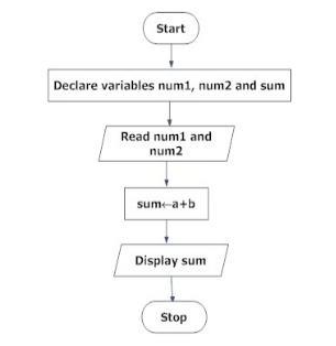
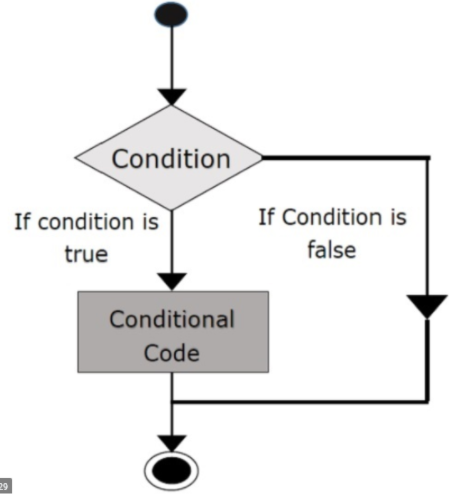
# **Sequence, Selection & Iteration**

When we write lines of code (programming), there are three ways we can control the order these code lines will be executed by the computer:

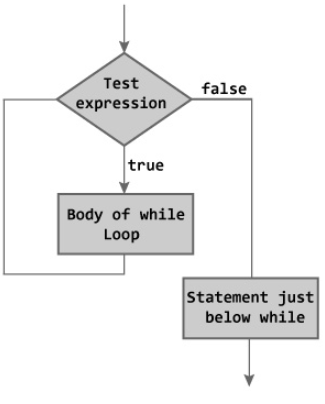
1. **Sequencing (step-by-step)**: This means that the computer will run our code in order (in a sequence), one line after the other from top to bottom of the program instruction. It will start at line 1, then execute line 2 then line 3 and so on till it reaches the last line of the program.



1. **Selection (conditional statements)**: Sometimes you only want some lines of code to be run (executed) only if a condition is met, otherwise you want the computer to ignore these lines and jump over them to the next instruction. This is achieved using the **IF** statements. For example, if a condition is met then lines 2, 3, and 4 are executed, otherwise the computer jumps to line 5 without even looking (executing) line 2,3 and 4.



1. **Iteration (looping)**: Sometimes you want the computer to execute the same lines of code several times based on a given instruction set. This is done using a loop. There are two types of loops in Python: **For loops**, and **while loops**. Loops are handy since they enable us not to have to copy the same lines of code over and over again.



In fact, every full fledged programming language has to have the above three programming constructs (1-3) to qualify as a programming language.

Writing lines of code is almost like writing a recipe to help in instructing on how to prepare a dish. Take for example, the below recipe on how to prepare Swahili pilau rice dish.

How to make Swahili pilau rice from scratch? First of all we have to gather the needed ingredients before we even get started cooking.

**Below are what the ingredients would be:**

**Ingredients**

1. 2 cups rice
2. 1/2 kg meat
3. 3 sliced onions
4. Garlic paste
5. Ginger paste
6. Grounded pilau masala
7. Cumin
8. 8 pieces Cardamon
9. 3 sticks Cinnamon
10. Oil
11. Salt
12. 4-5 cubed potatoes

**Steps involved in cooking the Swahili pilau rice dish**

**Steps**

1. Boil the meat until it's tender
2. In a sufuria Heat the oil and add the onions. Let it cook until golden brown
3. Add the garlic paste followed by the ginger paste and cook for 2 mins
4. Add the grounded pilau masala and the cumin, cardamon and cinnamon cook for a minute
5. Add the boiled meat and potatoes and cook for 2 mins
6. Add the rice and 4 cups of water, stir and let it boil
7. When the water reduces, cover the sufuria (pan) with a lid and reduce the flame to low for 5-10 mins
8. Your pilau is ready
9. It can now be served with pilipili ya kukaanga, kachumbari and any juice of your liking.
10. Complete and Enjoy

We plan and do the same in programming just like in following a recipe. We follow the **sequence** of instructions to achieve a goal. Sometimes in the same instructions you may find yourself skipping (**conditional selection**) some steps, either they may not be necessary or they may be redundant. But sometimes you may also find yourself repeating (**looping**) some steps over and over to make sure you achieve the desired end result.

**A selection statement will be like the program below.**

1. # list my contacts in sequence and print them on the screen.
2. # Python Program listing in sequence
3. print(“Name: MARTIN LUTHER KING JR”)
4. print(“Email: martin.luther@gmail.com”)
5. print(“Phone: 909-456-0392”)
6. print(“Gender: Male”)
7. print()
8. print(“Address: 457 Forrest Park Lane”)
9. print(“Savannah GA 30987-2132”)
10. print(“U.S.A”)

A program is not a good program until our program can interact with the outside world. In some cases a user input is needed. In other cases an input came in the form of a data set from a table in a Database, or an Excel file, or a JSON file or CSV file.

In Python 3.x programming, we use **input(prompt)** to accept user input. While in Python 2.7, we use **raw\_input(prompt)** to accept user input. As you can see, there are two different functions we use depending on the version of Python you pick to use for programming.

**Below is a typical simple example on how to enter input into a program.**

1 name = input(‘Enter your name’)

2 print(“Habari yako, “ + name)

**Or**

1 print (“Enter your names: “)

2 name = input()

3 print(‘Habari yako, ‘ + name)

That is good for the user input function for now. Make sure you practice this in several iterations and examples to get familiar with it.

**Python Conditional Programing**

**The Python IF Statements**

If statements in Python are control flow statements which help us to run a particular code only when a certain condition is satisfied.

For example, we may want to print a receipt at the cash register printer only when the credit card transaction is successfully transmitted to the bank, when a condition is true then we can use an if statement to accomplish this in programming. Or you want to allow a user to access your website content if they are above 18 years of age. This is a conditional argument we have to use to allow that to happen

There are other control flow statements available in Python such as **if..else, if..elif..else**, nested **if** etc.

The syntax of **if statements** in Python is pretty simple and straightforward.

**if condition:**

**block\_of\_code**

## **Python – If statement Example**

flag = True

if flag==True:

print("Welcome")

print("To The")

print("Largest Python Class in Sub-Saharan Africa")

**Program Output:**

Welcome

To The

Largest Python Class in Sub-Saharan Africa

In the above example code we are checking the value of the flag variable. If the value is **True** then we execute the subsequent print statements. The important point to note here is that even if we do not compare the value of the flag with the ‘**True**’ and simply put ‘**flag**’ in place of the condition, the code would run just fine. So the best way to write the above code would be:

flag = True

if flag:

print("Welcome")

print("To The")

print("Largest Python Class in Sub-Saharan Africa")

By looking at the above code, we understand how the if statement works. The out of the condition would either be true or false. If the outcome of condition is true then the statements inside the body of ‘**if**’ executes, however if the outcome of condition is false then the statements inside ‘**if**’ are **skipped**. Let's take a look at another example to understand this:

flag = False

if flag:

print("You Guys")

print("are a")

print("Fantastically great class")

For the above code there would be no output. Because the outcome of the condition is ‘**false**’.

## **Python “if” example without boolean variables**

In the above examples, we have used the boolean variables in place of conditions. However we can use any variables in our conditions. For example:

number = 300

if number < 500:

print("number is less than 500")

Program Output:

num is less than 500

# **Python “If else” Statement Example**

In the last tutorial we learned how to use if statements in Python. In this guide, we will learn another control statement ‘**if..else**’.

We use if statements when we need to execute a certain block of Python code when a particular condition is True. **If..else** statements are like extensions of ‘**if**’ statements, with the help of **if..else** we can execute certain statements if condition **is true and** a different set of statements **if condition is false**.

For example, you want to print ‘an even number’ if the number is even and ‘an odd number’ if the number is not even, we can accomplish this with the help of **if..else** statement.

## **Python – Syntax of “if..else” statement**

**if condition:**

**Code\_block\_1 #True**

**else:**

**Code\_block\_2 #False**

**code\_block\_1: This would execute if the given condition is true**

**code\_block\_2: This would execute if the given condition is false**

## **“if..else” flow control**

## **if-else example in Python**

number = 42

if number % 2 == 0:

print("Even Number") ## True

else:

print("Odd Number") ## False

Program Output:

Even Number

**How about for an odd number?**

**Can you do this homework?**

Find out if the following program is correct both logically and syntactically. What improvements can you make to this program so it can work as needed?

number = 33

if number % 2 != 0:

print("Even Number")

else:

print("Odd Number")

Program Output:

Even Number

In the previous section, we saw the “**if”** statement and “**if..else”** statement. In this section, we will learn about the “**if elif else”** statement in Python. The **if..elif..else** statement is used when we need to check on multiple conditions.

## **Syntax of if elif else statement in Python**

How to check on multiple conditions

if condition:

code\_block\_of\_1

elif condition\_2:

code\_block\_of\_2

elif condition\_3:

code\_block\_of\_3

..

..

..

else:

Code\_block\_of\_n

**Notes:**

1. As you can see, there can be multiple **‘elif’** blocks, however there is only one **‘else’** block allowed.

2. Out of all these blocks, only one code\_block gets executed. If the condition is **True** then the code inside ‘**if**’ gets executed, **if condition** is false then the next condition (associated with **elif**) is evaluated and so on.

If none of the conditions are true then the code inside ‘**else**’ gets executed.

## **Python – if..elif..else statement example**

In this example, we are **checking multiple conditions** using **if..elif..else** statements.

number = 1122

if 9 < number < 99:

print("Two digit number")

elif 99 < number < 999:

print("Three digit number")

elif 999 < number < 9999:

print("Four digit number")

else:

print("number is <= 9 or >= 9999")

**Program output:**

Four digit number

# **Python Nested If else statement**

In the previous tutorial section, we covered the **if statement**, **if..else statement** and **if..elif..else** statement. In this tutorial, we will learn the nesting of these control statements.

When there is an **if statement** (or **if..else** or **if..elif..else**) is present inside another if statement (or **if..else** or **if..elif..else**) then we call this a nesting of control statements.

## **Nested “if..else” statement example**

Below we have an **if statement** inside another **if..else statement** block. Nesting control statements makes us check multiple conditions.

number = -99

if number > 0:

print("Positive Number")

else:

print("Negative Number")

#nested if

if -99 <= number:

print("Two digit Negative Number")

**Program Output:**

**Negative Number**

**Two digit Negative Number**

**Python Iteration (loops)**

**Python Programming and Looping Explained**

Loops in Python programming are used for iterating over a set of statements repeatedly. In Python we have two types of loops:

**i) for**,

**ii) while** and

A syntax for a loop in Python Programming language

for <variable> in <sequence>:

# body\_of\_loop that has set of statements

# which requires repeated execution

Here **<variable>** is a variable that is used for iterating over a **<sequence>** of events. On every iteration, it takes the **next value** from **<sequence>** until the end of sequence is reached.

## **Python – For loop example**

The following example shows the use of a **for loop** to iterate over a list of numbers. In the body of a for loop, we are calculating the square of each number present in the list and displaying the same.

**# Program to print squares of all numbers present in a list**

**# List of integer numbers**

**numbers = [1, 2, 4, 6, 11, 20]**

**# variable to store the square of each num temporary**

**sq = 0**

**# iterating over the given list**

**for val in numbers:**

**# calculating square of each number**

**sq = val \* val**

**# displaying the squares**

**print(sq)**

**Program Output:**

**1**

**4**

**16**

**36**

**121**

**400**

We can use a for loop to iterate over a sequence of numbers (that is either a list, a tuple, a dictionary, a set, or a string).

This is less like the for keyword in other programming languages, and works more like an iterator method as found in other object-orientated programming languages.

With the for loop, we can execute a set of statements, once for each item in a list, tuple, set etc.

Another example of for loop.

fruits = ["apple", "banana", "cherry", “Avocado”, ”Mango”]

for fruit in fruits:

print(fruit)

As you have already noticed,the for loop does not require an indexing variable to set beforehand.

Example on how we loop through a string.

In Python, even strings are literal objects, and they contain a sequence of characters.

**Another example of a for loop**

for fruit in "Pinneaple":

print(fruit)

**Below would be the out of the above program**

**p**

**i**

**n**

**n**

**e**

**a**

**p**

**l**

**e**

## The break Statement in a loop

With the break statement we can stop the loop before it has looped through all the items:

### Example

Exit the loop when fruit is "Avocado":

fruits = ["Lemon", "apple", "banana", "Avocado", "cherry","Papaya"]

for fruit in fruits:

print(fruit)

if fruit == "Avocado":

Break

Exit the loop when x is "banana", but this time the break comes before the print:

fruits = ["Lemon", "apple", "banana", "Avocado", "cherry","Papaya"]

for fruit in fruits:

if fruit == "Avocado":

Break

print(fruit)

## The continue Statement

With the continue statement will be able to stop the current iteration of the loop, and continue with the next:

### 

### 

### 

### Example

Do not print banana:

fruits = ["Lemon", "apple", "banana", "Avocado", "cherry","Papaya"]

for fruit in fruits:

if fruit == "Avocado":

continue

print(fruit)

## The range() Function

To loop through a set of code a specified number of times, we can use the range() function,

The range() function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and ends at a specified number.

### Example

Using the range() function:

for fruit in range(10):

print(fruit)

Note that range(10) is not the values of 0 to 10, but the values 0 to 9.

The range() function defaults to 0 as a starting value, however it is possible to specify the starting value by adding a parameter: range(3, 10), which means values from 3 to 10 (but not including 10):

### Example

Using the start parameter:

for fruit in range(3, 10):

print(fruit)

The range() function defaults to increment the sequence by 1, however it is possible to specify the increment value by adding a third parameter: range(3, 50, **4**):

### Example

Increment the sequence with 4 (default is 1):

for fruit in range(3, 50, 4):

print(fruit)

**Program Output**

**3**

**7**

**11**

**15**

**19**

**23**

**27**

**31**

**35**

**39**

**43**

**47**

## Else in For Loop

The else keyword in a for loop specifies a block of code to be executed when the loop is finished:

### 

### 

### 

### 

### Example

Print all numbers from 0 to 9, and print a message when the loop has ended:

for fruit in range(10):

print(fruit)

else:

print("Finally done!")

**Program output**

**0**

**1**

**2**

**3**

**4**

**5**

**6**

**7**

**8**

**9**

**Finally done!**

## Nested Loops

A nested loop is a loop inside a loop or a loop within another loop.

The "**inner loop**" will be executed one time for each iteration of the "**outer loop**":

### Example

Print each adjective for every fruit:

adjective = ["red", "big", "tasty"]

fruits = ["apple", "banana", "cherry"]

for describe in adjective:

for name in fruits:

print(describe, name)

## The pass Statement

for loops cannot be empty, but if you for some reason have a for loop with no content, put in the pass statement to avoid getting an error.

**Example on how to use pass**

for x in [0, 1, 2, 3, 4]:

pass

**Do this exercise below to practice by filling in what is missing**

fruits = ["apple", "banana", "cherry"]

\_\_ x \_\_ fruits\_\_

print(x)

## The while Loop

With the while loop we can execute a set of statements as long as a condition is true.

**Example code**

Print index as long as i is less than 10:

index = 1

while index < 10:

print(index)

index += 1

**Note:** remember to increment i, or else the loop will continue forever.

The while loop requires relevant variables to be ready, in this example we need to define an indexing variable, index, which we set to 1.

## The break Statement

With the break statement we can stop the loop even if the while condition is true:

### Example

Exit the loop when index is 3:

index = 1

while index < 6:

print(index)

if index == 3:

break

index += 1

**Program out**

**1**

**2**

**3**

## The continue Statement

With the continue statement we can stop the current iteration, and continue with the next:

### Example

Continue to the next iteration if index is 3:

## The continue Statement

With the continue statement we can stop the current iteration, and continue with the next:

### Example

Continue to the next iteration if index is 3:

index = 0

while index < 6:

index += 1

if index == 3:

continue

print(index)

**Program output**

**1**

**2**

**4**

**5**

**6**

# Notice that number 3 is missing in the result

## The else Statement

With the else statement we can run a block of code once when the condition no longer is true:

### Example

Print a message once the condition is false:

index = 1

while index < 10:

print(index)

index += 1

else:

print("index is no longer less than 10")

**Program output**

1

2

3

4

5

i is no longer less than 6

## Exercise:

Print index as long as index is less than 6.

index = 1

\_\_\_\_\_ index < 6\_\_

print(index)

index += 1