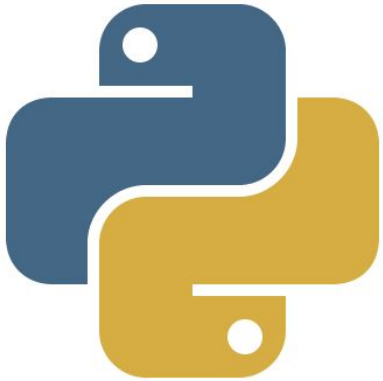


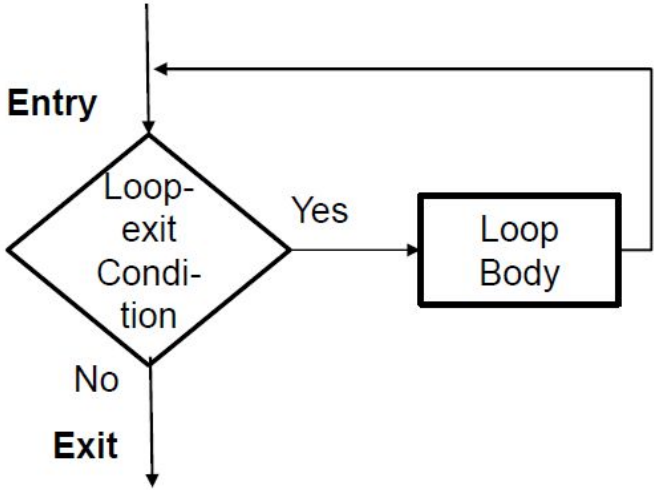
python

TOT - Python Programming Essentials

Day 4

Iteration

- Most useful and powerful structure
- Allows the repetition of instructions or statements in the loop body



Iteration

Parts of the iteration structure

- Loop body instruction(s) or statements which are repeated
- Loop-exit condition the condition to be tested before each repetition

Types

- while loop
- for loop
  - In
  - Range

When to use the while loop and the for loop

- Loops that are dependent on a sentinel value (or indicator) are better coded using a while loop
- The for loop is generally used for traversing and manipulating arrays
- When the number of times that the loop will be executed is known, the for loop provides a convenient shorthand.

Common Loop Applications

Using a loop to accumulate totals

- An accumulator is a variable that “sums up” or accumulates values
- It is similar to a counter whose values change each time the loop is entered. Counters, however, add a fixed value while accumulators accumulate undetermined values.

Common Loop Applications

-Using a loop to validate user entry

- Data entered by a user usually needs validation. It needs to be checked for errors. Incorrect data can lead to unwanted results and could end a program abnormally
- Usual checks for data are:
  - If it is the correct data type
  - For numeric data, if it is within an acceptable range of values

Iteration

Format: while loop

```
while condition:
        statement1
        statement2
```

The statements inside the while loop are executed as long as the condition remains true

Iteration (While Loop)

Sample while loop

```
a= 1
while a < 6:
  print(a)
  a += 1
```

Day 4 Act 1

1. Write a program that will loop the message 20 times. Use while loop only.

   Python while loop number 1
   Python while loop number 2
   ...
   Python while loop number 20

2. Save your file as **lastname_firstname_day4_act1.py**
3. Submit your file in google classroom

Iteration (For Loop)

Sample for For loop

```
names = ["Mike","Ana","Jun"]
for name in names:
    print(name)
```

Iteration (For Loop)

Sample for For loop using in keyword

```
names =
["Mike","Ana","Jun"]
for name in names:
    print(name)
```

Sample for For loop using range keyword

```
for a in range(10,20):
    print(a)

fruits = ['banana', 'apple',
'mango']
for index in range(len(fruits)):
  print('Current fruit :',
fruits[index])
```

Break Statements

- force immediate termination of a loop, bypassing the conditional expression and any remaining code in the body of the loop
- The loop is terminated and program control resumes at the next statement following the loop

Continue Statements

In loops, a continue statement cause control to be transferred directly to the conditional expression that controls the loop.

Pass Statements

The pass statement is a null operation; nothing happens when it executes. The pass is also useful in places where your code will eventually go, but has not been written yet

Day 4 Act 2

1. Write a program that adds two numbers.
2. The program will ask to **enter first** and **second number**
3. The program will display "**The sum of n1** and **n2** is **nTotal**"
4. The program will ask if the user wants to try again. The user will input **Y/y** if Yes and **N/n** if No
5. If **Yes**, refer to step 2.
6. If **No**, the program will display "**Thank you!**".
7. Save your file as **lastname_firstname_day4_act2.py**
8. Submit your file in google classroom

List

The list is a most versatile data type available in Python which can be written as a list of comma-separated values (items) between square brackets.

```
numbers= [1, 2, 3, 4, 5 ];
food = ["cake", "burger", "fries"]

print(numbers[index number])
print(numbers[0]) #1
print(food[1]) #burger
```

Methods to control list and its objects

Method	Description
append()	Appends object obj to list
count()	Count how many times the object occur in list
extend()	Appends the objects of other list
index()	Returns the index number of an object
insert()	Insert object into list using index number
pop()	Removes and returns the last object
remove()	Removes object from a list
reverse()	Reverse objects in place
sort()	Sort objects

Changing value in a list

```
male = ['John','Mike']
print(male[0])
male[0] = "Frank"
print(male[0])
```

Output
John
Frank

Adding object to a list

Using append()

```
numList = [1, 2, 3, 4, 5]
numList.append(99)
print(numList)
```

Output
[1, 2, 3, 4, 5, 99]

Using insert()

```
male = ['John','Mike']
male.insert(1,"Jake")
print(male)
```

Output
['John', 'Jake', 'Mike']

Removing object to a list

Using remove()

```
male = ['John','Mike','Jake']
male.remove("Jake")
print(male)
```

Output
['John', 'Mike']

Using pop()

```
male = ['John','Mike','Jake']
male.pop(1)
print(male)
```

Output
['John', 'Jake']

Arranging object in a list

Using sort()

```
numList = [20, 2, 33, 42, 25]
numList.sort()
print(numList)
```

numList.sort(reverse=True)

Output
[2, 20, 25, 33, 42]

Using reverse()

```
numList = [20, 2, 33, 42, 25]
numList.reverse()
print(numList)
```

Output
[25, 42, 33, 2, 20]

Counting object Checking object index

Using count()

```
numList = [25, 2, 33, 42, 25]
counter = numList.count(25)
print(counter)
```

Output

2

Using index()

```
numList = [11, 2, 33, 42, 25]
indexPosition = numList.index(42)
print(indexPosition)
```

Output

3

Combining lists

Using count()

```
male = ['Bien','John']
female = ['Jayde','Ana']
male.extend(female)
print(male)
print(male[3])
```

Output
['Bien', 'John', 'Jayde', 'Ana']

Day 4 Act 3

1. Write a word bank program
2. The program will ask to **enter a word**
3. The program will **store** the word in a list
4. The program will ask if the user wants to try again. The user will input **Y/y** if Yes and **N/n** if No
5. If **Yes**, refer to step 2.
6. If **No**, Display the **total number of words** and **all the words** that user entered.
7. Save your file as **lastname_firstname_day4_act3.py**
8. Submit your file in google classroom