Python 1 - Overview

Bootcamp will cover Python fundamentals while making a music playlist program

- Evaluating primitive types in python: type()
- Declaring variables and variable declaration conventions: =
- Math Operators and string concatenation: (+ , - , * , /,%)
- IF and WHILE statements with conditional operators: (==, >, >=, break)
- User input: input()
- Data collections - Lists: ([ ], append(), insert(), del, pop(), len(), sort())
- Data collections - Dictionaries: ({ },[ ], insert(), del, clear(), keys(), values())
- Declaring custom functions: def, return
- Classes and object oriented programming: class(), __init__(), methods
- Automating with FOR loops: for, in
- Use Case: Song Playlist Program

Our program will:

1. Display a menu
2. Add new songs from user input
3. Display the playlist
4. Remove songs from the playlist

Data Types

- Four primitive types in Python
  1. Integers
  2. Booleans
  3. Floats
  4. Strings
- Types may be changed using int(), str(), float(), and bool() methods

In [8]:

```
#The code to output the type of data for (1) has already been filled in for you as

print(type(3))
print(str(3)) # "3"
print(float(3))
print(bool(3))
```

```
<class 'int'>
3
3.0
True
```

In [9]:

```
# Edit the following lines of code so that they would output the types of data for

type("Drake")
```

Out[9]:

```
str
```

In [10]:
```
type(True)
```

Out[10]: bool

In [11]:
```
type(3.14)
```

Out[11]: float

Variables

- May consist of letters, numbers, and underscores, but not spaces.
  - **Cannot start with a number.**
- Avoid using Python keywords (for, if, and, or, etc.)
- Be careful when using 1s and lower case ls, as well as 0s and Os.
- Keep it short.

In [0]:
```
# In the code below, the variable `num1` has been assigned an integer value of 10.

num_songs = 10
```

In [13]:
```
print(num_songs)
```

```
10
```

In [0]:
```
# Create variable `num2`by assigning it a value of 30.
```

Math Operators

- Addition, Subtration, Multiplication and Division may be done using basic math operators (+ , - , * , /,%).
- Many built-in string methods (title, upper, lower, index, split).
- Python will also try to interpret your code with other data types
  - (+) may be used with strings!

In [14]:
```
# Create two variables, song1_dur and song2_dur that have float values representin
# Create a new variable whose value is the sum of the duration of both songs
song1_dur = 3.40
song2_dur = 2.51
tot_dur = song1_dur + song2_dur
print(tot_dur)
```

```
5.91
```

In [0]:
```
#The string variables "i" and "name" have been defined for you.
current = "Currently Playing:"
artist_name = "drake"
```

In [46]:
```
#Print output by using the addition operator.
current_artist = current + " " + artist_name

print(artist_name.title())
print(artist_name.lower())
print(artist_name.upper())

print(current_artist)
print(current_artist.index("drake"))
print(current_artist.split(" "))
```

```
Drake
drake
DRAKE
Currently Playing: drake
19
['Currently', 'Playing:', 'drake']
```

In [48]:
```
#A few ways to combine strings and variables
print(f"Currently Playing: {artist_name.title()}")
```

```
Currently Playing: Drake
```

In [0]:
```
#Boolean can only have one of two values. Either they are "True" or "False".
#Variables "yes" and "no" have been assigned boolean variables of "True" and "Fals
yes = True
no = False
```

IF and WHILE Statements

- Will only run indented code if condition is true
- Make use of **conditional operators** to create tests
  - (==) will return true if both variables are equal
  - (>) will return true if left variable is larger
  - (>=) will return if left variable is larger or equal to right variable
- IF will only run indented code once, WHILE will run indented code until condition is no longer true

In [50]:
```
#Boolean variables are generally used for conditional statements such as an if sta
#The below lines of code uses boolean variables to determine whether or not the fo
if yes:
  print("True Statement!")

if no:
  print("Will not print")
```

```
True Statement!
```

In [0]:
```
# The below code is asking if 3 is greater than 2 and if so print "Three is greate
num_songs = 1

if num_songs < 11:
  print("Song Added")
```

```
Song Added
```

In [0]:
```
tot_dur = 0
```

In [65]:
```
# if else statments can also be used with math or anything really!
song_dur = 5

if tot_dur < 20.00:
    tot_dur += song_dur
    print(f"Song Added. Your playlist is {tot_dur} minutes long.")
else:
    print("Allowed time exceeded!")
```

```
Allowed time exceeded!
```

In [72]:
```
limit = 10
num_songs = 0

while num_songs < limit:
  print(num_songs)
  num_songs += 1

num_songs = 0

while True:

  if num_songs == 8:
    break

  print(num_songs)
  num_songs += 1
```

```
0
1
2
3
4
5
6
7
8
9
0
1
2
3
4
5
6
7
```

Lists

- Collection of items in a particular order
- Indexing (order) starts from **0**
- Accessing items in a list can be done with square brackets ([ ])
- Items can be easily added to lists using append() and insert() methods

In [80]:

```
#Lists are a collection of data. The lists start at 0.

artists = ["Drake", "Ariana", "Billy", "Kanye"]
print(artists[0])
print(artists[3])
print(artists[0:3])
print(artists[:1])
print(artists[2:])
print(artists[-1])

#Reassign values with square brackets as well
artists[0] = "Maroon5"
print(artists)

#Cannot do artists[4]  = ""
```

```
Drake
Kanye
['Drake', 'Ariana', 'Billy']
['Drake']
['Billy', 'Kanye']
Kanye
['Maroon5', 'Ariana', 'Billy', 'Kanye']
```

In [81]:

```
# add value to end of a list
artists.append("Khalid")
print(artists)

# add vaue to the start of a list
artists.insert(0, "Diplo")
print(artists)

# Return the length of the list
len(artists)

del artists[4]

print(artists)
```

```
['Maroon5', 'Ariana', 'Billy', 'Kanye', 'Khalid']
['Diplo', 'Maroon5', 'Ariana', 'Billy', 'Kanye', 'Khalid']
['Diplo', 'Maroon5', 'Ariana', 'Billy', 'Khalid']
```

In [84]:

```
last_artist = artists.pop()
print(f"{last_artist} has been removed")
print(artists)
```

```
Billy has been removed
['Diplo', 'Maroon5', 'Ariana']
```

In [85]:
```
# lists can contain any type of data
mix_list = ['Drake', 3.45, True, "Motion"]
print(mix_list)
print(mix_list[3])
```

```
['Drake', 3.45, True, 'Motion']
Motion
```

In [102]:
```
print(f"{mix_list[0]} - {mix_list[3]}")
```

```
Drake - Motion
```

Dictionaries

- Collection of key-value pairs
- No positions as with lists, values stored at specific key
  - keys can be of any data type
- Accessing values in a dictionary can still be done with square brackets ([ ])
- Declared using braces ({ })

In [125]:
```
# collection of "data" which is unordered, changeable and indexed. They have keys
song_dict = { "name": "Motion", "artist": "Drake", "duration": 3.40}
print(song)
```

```
{'name': 'Motion', 'artist': 'Drake', 'duration': 3.25}
```

In [126]:
```
song_dict["name"]
```

Out[126]:
```
'Motion'
```

In [127]:
```
song_dict["duration"] = 3.25
print(song["duration"])
```

```
3.25
```

In [130]:
```
song_dict["album"] = "Scorpion"
print(song_dict)
```

```
{'name': 'Motion', 'artist': 'Drake', 'duration': 3.25, 'album': 'Scorpion'}
```

In [132]:
```
del song_dict["album"]
print(song_dict)
```

```
{'name': 'Motion', 'artist': 'Drake', 'duration': 3.25}
```

In [133]:
```
#Dictionary methods return iterables
print(person.items())
print(person.keys())
print(person.values())

# Cannot do print(person.keys[0])
```

```
dict_items([('name', 'Ted'), ('age', 20), ('single', True), ('siblings', 1.3)])
dict_keys(['name', 'age', 'single', 'siblings'])
dict_values(['Ted', 20, True, 1.3])
```

In [140]:
```
## you can use dictionaries and lists in if statments.

if "Drake" in song_dict:
    print("Yes, Drake is one of the keys in the this dictionary")
else:
    print("no")


if "Drake" in mix_list:
  print("Drakes song")
```

```
no
Drakes song
```

For Loops

- Execute a block of code once for each item in collection (List/Dictionary)
- Declare temporary variable to iterate through collection
- Can be used in combination with IF statements

In [142]:
```
for artist in artists:
  print(artist)
```

```
Diplo
Maroon5
Ariana
```

In [143]:
```
#Check if after 2007 and even number

for key, value in song_dict.items():
  print(f"{key}: {value}")
```

```
name: Motion
artist: Drake
duration: 3.25
```

In [148]:
```
for i in range(len(artist)):
  print(i)
```

```
0
1
2
3
4
5
```

Functions

- Named blocks of code that do one specific job
- Prevents rewriting of code that accomplishes the same task
- Keyword *def* used to declare functions
- Variables may be passed to functions

In [141]:
```
# A function is a block of organized, reusable code that is used to perform a sing
def greeting():
  print("Hi!")

greeting()
```

```
Hi!
```

In [145]:
```
def description(name, artist, album):
    print(f"{name} by {artist} from {album}!")

description("motion", "drake", "Scorpion")
```

```
motion by drake from Scorpion!
```

Classes

- Object-orientated programming approach popular and efficient
- Define classes of real-world things or situations
  - Attributes of various data types
  - Functions inside of a class are the same except called methods
  - Methods may be accessed using the dot operator
- Instanciate objects of your classes
- __init()__ method used to prefill attributes
- Capitalize class names

In [0]:
```
class Song():
  def __init__(self, name, artist, duration):
    self.name = name
    self.artist = artist
    self.duration = duration

  def description(self):
    return f"{self.name} by {self.artist}!"
```

In [147]:
```
song1 = Song("Motion", "Drake", 4.40)
print(song1.description())
```

```
Motion by Drake!
```

User Input

- Pauses your program and waits for the user to enter some text
- Variable used with Input() will be a **string** even if user inputs an integer
  - Will need to make use of type casting

In [0]:
```
x = input("Enter your age.\n")
print(f"Entered age is {x}")
```

```
Enter your age.
22
Entered age is 22
```

Putting It All Together

In [151]:

```
!pip install Texttable
#dir(): Attempt to return a list of valid attributes for that object
#dir(table)
import texttable as tt

songs_dict = {}

message = ""
prompt = "\nPlease choose from one of the following options:"
prompt += "\n1. Add A Song"
prompt += "\n2. List Playlist"
prompt += "\n3. Remove a Song"
prompt += "\n4. Quit\n"


while message != "4":
  message = input(prompt)

  if message == "1":
    num_songs = input("How many songs would you like to add?\n")

    for i in range(int(num_songs)):
      prompt2 = "Please enter the name, artist, duration, and album of the song\n"
      name, artist, duration = input(prompt2).split()
      song = Song(name, artist, duration)
      songs_dict[song.name.lower()] = song

    print("Song(s) Added!")

  elif message == "2":

    if len(songs_dict) == 0:
      print("No songs in the playlist!")
    else:
      menu = tt.Texttable()
      menu.add_row(["Name", "Artist", "Duration", "Description"])
      for key, song in songs_dict.items():
        menu.add_row([song.name.title(), song.artist.title(), song.duration, song.
      print(menu.draw())

  elif message == "3":
    prompt3 = "Please enter the name of the song you'd like to remove\n"
    removed_name = input(prompt3)

    if removed_name in songs_dict:
      del songs_dict[removed_name]
      print(f"{removed_name.title()} removed from the playlist.")
    else:
      print("That song is not in the playlist. Please try again.")

  elif message == "4":
    break
  else:
    print("That is not one of the options.")
```

```
Requirement already satisfied: Texttable in /usr/local/lib/python3.6/dist-packag
es (1.6.1)

Please choose from one of the following options:
1. Add A Song
2. List Playlist
3. Remove a Song
```

In [0]: