Python Part 2

Recap

- Python is an interpreted language
- Indentation used to define scope
- Python packages managed using pip
- Most procedures already exist in available packages (modules)
- Variables defined when values are assigned
- Input and output done using print() and input() functions
- Logical flow controls and loops similar to C++

Formatting strings

- Text can be formatted to use variables using {}

```
num1 = 8
num2 = 3
mytext = "My numbers are {} and {}"
print(mytext.format(num1,num2))
```

Collections

- Python collections can include mismatched data types
- 4 kinds of collections available:

	Ordered	Changeable	Duplicates allowed
List	✓	✓	✓
Tuple	✓		✓
Set			
Dictionary	✓	✓	

Lists

- Initialized using square brackets

```
mylist = ["lion","tiger","bear",17,False,"lion"]
```

- Print list items using print(mylist)
- Get length of list using len(mylist)
- Items accessed using indices: [0] is first item, [-1] is last item
- Add item using mylist.append("dog")
- Remove item using mylist.remove("bear")
- Sort list using mylist.sort()
- Join lists using mylist + mylist2

Tuples

- Initialized using parentheses

```
mytuple = ("lion","tiger","bear",17,False,"lion")
```

- Print tuple items using print(mytuple)
- Get length of tuple using len(mytuple)
- Items accessed using indices: [0] is first item, [-1] is last item
- Items cannot be added, removed, or changed
    - Modify tuples by casting to a list and casting back to a tuple
- Join tuples using mytuple + mytuple2

Sets

- Initialized using curly brackets

```
myset = {"lion","tiger","bear",17,False}
```

- Print set items using print(myset)
- Get length of set using len(myset)
- Items accessed by looping
- Check if item is in set using <item> in myset
- Add item using myset.add("dog")
- Remove item using myset.remove("bear")
- Join sets using myset.update(myset2) or myset3 = myset.union(myset2)

Dictionaries

- Data stored in key:value pairs

```
mydictionary = {"species":"tiger",
                "weight":300,
                "isFish":False}
```

- Print dictionary items using print(mydictionary)
- Get length of dictionary using len(mydictionary)
- Items accessed (read and write) using key: mydictionary["species"]
- Add item using key or mydictionary.update({"color":"orange"})
- Remove item using mydictionary.pop("isFish") or del mydictionary["isFish"]

Functions

- Python functions are similar to C++ functions
- Define and call functions using:

```
def myfunction(x,y):
    print("In myfunction")
    return x**y

myfunction(2,3)
myfunction(y=5,x=2)
```

- High end python programming uses lambda function - beyond our scope

Functions default arguments

- Define functions with default argument values:

```
def myfunction(x,y=2):
    print("In myfunction")
    return x**y

myfunction(3,4)
myfunction(x=3)
myfunction(3)
```

Functions arbitrary arguments

- Functions can be defined with arbitrary number of arguments
- Arguments will be received as a tuple

```
def myfunction(*args):
    for x in args
        print(x)

myfunction(3,4)
```

Functions arbitrary keyword arguments

- Functions can be defined with arbitrary number of keyword arguments
- Arguments will be received as a dictionary

```
def myfunction(**args):
    print(args["val2"])

myfunction(val1 = 3, val2 = 4)
```

Empty functions

- Function definitions cannot be empty
- Use the pass statement to avoid errors

```
def emptyfunction():
    pass
```

Modules (packages)

- Any *.py file can be used as a module that can be used with import
  - Module name is the same as the file minus .py
- Use dir(<module>) to list all member methods and variables
- Import only one function or variable from module to reduce overhead:

```
from mymodule import function1
```

Classes - basics

- Classes are defined using:

```
class myClass:
    var1 = 7
```

- Create instance of class using:

```
x = myClass()
```

- Access member variables using:

```
print(x.var1)
```

Classes - initialization

- Classes are initialized using __init__() method:

```
class myClass:
    def __init__(self, first,second):
        self.var1 = first
        self.var2 = second
```

- Create instance of class using:

```
x = myClass("apple",7)
print(x.var1)
print(x.var2)
```

Classes - printing

- By default, printing an object of a user defined class has little information

<__main__.myClass object at 0x1005b85d0>

- Define __str__() function to provide string with information

```
class myClass:
    def __str__(self):
        return f"{x.var1} and {x.var2}"

x = myClass("apple",7)
print(x)
```

Classes - inheritance

- Parent class:

```
class myClass:
    def __init__(self, first,second):
        self.var1 = first
        self.var2 = second
```

- Derived class:

```
class myDerivedClass(myClass):
    pass
```

- Additional members can be added in place of pass

Resources

- https://www.w3schools.com/ - Great online learning resource
- https://www.youtube.com/@codebreakthrough - Excellent tutorial videos
- https://wiki.python.org/moin/BeginnersGuide - Good documentation
- https://learn.microsoft.com/en-us/windows/python/ - For Windows users
- https://www.python.org/
- https://stackoverflow.com/ - Ask questions to experts