Condition and Branching

# If statement example

#age = 19

age = 18

#expression that can be true or false

if age > 18:

    #within an indent, we have the expression that is run if the condition is true

    print("you can enter" )

#The statements after the if statement will run regardless if the condition is true or false

print("move on")

# Else statement example

age = 18

#age = 19

if age > 18:

    print("you can enter" )

else:

    print("go see Meat Loaf" )

print("move on")

# Elif statment example

age = 18

if age > 18:

    print("you can enter" )

elif age == 18:

    print("go see Pink Floyd")

else:

    print("go see Meat Loaf" )

print("move on")

# Condition statement example

#album\_year = 1983

album\_year = 1994

if album\_year > 1980:

    print("Album year is greater than 1980")

print('do something..')

# Condition statement example

album\_year = 1983

#album\_year = 1970

if album\_year > 1980:

    print("Album year is greater than 1980")

else:

    print("less than 1980")

print('do something..')

album\_year = 1980

if(album\_year > 1979) and (album\_year < 1990):

    print ("Album year was in between 1980 and 1989")

print("")

print("Do Stuff..")

 Condition statement example

album\_year = 1990

if(album\_year < 1980) or (album\_year > 1989):

    print ("Album was not made in the 1980's")

else:

    print("The Album was made in the 1980's ")

# Condition statement example

album\_year = 1983

if not (album\_year == 1984):

    print ("Album year is not 1984")

# Example of for loop

for i in range(0, 8):

    print(i)

for year in dates:

    print(year)

# Use for loop to change the elements in list

squares = ['red', 'yellow', 'green', 'purple', 'blue']

for i in range(0, 5):

    print("Before square ", i, 'is',  squares[i])

    squares[i] = 'white'

    print("After square ", i, 'is',  squares[i])

Output:

Before square 0 is red

After square 0 is white

Before square 1 is yellow

After square 1 is white

Before square 2 is green

After square 2 is white

Before square 3 is purple

After square 3 is white

Before square 4 is blue

After square 4 is white

# Loop through the list and iterate on both index and element value

squares=['red', 'yellow', 'green', 'purple', 'blue']

for i, square in enumerate(squares):

    print(i, square)

To get index and element at the same time we can use enumerate() function.

count = 1

while count <= 5:

    print(count)

    count += 1

Animals = ["lion", "giraffe", "gorilla", "parrots", "crocodile","deer", "swan"]

new\_animals = []

i = 0 # initialising the count

while (i < len(Animals)):

    animal = Animals[i]

    if len(animal) == 7:

        new\_animals.append(animal)

    i += 1

print(new\_animals)

# Write your code here

print("Multiplication tables for 6")

print("----------------------------")

#the second number in the range is excluded which it does not involved 13

for num in range(1, 13):

    print(f"6 X {num} = {6\*num}")

print("Multiplication tables for 7")

print("----------------------------")

for num in range(1, 13):

    print(f"7 X {num} = {7\*num}")

# Write your code below and press Shift+Enter to execute

# To exercise more you can write a code that copy all orange elements to the new\_square

squares = ['orange', 'orange', 'purple', 'blue ', 'orange']

new\_squares = []

count = 0

while ( count < len(squares) and squares[count] == 'orange'):

    new\_squares.append(squares[count])

    count+= 1

print(new\_squares)

"""while(count < len(squares)):

    if(squares[count] == "orange"):

        new\_squares.append(squares[count])

    count += 1

print(new\_squares)"""

**Functions**

def add(a):

    """

    add 1 to a

    """

    b = a + 1

    print(a, "if you add one", b)

    return(b)

help(add)

# Define functions, one with return value None and other without return value

def MJ():

    print('Michael Jackson')

def MJ1():

    print('Michael Jackson')

    return(None)

# See what functions returns are

print(MJ())

print(MJ1())

# Define the function for combining strings

def con(a, b):

    return(a + b)

Predefined functions

# Use sum() to add every element in a list or tuple together

sum(album\_ratings)

# Show the length of the list or tuple

len(album\_ratings)

# Function example

def type\_of\_album(artist, album, year\_released):

    print(artist, album, year\_released)

    if year\_released > 1980:

        return "Modern"

    else:

        return "Oldie"

x = type\_of\_album("Michael Jackson", "Thriller", 1980)

print(x)

# Example for setting param with default value

def isGoodRating(rating=4):

    if(rating < 7):

        print("this album sucks it's rating is",rating)

    else:

        print("this album is good its rating is",rating)

Making an internal variable global within the function.

artist = "Michael Jackson"

def printer(artist):

    global internal\_var

    internal\_var= "Whitney Houston"

    print(artist,"is an artist")

printer(artist)

printer(internal\_var)

def printAll(\*args): # All the arguments are 'packed' into args which can be treated like a tuple

    print("No of arguments:", len(args))

    for argument in args:

        print(argument)

#printAll with 3 arguments

printAll('Horsefeather','Adonis','Bone')

#printAll with 4 arguments

printAll('Sidecar','Long Island','Mudslide','Carriage')

\*args are packed into tuples.

def printDictionary(\*\*args):

    for key in args:

        print(key + " : " + args[key])

printDictionary(Country='Canada',Province='Ontario',City='Toronto')

\*\*args are packed into a dictionary.

**Exception Handling**

# potential code before try catch

try:

    # code to try to execute

    y= b +5

except:

    # code to execute if there is an exception

    print("undefined variable")

# code that will still execute if there is an exception

# potential code before try catch

try:

    # code to try to execute

except ZeroDivisionError:

    # code to execute if there is a ZeroDivisionError

except NameError:

    # code to execute if there is a NameError

except:

    # code to execute if ther is any exception

else:

    # code to execute if there is no exception

finally:

    # code to execute at the end of the try except no matter what

# code that will execute if there is no exception or a one that we are handling

a = 1

try:

    b = int(input("Please enter a number to divide a"))

    a = a/b

except ZeroDivisionError:

    print("The number you provided cant divide 1 because it is 0")

except ValueError:

    print("You did not provide a number")

except:

    print("Something went wrong")

else:

    print("success a=",a)

a = 1

try:

    b = int(input("Please enter a number to divide a"))

    a = a/b

except ZeroDivisionError:

    print("The number you provided cant divide 1 because it is 0")

except ValueError:

    print("You did not provide a number")

except:

    print("Something went wrong")

else:

    print("success a=",a)

finally:

    print("Processing Complete")

Hadling ValueError

import math

def squareroot (number1):

    try:

        result = math.sqrt(number1)

        return result

    except ValueError:

        print("Invalid input! Please enter a positite integer or a float value")

number = float(input("Please enter a positive or a float number"))

print(squareroot(number))

Handling Generic Exception

def complex\_cal (num):

    try:

        result = num / (num - 5)

        print(f"Result: {result}")

    except Exception as e:

        print("An error has occured during the calculation")

#Test case

user\_input = float(input("Please enter a number"))

complex\_cal(user\_input)

Classes and objects

An instance of object is the realisation of a class. Each object has different attributes.

Methods give you a way to interact with the object; They are functions that interact with objects.

Constructor is used to initialize the object. \_\_init\_\_

# Create a class Circle

class Circle(object):

    # Constructor

    def \_\_init\_\_(self, radius=3, color='blue'):

        self.radius = radius

        self.color = color

    # Method

    def add\_radius(self, r):

        self.radius = self.radius + r

        return(self.radius)

    # Method

    def drawCircle(self):

        plt.gca().add\_patch(plt.Circle((0, 0), radius=self.radius, fc=self.color))

        plt.axis('scaled')

        plt.show()

# Create an object RedCircle

RedCircle = Circle(10, 'red')

# Find out the methods can be used on the object RedCircle

dir(RedCircle)

# Print the object attribute radius

RedCircle.radius

# Set the object attribute radius

RedCircle.radius = 1

RedCircle.radius

# Call the method drawCircle

RedCircle.drawCircle()

# Create a new Rectangle class for creating a rectangle object

class Rectangle(object):

    # Constructor

    def \_\_init\_\_(self, width=2, height=3, color='r'):

        self.height = height

        self.width = width

        self.color = color

    # Method

    def drawRectangle(self):

        plt.gca().add\_patch(plt.Rectangle((0, 0), self.width, self.height ,fc=self.color))

        plt.axis('scaled')

        plt.show()