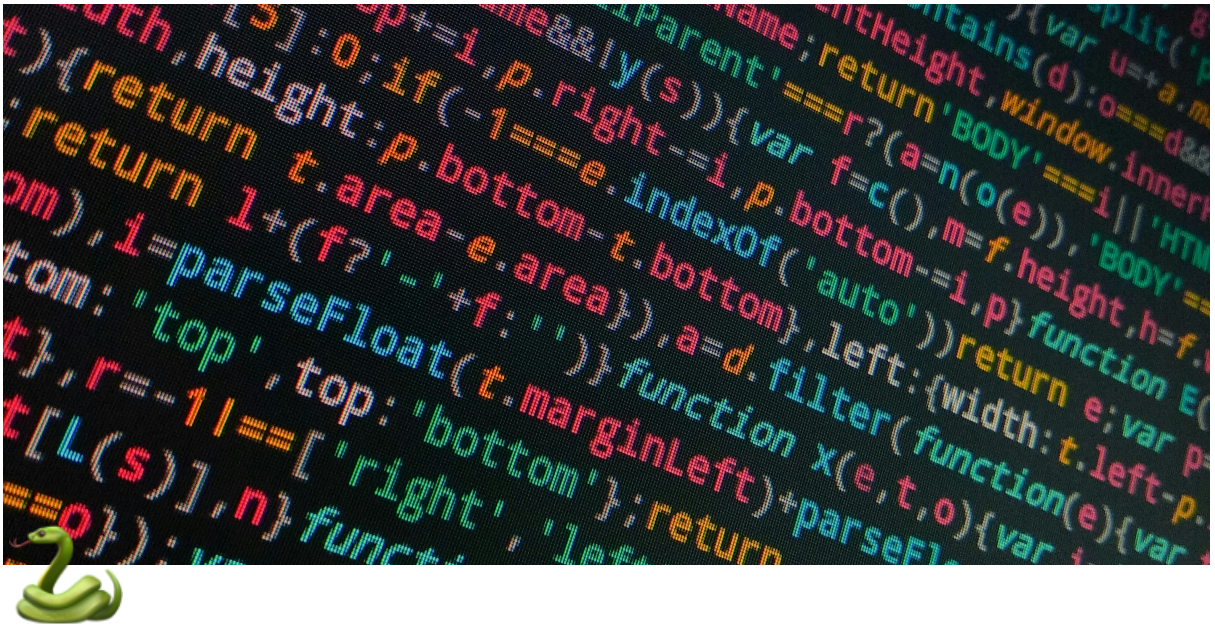


Python Programming

▼ Unit 2: Statements and ControlStructures:

→ Assignment statement, import statement, print statement

1. Assignment Statement

An **assignment statement** is used to assign a value to a variable. The syntax is straightforward:

```
variable_name = value
```

Example:

```
x = 5
name = "Alice"
is_active = True
```

- `x = 5`: Assigns the integer value `5` to the variable `x`.

- `name = "Alice"`: Assigns the string `"Alice"` to the variable `name`.
- `is_active = True`: Assigns the boolean value `True` to the variable `is_active`.

You can also assign the result of an expression to a variable:

```
y = x + 3  # y becomes 8
```

Multiple Assignments

Python allows multiple assignments in a single statement:

```
a, b, c = 1, 2, 3  # a=1, b=2, c=3
```

2. Import Statement

The **import statement** is used to include the functionality from a module or library into your code. Python has a rich standard library, and you can also import custom or third-party modules.

Syntax:

```
import module_name
```

Example:

```
import math  # Imports the math module

print(math.sqrt(16))  # Uses the sqrt function from the
math module, Output: 4.0
```

You can also import specific functions or variables from a module:

```
from math import pi, sqrt  # Imports only pi and sqrt fr
om the math module

print(pi)        # Output: 3.141592653589793
print(sqrt(25))  # Output: 5.0
```

Aliasing Modules:

You can give an alias to a module using the `as` keyword:

```
import numpy as np  # Imports the numpy module with an a
lias np

arr = np.array([1, 2, 3])
print(arr)  # Output: [1 2 3]
```

3. Print Statement

The **print statement** (or more accurately, the `print()` function in Python 3) is used to output text or other data to the console.

Syntax:

```
print(*objects, sep=' ', end='\n', file=sys.stdout, flus
h=False)
```

- `objects`: Any number of objects to be printed. They will be converted to strings.
- `sep`: Separator between objects, default is a space.
- `end`: What to print at the end, default is a newline.
- `file`: Where to print, default is `sys.stdout`.
- `flush`: Whether to forcibly flush the stream.

Example:

```
print("Hello, World!")  # Output: Hello, World!
```

Printing multiple items:

```
print("The answer is", 42)  # Output: The answer is 42
```

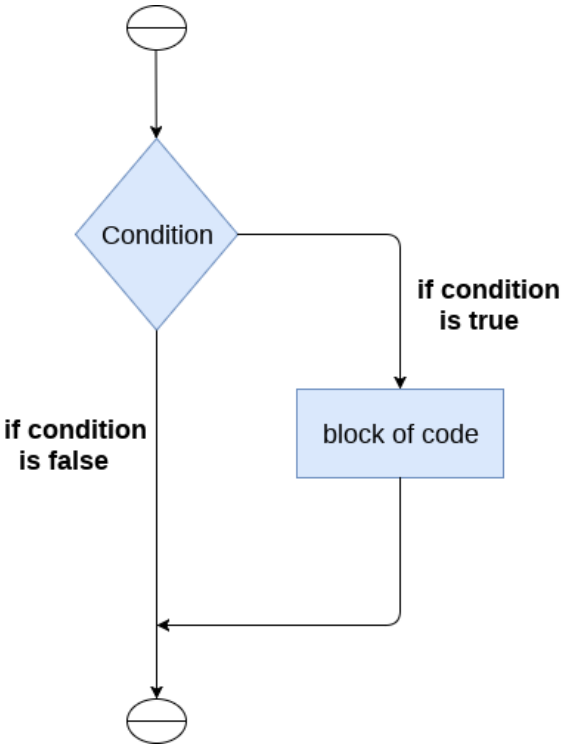
Changing the separator and end character:

```
print("apple", "banana", "cherry", sep=", ", end="!")
# Output: apple, banana, cherry!
```

→ if: elif: else: statement, for: statement., while: statement., continue and break statements, try: except statement

The if statement

The if statement is used to test a particular condition and if the condition is true, it executes a block of code known as if-block. The condition of if statement can be any valid logical expression which can be either evaluated to true or false.



The syntax of the if-statement is given below.

```
if expression:
    statement
```

Example

```
# Simple Python program to understand the if statement
num = int(input("enter the number:"))
# Here, we are taking an integer num and taking input dynam
if num%2 == 0:
# Here, we are checking the condition. If the condition is
    print("The Given number is an even number")
```

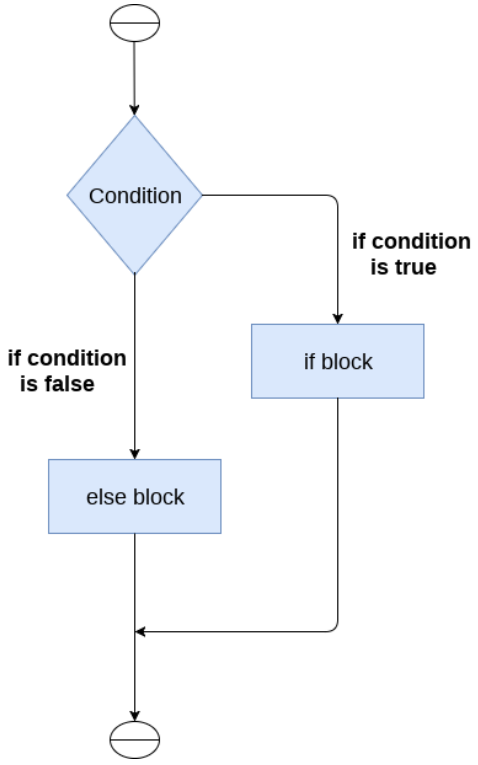
Output:

```
enter the number: 10
The Given number is an even number
```

The if-else statement

The if-else statement provides an else block combined with the if statement which is executed in the false case of the condition.

If the condition is true, then the if-block is executed. Otherwise, the else-block is executed.



The syntax of the if-else statement is given below.

```
if condition:
    #block of statements
else:
    #another block of statements (else-block)
```

Example 1 : Program to check whether a person is eligible to vote or not.

```
# Simple Python Program to check whether a person is eligib
age = int (input("Enter your age: "))
# Here, we are taking an integer num and taking input dynam
if age>=18:
# Here, we are checking the condition. If the condition is
    print("You are eligible to vote !!");
else:
    print("Sorry! you have to wait !!");
```

Output:

```
Enter your age: 90
You are eligible to vote !!
```

The elif statement

The elif statement enables us to check multiple conditions and execute the specific block of statements depending upon the true condition among them. We can have any number of elif statements in our program depending upon our need. However, using elif is optional.

The elif statement works like an if-else-if ladder statement in C. It must be succeeded by an if statement.

The syntax of the elif statement is given below.

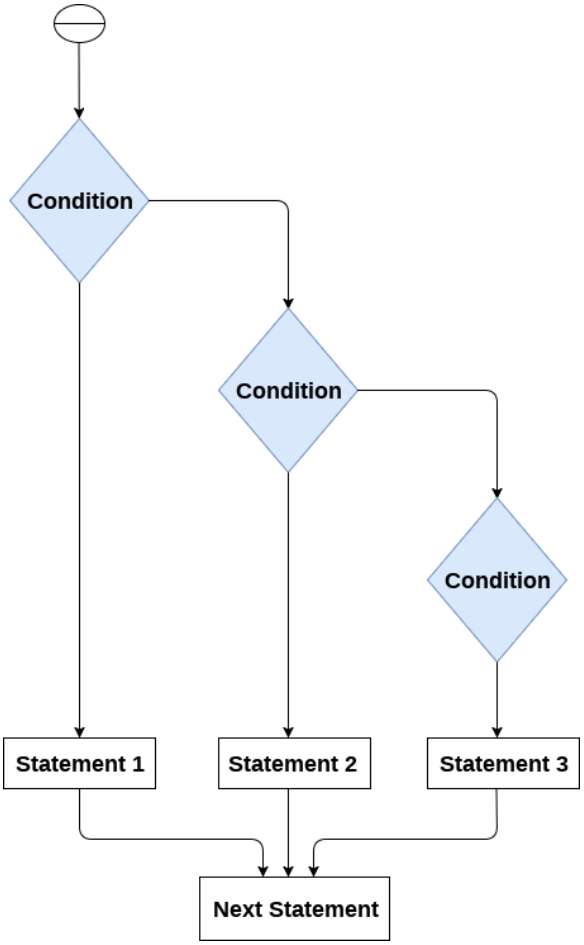
```
if expression 1:
    # block of statements

elif expression 2:
    # block of statements

elif expression 3:
```

```
    # block of statements

else:
    # block of statements
```



Example 1

```
# Simple Python program to understand elif statement
number = int(input("Enter the number?"))
# Here, we are taking an integer number and taking input dy
if number==10:
# Here, we are checking the condition. If the condition is
    print("The given number is equals to 10")
elif number==50:
# Here, we are checking the condition. If the condition is
    print("The given number is equal to 50");
elif number==100:
```

```
# Here, we are checking the condition. If the condition is
    print("The given number is equal to 100");
else:
    print("The given number is not equal to 10, 50 or 100")
```

Output:

```
Enter the number?15
The given number is not equal to 10, 50 or 100
```

Introduction to for Loop in Python

Python frequently uses the Loop to iterate over iterable objects like lists, tuples, and strings. Crossing is the most common way of emphasizing across a series, for loops are used when a section of code needs to be repeated a certain number of times. The for-circle is typically utilized on an iterable item, for example, a rundown or the in-fabricated range capability. In Python, the for Statement runs the code block each time it traverses a series of elements. On the other hand, the "while" Loop is used when a condition needs to be verified after each repetition or when a piece of code needs to be repeated indefinitely. The for Statement is opposed to this Loop.

Syntax of for Loop

```
for value in sequence:
    {loop body}
```

The value is the parameter that determines the element's value within the iterable sequence on each iteration. When a sequence contains expression statements, they are processed first. The first element in the sequence is then assigned to the iterating variable iterating_variable. From that point onward, the planned block is run. Each element in the sequence is assigned to iterating_variable during the statement block until the sequence as a whole is completed. Using indentation, the contents of the Loop are distinguished from the remainder of the program.

Example of Python for Loop

```
# Code to find the sum of squares of each element of the li
```

```
# creating the list of numbers
numbers = [3, 5, 23, 6, 5, 1, 2, 9, 8]

# initializing a variable that will store the sum
sum_ = 0

# using for loop to iterate over the list
for num in numbers:

sum_ = sum_ + num ** 2

print("The sum of squares is: ", sum_)
```

Output:

```
The sum of squares is:  774
```

The range() Function

Since the "range" capability shows up so habitually in for circles, we could erroneously accept the reach as a part of the punctuation of for circle. It's not: It is a built-in Python method that fulfills the requirement of providing a series for the for expression to run over by following a particular pattern (typically serial integers). Mainly, they can act straight on sequences, so counting is unnecessary. This is a typical novice construct if they originate from a language with distinct loop syntax:

```
my_list = [3, 5, 6, 8, 4]
for iter_var in range( len( my_list ) ):
    my_list.append(my_list[iter_var] + 2)
print( my_list )
```

Output:

```
[3, 5, 6, 8, 4, 5, 7, 8, 10, 6]
```

Nested Loops

If we have a piece of content that we need to run various times and, afterward, one more piece of content inside that script that we need to run B

several times, we utilize a "settled circle." While working with an iterable in the rundowns, Python broadly uses these.

Code

```
import random
numbers = [ ]
for val in range(0, 11):
    numbers.append( random.randint( 0, 11 ) )
for num in range( 0, 11 ):
    for i in numbers:
        if num == i:
            print( num, end = " " )
```

Output:

```
0 2 4 5 6 7 8 8 9 10
```

Introduction of Python While Loop

The Python while loop iteration of a code block is executed as long as the given Condition, i.e., conditional_expression, is true.

If we don't know how many times we'll execute the iteration ahead of time, we can write an indefinite loop.

Syntax of Python While Loop

```
Statement
while Condition:
        Statement
```

The given condition, i.e., conditional_expression, is evaluated initially in the Python while loop. Then, if the conditional expression gives a boolean value True, the while loop statements are executed. The conditional expression is verified again when the complete code block is executed. This procedure repeatedly occurs until the conditional expression returns the boolean value False.

ADVERTISEMENT

- The statements of the Python while loop are dictated by indentation.

- The code block begins when a statement is indented & ends with the very first unindented statement.
- Any non-zero number in Python is interpreted as boolean True. False is interpreted as None and 0.

Example

Now we give some examples of while Loop in Python. The examples are given in below -

Program code :

Now we give code examples of while loops in Python for printing numbers from 1 to 10. The code is given below -

```
i=1
while i<=10:
    print(i, end=' ')
    i+=1
```

Output:

Now we compile the above code in python, and after successful compilation, we run it. Then the output is given below -

```
1 2 3 4 5 6 7 8 9 10
```

Python break statement

The break is a keyword in python which is used to bring the program control out of the loop. The break statement breaks the loops one by one, i.e., in the case of nested loops, it breaks the inner loop first and then proceeds to outer loops. In other words, we can say that break is used to abort the current execution of the program and the control goes to the next line after the loop.

The break is commonly used in the cases where we need to break the loop for a given condition. The syntax of the break statement in Python is given below.

Syntax:

```
#loop statements
break;
```

Example : break statement with for loop

Code

```
# break statement example
my_list = [1, 2, 3, 4]
count = 1
for item in my_list:
    if item == 4:
        print("Item matched")
        count += 1
        break
print("Found at location", count)
```

Output:

```
Item matched
Found at location 2
```

In the above example, a list is iterated using a for loop. When the item is matched with value 4, the break statement is executed, and the loop terminates. Then the count is printed by locating the item.

Python continue Statement

Python continue keyword is used to skip the remaining statements of the current loop and go to the next iteration. In Python, loops repeat processes on their own in an efficient way. However, there might be occasions when we wish to leave the current loop entirely, skip iteration, or dismiss the condition controlling the loop.

We use Loop control statements in such cases. The continue keyword is a loop control statement that allows us to change the loop's control. Both Python while and Python for loops can leverage the continue statements.

Syntax:

```
continue
```

Python Continue Statements in for Loop

Printing numbers from 10 to 20 except 15 can be done using continue statement and for loop. The following code is an example of the above scenario:

Code

```
# Python code to show example of continue statement

# looping from 10 to 20
for iterator in range(10, 21):

    # If iterator is equals to 15, loop will continue to th
    if iterator == 15:
        continue
    # otherwise printing the value of iterator
    print( iterator )
```

Output:

```
10
11
12
13
14
16
17
18
19
20
```

Explanation: We will execute a loop from 10 to 20 and test the condition that the iterator is equal to 15. If it equals 15, we'll employ the continue statement to skip to the following iteration displaying any output; otherwise, the loop will print the result.

Python Exceptions

When a Python program meets an error, it stops the execution of the rest of the program. An error in Python might be either an error in the syntax of an

expression or a Python exception. We will see what an exception is. Also, we will see the difference between a syntax error and an exception in this tutorial. Following that, we will learn about trying and except blocks and how to raise exceptions and make assertions. After that, we will see the Python exceptions list.

What is an Exception?

An exception in Python is an incident that happens while executing a program that causes the regular course of the program's commands to be disrupted. When a Python code comes across a condition it can't handle, it raises an exception. An object in Python that describes an error is called an exception.

When a Python code throws an exception, it has two options: handle the exception immediately or stop and quit.

Try and Except Statement - Catching Exceptions

In Python, we catch exceptions and handle them using try and except code blocks. The try clause contains the code that can raise an exception, while the except clause contains the code lines that handle the exception. Let's see if we can access the index from the array, which is more than the array's length, and handle the resulting exception.

Code

```
# Python code to catch an exception and handle it using try

a = ["Python", "Exceptions", "try and except"]
try:
    #looping through the elements of the array a,
    #choosing a range that goes beyond the length of the ar
     for i in range( 4 ):
        print( "The index and element from the array is", i
#if an error occurs in the try block,
#then except block will be executed by the Python interpret
```

```
except:
    print ("Index out of range")
```

Output:

```
The index and element from the array is 0 Python
The index and element from the array is 1 Exceptions
The index and element from the array is 2 try and except
Index out of range
```

The code blocks that potentially produce an error are inserted inside the try clause in the preceding example. The value of i greater than 2 attempts to access the list's item beyond its length, which is not present, resulting in an exception. The except clause then catches this exception and executes code without stopping it.

→ raise statement., with statement, del, case statement

- `raise` **Statement**: Used to raise an exception. You can raise a built-in exception or a custom one.
  - Example: `raise ValueError("Invalid input")`
- `with` **Statement**: Simplifies exception handling by encapsulating common preparation and cleanup tasks in context managers.
  - Example:

    ```
    pythonCopy code
    with open('file.txt', 'r') as file:
        content = file.read()
    ```

  - The file is automatically closed after the block inside the `with` statement is executed.
- `del` **Statement**: Used to delete objects, such as variables, list items, or entire lists.
  - Example:

    ```
    pythonCopy code
    del my_list[0]  # Deletes the first item in the li
    ```

```
st
del my_variable  # Deletes the variable
```