mwandoe", “Karisa“ , ”Kimeu”]
# Attempt to add each person in the list to a flight
for person in people:
if flight.add_passenger(person):
 print(f"Added {person} to flight successfully")
 else:
 print(f"No available seats for {person}")
```

Functional Programming

- Functional Programming Paradigm - functions are treated as values just like any other variable.
- Supported in Python

Decorators

- Made possible by functional programming.
- Higher-order functions can modify other functions.
- Example: *a decorator that announces when a function is about to begin, and when it ends*. Applied using the @ symbol.

```
def announce(f):
    def wrapper():
        print("About to run the function")
        f()
        print("Done with the function")
    return wrapper

@announce
def hello():
    print("Hello, world!")

hello()
```

Lambda Functions

- provide another way to create functions in python.
- For example, if we want to define the same square function we did earlier, we can write:

```
square = lambda x: x * x
```

  - Where the input is to the left of the : and the output is on the right.

- useful when we don’t want to write a whole separate function for a single, small use.

For example: sorting dictionary items

- sorting some objects where it's not clear at first how to sort them. Consider a list of people, with names and houses that need's sorting:

```
people = [
    {"name": "Harry", "house": “Flat"}, {"name": “Kim", "house": “Bungalow"},
    {"name": “Mercy", "house": “Mansion"}]

people.sort()

print(people)
```

This lead's to an error:
- because Python doesn't know how to compare two Dictionaries to check if one is less than the other.

Solution:

- Include a key argument to the sort function, which specifies which part of the dictionary to use to sort:

```
people = [{"name": "Harry", "house": "Flat"},
{"name": "Kim", "house": "Bungalow"},    {"name":
"Mercy", "house": "Mansion"}]

def f(person):
    return person["name"]

people.sort(key=f)

print(people)
```

Try it out!

Better solution with lambda function:

- In the previous, an entire function was written that was only used once.
- The code can be more readable using lambda function as below:

```
people = [{"name": "Harry", "house": "Flat"},
{"name": "Kim", "house": "Bungalow"},
    {"name": "Mercy", "house": "Mansion"}]
people.sort(key=lambda person: person["name"])
print(people)
```

Exceptions handling

- Example: The code is okay until you try to divide by zero.

```
“””taking two integers from the user,
and attempting to divide them:”””
x = int(input("x: "))
y = int(input("y: "))

result = x / y

print(f"{x} / {y} = {result}")
```

```
x: 5
y: 0
Traceback (most recent call last):
  File "exceptions.py", line 4, in <module>
    result = x / y
ZeroDivisionError: division by zero
```

Exemption handling cont…

- The following block of code: try to divide the two numbers, except when we get a ZeroDivisionError:

```
import sys

x = int(input("x: "))
y = int(input("y: "))

try:
    result = x / y
except ZeroDivisionError:
    print("Error: Cannot divide by 0.")
    # Exit the program
    sys.exit(1)

print(f"{x} / {y} = {result}")
```

Output on zero division attempt:

```
x: 5
y: 0
Error: Cannot divide by 0.
```

Exception handling:

- Say the user enters non-numbers for x and y: - an error occurs.
- Solve as below:

```
import sys

try:
    x = int(input("x: "))
    y = int(input("y: "))
except ValueError:
    print("Error: Invalid input")
    sys.exit(1)
```

```
try:
    result = x / y
except ZeroDivisionError:
    print("Error: Cannot divide by 0.")
    # Exit the program
    sys.exit(1)

print(f"{x} / {y} = {result}")
```

The end

Next Python Django module!