CONSTRUCTS - A

1. Decision Control.

In simple terms, the Python **if** statement selects actions to perform.

- Decision-making is the anticipation of **conditions** occurring during the execution of a program and specified actions taken according to the conditions.
- Decision structures **evaluate multiple expressions**, which produce TRUE or FALSE as the outcome. You need to determine which action to take and which statements to execute if the outcome is TRUE or FALSE otherwise.

Following is the general form of a typical decision making structure found in most of the programming languages-

Python programming language assumes:

- Any **non-zero** and **non-null** values as **TRUE**
- Either **zero** or **null**, then it is assumed as **FALSE** value.

Python programming language provides following types of decision making statements.

THE ONE-WAY IF STATEMENT

if statements

if statement consists of a Boolean expression followed by one or more statements.

THE TWO-WAY IF STATEMENT

if...else statements

An **if** statement can be followed by an optional else statement, which executes when the Boolean expression is FALSE.

THE MULTIWAY IF STATEMENT

if...elif...elif...else

The **elif** statement allows you to check multiple expressions for TRUE.

THE NESTED IF STATEMENT

nested if statements

You can use one **if** or **else if** statement inside another **if** or **else if** statement(s).

CONSTRUCTS - A

THE IF STATEMENT

```
>>> if 1:
        print('true')

true
>>>
>>>
```

```
age =int(input("Enter your age :"))
if (age <= 18):
        print("You are not eligible for voting, try in next election!")
        print("Program ends")
```

The output of the above program is as shown below:

```
Enter your age :18
You are not eligible for voting, try in next election!
Program ends

Enter your age :41
Program ends
```

- When we enter the input value up to 18, then the **if** condition checks if age <= 18 i.e.,
- the **expression** is evaluated and return TRUE value so the next statement of the **if** block runs and print the results.
- Similarly when we enter a value higher than the range i.e., greater then 18, then the **expression** evaluates and return FALSE so the Python interpreter do not execute the **if** block at all and so come out of it.
- The next statement which is not a part of the **if** block will execute.

Note: In Python the block of code will be treated as one block as per the indentation, after the conditional statement followed by a colon as shown in the above examples.

Example : *A Supermarket offers you, a discount of 10% if the quantity purchased is more than 2000. Write a program to calculate the total expenses and enter the quantity and price per item as input through the keyboard.*

```
quantity = int(input("Enter quantity :"))
priceperitem = float(input("Enter rate :"))
discount=0.0
if(quantity >2000):
        discount = 0.10
total=(quantity * priceperitem) - (quantity * priceperitem * discount)
print("Discount =%.2f" %discount)
print("Total expenses = %.2f" %total)
```

The output is shown below:

```
Enter quantity :1000
Enter rate :10.5
Discount =0.00
Total expenses = 10500.00

Enter quantity :2500
Enter rate :10.5
Discount =0.10
Total expenses = 23625.00
```

SINGLE STATEMENT SUITES

If the suite of an **if** clause consists only of a single line, it ***may go on the same line as the header statement***. Here is an example of a one-line **if** clause-

```
var = 100
if ( var == 100 ) :
        print ("Value of expression is 100")
print ("Good bye!")
```

When the above code is executed, it produces the following result-

```
Value of expression is 100
Good bye!
```

MULTIPLE STATEMENTS WITHIN IF

It is common in many programs to have more than one statement to be executed if the expression following **if** is satisfied. If such multiple statements are to be executed then they must be properly indented, as illustrated in the following example.

Example: In the below program the current year and the year in which the employee joined the organization are entered through the keyboard using **input()** function. If the number of years for which the employee has served the organization is greater than 2 then a bonus of USD 1500/- is given to the employee. If the years of service are not greater than 2, then the program should do nothing.

```
bonus = 0.0
currentyear = int(input("Enter current year :"))
yearofjoining = int(input("Enter year of joining :"))
yearofservice = currentyear – yearofjoining
if(yearofservice >2):
        bonus = 1500
        print("Bonus =%d" %bonus)
        print("Congratulation! We are happy to provide you a bonus of %d" %bonus)
        print("We appreciate your long term association with us!")

print("\nThis statement is not part of the above if condition because it is not indented as a part of the if statement")
```

The output of the above program is as follows:

```
Enter current year :2015
Enter year of joining :2000
Bonus =1500
Congratulation! We are happy to provide you a bonus of 1500
We appreciate your long term association with us!
This statement is not part of the above if condition because it is not indented as a part of the if statement

Enter current year :2015
Enter year of joining :2014
This statement is not part of the above if condition because it is not indented as a part of the if statement
```

THE IF-ELSE STATEMENT

We can execute one group of statements if the expression evaluates to **true** and another group of statements if the expression evaluates to **false**. This could be done with the help of **else** statement.

Example : In a xyz company, an employee is paid his remuneration under these condition: If his basic salary is less than USD. 2000, then HRA = 10% of basic salary and DA = 90% of basic salary. If his salary is either equal to or above USD. 2000, then HRA = 5% and DA = 80% of basic salary. We will write a program to find his gross salary.

```
basicsalary = int(input("Enter Basic Salary :"))
if(basicsalary < 2000):
        hra = basicsalary * 10/100;da = basicsalary * 90/100
else:
        hra = basicsalary * 5/100;da = basicsalary * 80/100
grosssalary = basicsalary + hra + da
print("Salary Slip :")
print("-" *50)
print("Your Basic Salary is %d" %basicsalary)
print("Your hra is calculated as %.2f" %hra)
print("Your da is calculated as %.2f" %da)
print("-" *50)
print("Gross salary is %.2f" %grosssalary)
```

The output of the above program is as follows:

```
Enter Basic Salary :1800
Salary Slip :
--------------------------------------------------
Your Basic Salary is 1800
Your hra is calculated as 180.00
Your da is calculated as 1620.00
--------------------------------------------------
Gross salary is 3600.00
Enter Basic Salary :3000
Salary Slip :
--------------------------------------------------
Your Basic Salary is 3000
Your hra is calculated as 150.00
Your da is calculated as 2400.00
--------------------------------------------------
Gross salary is 5550.00
```

THE ELIF STATEMENT

- The **elif** statement allows you to check multiple expressions for TRUE and execute a block of code as soon as one of the conditions evaluates to TRUE.
- Similar to the **else**, the **elif** statement is optional.
- However, unlike **else**, for which there can be at most one statement, there can be an arbitrary number of **elif** statements following an **if**.

```
if expression1:
        statement(s)
elif expression2:
        statement(s)
elif expression3:
        statement(s)
else:
        statement(s)
```

Example

```
x = int(input("Please enter an integer: "))
if x < 0:
        print('Negative number')
elif x == 0:
        print('Zero')
elif x == 1:
        print('Single')
else:
        print('Bigger number .....')
```

The output is as follows:

```
Please enter an integer: -1
Negative number
Please enter an integer: 0
Zero
Please enter an integer: 1
Single
Single
Please enter an integer: 100
Bigger number .....
```

Core Python **does not provide switch / case statements** as in other languages, but we can use **if..elif...** statements to simulate switch case as follows:

Example :

Pizza choice example: Suppose you have visited a Pizza restaurant and you want to order a pizza of your choice. Select Pizza from the menu.

```
print("Let's enjoy a Pizza, okay let's go inside the Pizzahut!")
print("Waiter, Please select Pizza of your choice from the menu")
pizzachoice = int(input("Please enter your choice of Pizza: "))
if pizzachoice == 1:
        print('I want to enjoy a pizza napoletana')
elif pizzachoice == 2:
        print('I want to enjoy a pizza rustica')
elif pizzachoice == 3:
        print('I want to enjoy a pizza capricciosa')
else:
        print("Sorry I do not want any of the listed Pizza's, please bring a Coca Cola for me")
```

The output is as follows:

```
Let's enjoy a Pizza, okay let's go inside the Pizzahut!
Waiter, Please select Pizza of your choice from the menu
Please enter your choice of Pizza: 1
I want to enjoy a pizza napoletana

Let's enjoy a Pizza, okay let's go inside the Pizzahut!
Waiter, Please select Pizza of your choice from the menu
Please enter your choice of Pizza: 2
I want to enjoy a pizza rustica

Let's enjoy a Pizza, okay let's go inside the Pizzahut!
Waiter, Please select Pizza of your choice from the menu
Please enter your choice of Pizza: 3
I want to enjoy a pizza capricciosa

Let's enjoy a Pizza, okay let's go inside the Pizzahut!
Waiter, Please select Pizza of your choice from the menu
Please enter your choice of Pizza: 10
Sorry I do not want any of the listed Pizza's, please bring a Coca Cola for me
```

CONSTRUCTS - A

THE NESTED IF STATEMENTS

- There may be a situation when you want to check for another condition after a condition resolves to true.
- In such a situation, you can use the nested **if** construct.
- In a nested **if** construct, you can have an **if**...**elif**...**else** construct inside another **if**...**elif**...**else** construct.

```
if expression1:
        statement(s)
        if expression2:
                statement(s)
        elif expression3:
                statement(s)
        else
                statement(s)
elif expression4:
        statement(s)
else:
        statement(s)
```

Example

```
var = 1000
if var < 2000:
        print ("Expression value is less than 2000")
        if var == 1500:
                print ("Which is 1500")
        elif var == 1000:
                print ("Which is 1000")
        elif var == 500:
                print ("Which is 500")
elif  var < 500:
        print ("Expression value is less than 500")
else:
        print ("Could not find true expression")

print "Good bye! Will enjoy Python Programming again!"
```

The output is as follows:

```
Expression value is less than 2000
Which is 1000
Good bye! Will enjoy Python Programming again!
```

- In the above example, the variable var is assigned the value 1000 and
- the first **if** statement checks if the var has less than 2000, the answer is yes,
- the print statement is executed and
- the below **if** condition is false,
- so the **elif** statement executed,
- it's value is 1000 which is equal to the value in var so this is printed.
- After that the rest statement is false so did not executed.
- Finally the print statement which is outside of the **if-elif-else** construct is executed.

2. LOOPS.

A loop statement allows us to execute a statement or group of statements multiple times. The following diagram illustrates a loop statement. Python programming language provides a **while** loop to handle looping requirements.

The WHILE Loop.

- The **while** ... : operation will first check for its test condition (the ... between the while and the :) and if it's ***True*** , it will evaluate the statements in its indented block a first time.
- After it reaches the end of its indented block, which can include other indented blocks, it will once again evaluate its test condition to see whether it is still ***True*** .
- If it is, it will ***repeat*** its actions again;
- However, if it is False , Python ***leaves*** the indented section and continues to evaluate the rest of the program after the **while** ...: section.

The code in a **while** clause will be executed as long as the **while** statement's condition is ***True***. In code, a **while** statement always consists of the following:

- The **while** keyword
- A **condition** (that is, an expression that evaluates to True or False)
- A **colon**
- Starting on the next line, an **indented** block of code (called the while clause)

```
>>> while True:
        print('Type Ctrl-C to stop me!')

>>> x = 'spam'
>>> while x:                       # While x is not empty
        print(x, end=' ')           # In 2.X use print x,
        x = x[1:]

spam pam am m
>>> a=0; b=10
>>> while a < b:          # One way to code counter loops
        print(a, end=' ')
        a += 1                  # Or, a = a + 1

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

You could also use a loop to ensure that the user enters a name, as follows:

```
name = ''
while not name:
        name = input('Please enter your name: ')
        print( 'Hello, %s!' % name)
```

We could use relational or logical operators in the **while** loop for example:

```
while(i<=10)
while(i>=10 and j <=20)
while(j > 10 and (b<15) or c<20))
```

```
x = 1
while x <= 10:
        print( x);
        x += 1
```

CONSTRUCTS - A

Calculation of simple interest. Ask user to input principal, rate of interest, number of years

```
counter = 1
while(counter <= 3):
        principal = int(input("Enter the principal amount :"))
        numberofyears = int(input("Enter the number of years :"))
        rateofinterest = float(input("Enter the rate of interest :"))
        simpleinterest = principal * numberofyears * rateofinterest/100
        print("Simple interest =%.2f" %simpleinterest)
        counter = counter + 1

print("You have calculated simple interest for 3 time!")
```

```
Enter the principal amount :1000
Enter the number of years :5
Enter the rate of interest :10
Simple interest =500.00

Enter the principal amount :1000
Enter the number of years :5
Enter the rate of interest :13.5
Simple interest =675.00

Enter the principal amount :2500
Enter the number of years :2
Enter the rate of interest :10
Simple interest =500.00

You have calculated simple interest for 3 time!
```

CONSTRUCTS - A

THE LOOP CONTROL STATEMENTS

- ***break,***
- ***continue,***
- ***pass,***
- ***Loop else***

Now that we've seen a few Python loops in action, it's time to take a look at two simple statements that have a purpose only when nested inside loops—the **break** and **continue** statements.

While we're looking at oddballs, we will also study the loop **else** clause here because it is intertwined with break, and Python's empty placeholder statement, **pass** (which is not tied to loops per se, but falls into the general category of simple one-word statements).

In Python:

break
Jumps ***out of the*** closest enclosing loop (past the entire loop statement)
continue
Jumps to the ***top of the*** closest enclosing loop (to the loop's header line)
pass
Does ***nothing*** at all: it's an empty statement placeholder
Loop else block
Runs if and only if ***the loop is exited normally*** (i.e., without hitting a break)

- If you use break , it will only take you out of the most recent loop
- If you have a **while ... : loop** that contains a **for ... in ... : loop** indented within it, a break within the **for ... in ... :** will not break out of the **while ... :** .
- Both **while ... :** and **for ... in ... :** loops can have an **else:** statement at the end of the loop, but it will be run only if the loop doesn't end due to a break statement.

The general format of the **while** loop looks like this:

```
while test:
    statements
    if test:
        break          # Exit loop now, skip else if present
    if test:
        continue       # Go to top of loop now
    statements
    statements
```

else: # Run if we didn't hit a 'break'
    statements

PASS

If you want to code an infinite loop that ***does nothing*** each time through, do it with a pass:

```
>>>while True: pass          # Type Ctrl-C to stop me!
```

Because the body is just an empty statement, Python gets stuck in this loop. **pass** is roughly to statements as **None** is to objects—an explicit nothing.

ELSE

The loop **else** clause is unique to Python. In its most complex form, the **while** statement consists of a header line with a test expression, a body of one or more normally indented statements, and an optional **else** part that is executed *if control exits the loop without a break statement* being encountered. Python keeps evaluating the test at the top and executing the statements nested in the loop body until the test returns a false value:

```
while test:          # Loop test
    statements       # Loop body
else:                # Optional else
    statements       # Run if didn't exit loop with break
```

BREAK

There is a shortcut to getting the program execution to ***break out*** of a **while** loop's clause early. If the execution reaches a **break** statement, it immediately exits the **while** loop's clause. In code, a **break** statement simply contains the **break** keyword.

```
while True:
    print('Please type your name.')
    name = input()
    if name == 'your name':
        break
print('Thank you!')
```

The first line creates an infinite loop; it is a **while** loop whose condition is always **True**. The program execution will always enter the loop and will exit it only when a **break** statement is executed.

CONSTRUCTS - A

CONTINUE

When the program execution reaches a **continue** statement, the program execution immediately ***jumps back to the start*** of the loop and re-evaluates the loop's condition.

```
while True:
        print('Who are you?')
        name = input()
        if name != 'Joe':
                continue
        print('Hello, Joe. What is the password? (It is a fish.)')
        password = input()
        if password == 'swordfish':
                break
print('Access granted.')
```

The following piece of code determines whether a number is **prime**.

```
ctr = 2
num = int(input("Enter a Number :"))
while(ctr <= num-1):
        if(num % ctr == 0):
                print("Number is not prime!")
                break;
        ctr = ctr + 1

print("End of the program")
```

The following piece of code determines whether a positive integer **y** is **prime** by searching for ***factors greater than 1*****:**

```
y = int(input("Enter a Number :"))
x = y // 2                          # y > 1
while x > 1:
        if y % x == 0:              # Remainder
                print(y, 'has factor', x)
                break               # Skip else
        x -= 1
else:                               # Normal exit
        print(y, 'is prime')
```

- Rather than setting a flag to be tested when the loop is exited, it inserts a break where a factor is found. This way, the loop **else** clause can assume that it will be executed only if no factor is found;
- If you don't hit the **break**, the number is prime.
- The loop **else** clause is also run if the body of the loop is never executed, as you don't run a **break** in that event either;
- In a **while** loop, this happens if the test in the header is **false** to begin with.
- Thus, in the preceding example you still get the "is prime" message if x is initially less than or equal to 1 (for instance, if y is 2).