ICT LITERACY AND COMPETENCY DEVELOPMENT BUREAU

python

PROGRAMMING ESSENTIALS WEBINAR

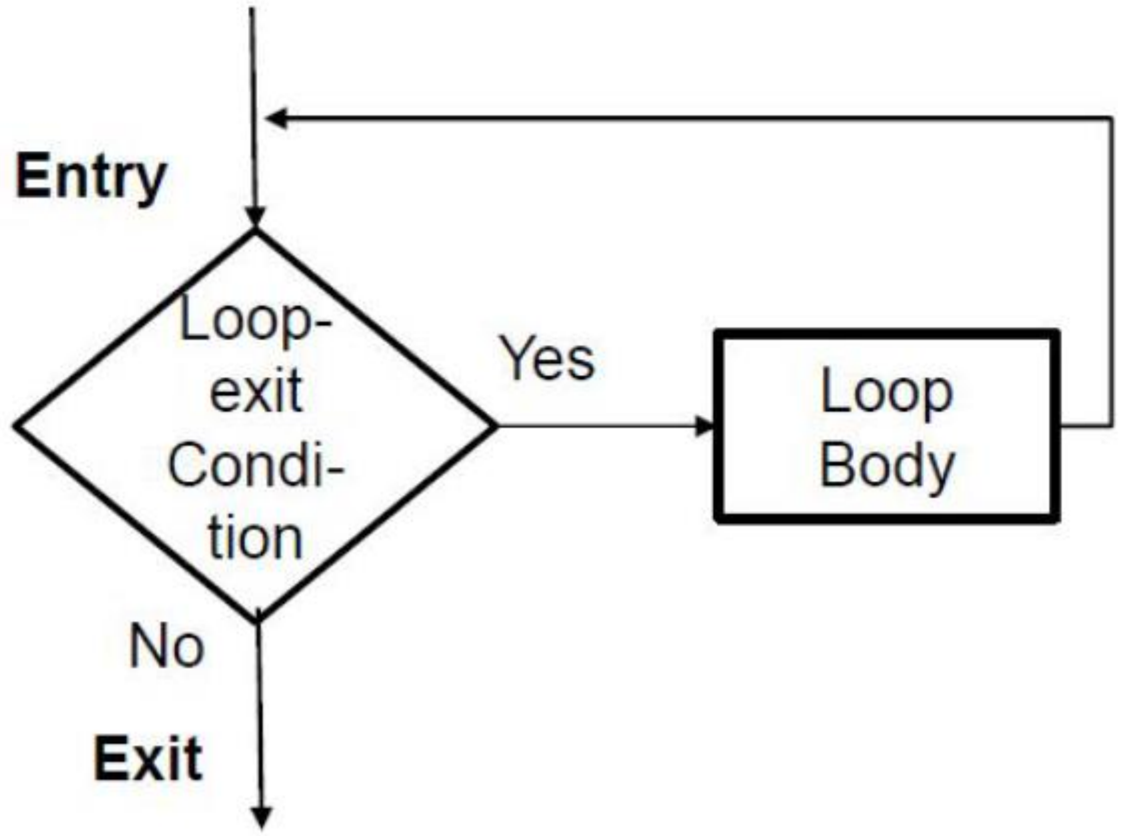
DAY 4

Contents

iteration

Nature of Iteration /1

- Iteration is the most useful and powerful control structure in programming.
- Allows the *repetition of instructions* or *statements* in the loop body.
- Types:
  - While-loop
  - For-loop



Nature of Iteration /2

- While-loops
  - Dependent on a *sentinel value* (or indicator)
- For-loops
  - Generally used for traversing and manipulating sequences.
  - Best used if the number of times that the loop will be executed is known.

Nature of Iteration /3

- Common loop applications
  - Using a loop to accumulate totals
    - Best loop for this application: for, or while loop
  - Using a loop to validate user entry
    - Best loop for this application: while loop

While-loop in Python /1

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

```
while condition:
      statement1
      statement2
```

- Note: Do-While structure doesn't exist in Python.

While-loop in Python /2

- Example:

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

For-loop in Python

- Python's for-loop can be used on any type of *iterable* sequences (such as a string, and other sequences to be introduced later on).
- Format:

```
for element in sequence:
    statement1
    statement2
```

- Note: The for-loop behaves like other PL's *foreach*.
- The `in` keyword checks whether a value is within a sequence or not.

Loops on strings /1

- The following is a script which counts the number of vowels within a string, using while loop.

```
word = input()
vowCount, index = 0, 0
while index < len(word):
    if word[index] in "aeiouAEIOU":
        vowCount += 1
    index += 1
print(f"Vowel Count: {vowCount}")
```

Loops on strings /2

- The following is a script which counts the number of vowels within a string, using for loop.

```
word = input()
vowCount = 0
for letter in word:
    if letter in "aeiouAEIOU":
        vowCount += 1
print(f"Vowel Count: {vowCount}")
```

More on the in Statement

- The in keyword is part of the for loop statement. But other than the for-loop it can also be used in conditionals as well.

```
if answer == 'Y' or answer == 'y':
    print('yes!')
else:
    print('no!')
```

```
if answer in 'Yy':
    print('yes!')
else:
    print('no!')
```

Range

- The range() function allows an iteration over a sequence of integers.
- Formats:

```
range(stop)
range(start, stop)
range(start, stop, step)
```

Range | one argument

```
for x in range(10):
    print(x)
# prints seamlessly from 0 to 9
```

Range | two arguments

```
for x in range(3, 10):
    print(x)
# prints seamlessly from 3, and ends
# at 9
```

Range | three arguments

```
for x in range(3, 10, 2):
    print(x)
# prints seamlessly from 3, and ends
# at 9, and increments by 2 per
# iteration.
```

Break statements

- The `break` statement forces 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 the next statement following the loop.

Continue and Pass statements

- The `continue` statement causes control to be transferred directly to the conditional expression that follows the loop.
- The `pass` statement is a null operation. Nothing happens when it executes. This statement is also useful in parts of your code – that are yet to be determined or conceptualized.

list

Nature of Lists /1

- The list is the most versatile data type and sequence available in Python, which can be written as series of comma-separated values (known as *items* or *elements*) between square brackets – [].
- In other languages the pythonic *list* can be called the *array* – however unlike arrays in other languages which are bound to contain *one data type* for all of its items – pythonic *lists* can contain *any* type of data!
- Note: Pythonic Lists are considered to be *mutable*.

Nature of Lists /2

- Creating lists:

```
list_a = []
list_b = list()
list_c = [1,2,3,4]
list_d = ['a','b']
list_e = ['a','b',1,2,3.456]
```

Indexing Lists /1

```
>>> toys = ['car','doll','top']
>>> print(toys[2])
top
>>> print(toys[0])
car
>>> print(toys[1])
doll
```

item	car	doll	top
index	0	1	2

Indexing Lists /2

```
>>> toys = ['car','doll','top']
>>> print(toys[-1])
top
>>> print(toys[-2])
doll
>>> print(toys[-3])
car
```

item	car	doll	top
index	0	1	2
neg. index	-3	-2	-1

List methods

List Method	Description
.append()	Appends (adds) object to a list
.count()	Counts how many times an object occur in a list
.extend()	Appends objects of another list to the current list
.index()	Returns the index number of an object in the list
.insert()	Inserts an object into a list using the index number
.pop()	Removes and returns the last object from the list
.remove()	Removes an object from the list
.reverse()	Reverse objects in place
.sort()	Sorts objects (alphabetical by default)

Changing values of list items

```
>>> toys = ['car','doll','top']
>>> print(toys[2])
top
>>> toys[2] = 'lego'
>>> print(toys[2])
lego
```

Adding objects to a list

```
>>> toys = ['car','doll','top']
>>> print(toys[-1])
top
>>> toys.append('marbles')
>>> print(toys[-1])
marbles
>>> toys.insert(1,'tamiya')
>>> print(toys)
['car','tamiya','doll','top','marbles']
```

Removing objects from a list /1

```
>>> toys = ['car','doll','top']
>>> toys.remove('doll')
>>> print(toys)
['car','top']
```

Removing objects from a list /2

```
>>> eheads = ['ely','buddy','rayms','marcus']
>>> eheads.pop(0)
>>> print(eheads)
['buddy','rayms','marcus']
>>> eheads.pop()
>>> print(eheads)
['buddy','rayms']
```

Arranging list elements /1

```
>>> numbers = [5,1,3]
>>> numbers.sort()
>>> print(numbers)
[1,3,5]
>>> numbers.sort(reverse=True)
>>> print(numbers)
[5,3,1]
```

Arranging list elements /2

```
>>> numbers = [5,1,3]
>>> numbers.reverse()
>>> print(numbers)
[3,1,5]
```

Finding items in lists

```
>>> numbers = [1,2,3,4,5]
>>> num_index = numbers.index(2)
>>> print(num_index)
1
```

Counting items in lists

```
>>> numbers = [1,1,1,1111,11,11,1]
>>> count = numbers.count(1)
>>> print(count)
4
```