BRANCHING AND LOOPING STATEMENTS IN PYTHON

**- Branching**

* Until now, the execution flow of programs was linear. But there are very few codes or algorithms that can be written in sequential way.
* For example, you have to take input number from user as marks. If number is less than 33, print fail and if number is greater than 33, print pass.
* We cannot write such program is sequential manner. For such scenarios, we require ability from programming language to get view of current state of data and take decisions based on that. This is where branching comes in the picture.
* Branching mechanism gives us the ability to execute block statements written inside the code conditionally.
* Before going to the branching and looping statement, take a note that, in python indentation is not just for beautification, it is part of the syntax.
* In a python file, all statements at indentation level 0 are executed sequentially and unconditionally by python interpreter / IDLE.
* If a statement in python file ends with ‘:’ (colon), the next statement below it must start at next indentation level.

For example,

x = 5

y = 10

if(x < y):

print(“x is less than y)

if(y/x == 0):

print(“y is divisible by x”)

print(“done”)

* Branching statements present in python are –
* 1. If
* 2. If-else
* 3. If-elif
  1. • If Statement o
  2. If statement is written as ‘If<condition>:’
  3. Condition is any right-hand side expression which yields a Boolean value.
  4. If the evaluated value is True, block of statement at next indentation level below if will be executed by python interpreter, else they will be skipped.
  5. <if\_condition> statement is header of If-block. And statements indented below it are body of if block.
  6. If-block = Header(if\_condition) + Body(Statements)
  7. if-block is considered as one statement by Python interpreter.

**- Syntax:**

St 1

St 2

St 3

if<condition>: Header

St 4 St

St Body

St

St 5

* 1. • If-else Statement
  2. For a certain conditional execution, if we want to execute some statements if the condition gets false, then we use ‘else:’ statement along with ‘if’.
  3. ‘else’ statement is written at same indentation level as corresponding ‘if’ statement.
  4. Complete if-else block is also considered as one statement by Python interpreter.

**- Syntax:**

St 1

St 2

St 3

if<condition>:

St

St

St

else: St 4

St

St

St

St 5

* 1. • **If-else-if Statement**
  2. Else-if statement is represented by ‘elif’ keyword in Python.
  3. Whole If-elif-else ladder’s header statements should be at same indentation level, and the whole block is considered as one statement by interpreter.
  4. ‘Switch case’ or similar syntax is not provided in Python for replacement of If-elif-else ladder, because Python is interpreted language.

**- Syntax:**

St 1

St 2

if<if\_condition>:

St

St

St

elif<elif-1\_condition>:

St

St

elif<elif-2\_condition>:

St

St

. St 3

.

.

elif<elif-N\_condition>:

St

St

St

else:

St

St

St

St 4

**- Looping**

* We may want to execute a block of code in our program repeatedly depending the application. For this, we use looping constructs provided by the programming language.

For example, if we want to print table of a number we can write,

num = 5

i = 1

while(i <= 10):

print(num \* i)

i = i + 1

print(“done”)

Here, statements in while block are executed repeatedly till the condition is met i. e. the statement are executed in a loop.

* Looping statements can be also termed as, ‘statements which are function of loop variable i.’
* Every branching and looping statements are comprised of header and body.
* Looping statements present in python are –
* 1. While
* 2. For
* 3. While-else
* 4. For-else
* We will also see following statements which are generally used along with branching and looping statements –
* 1. Pass
* 2. Break
* 3. Continue
* • **While Statement**
* ‘while’ is also condition based looping statement similar to ‘for’.
* Every ‘for’ block can be converted into ‘while’ block, but vice versa is not true.

Example,

L = [10, 20, 30]

i = len(L) for x in L:

x = 0 print(x)

while(x < i):

print(L[x])

x+1

**- Syntax:**

St 1

St 2

St 3

while<condition>:

while\_St 1

while\_st 2

. St 4

.

.

while\_st\_N

St 5

* 1. • For Statement
  2. ‘for’ is looping statement, which executes its body repeatedly depending on loop variable condition.
  3. ‘for’ expression can be written using built-in iterable data-types, as well as ‘Range’ class.

Examples,

i) cnt = 0

L = [ 10, 20, 30, 40, 50]

for x in L:

L[cnt] = x + 5

Cnt + 1

print(L)

* 1. ii) for i in range(len(L)): o ‘for’ expression at indent Ik is header of for-block, and statements below it at indent Ik+1 are body of ‘for-block’.

L[i] = L[i] + 5

print(L)

**- Syntax:**

St 1

St 2

for <v> in <iterable>: Header

for\_St 1

for\_St 2

St 4 . Body

.

.

for\_St N

St 5

• **Pass Statement**

* ‘Pass’ statement is required to keep body of any branching/looping block as empty.
* Pass statement is used just to satisfy the syntactic requirement of python interpreter, because we cannot use ‘;’ (semi-colon), like we use it in C.
* Syntactic significance of body of a header is given by indentation. Hence, the pass statement is used to signify the empty body.
* It acts as just a place-holder. While a comment is completely ignored by python interpreter, pass statement is not ignored. But executing a pass statement does nothing.

**- Syntax:**

St 1

St 2

St 3

for <v> in <iterable>:

if<condition>:

pass

St 4 for\_St 2

.

.

for\_St N

St 5

* • **Break Statement**
* ‘break’ statement can be used inside loop body or condition to break out of the loop.
* After a break statement is encountered, control flow is transferred to next statement after the loop.

- Syntax:

St 1

St 2

St 3

for <v> in <iterable>:

if<condition>:

break

St 4 for\_St 2

.

.

for\_St N

St 5

• **Continue Statement**

* **‘**continue’ is also same as ‘break’ statement, but it doesn’t stop iterating process.
* When ‘continue’ statement is encountered, loop or condition block’s execution is skipped for that iteration.

**- Syntax:**

St 1

St 2

St 3

for <v> in <iterable>:

if<condition>:

continue

St 4 .

.

for\_St N

St 5

• **for-else & while-else Statements**

* These two statements are used to check whether the execution breaks prematurely out of the loop.
* Both statements can be used only when ‘break’ statement is used in loop.

‘for-else’ or ‘while-else’ statements can be easily replaced by using a flag variable with for / while condition.

**- Syntax:**

St 1

St 2

for / while loop condition:

if<condition>:

break

St 3 loop\_St 1

loop\_St 2

.

loop\_St N

else:

St

St 4