CODERZCOLUMN

PYTHON PROGRAMMING FOR BEGINNERS

CoderzColumn

Wonderful Code Snippets - Theory For Python Programming Beginners

An easy-to-follow guide for python beginners on how to learn programming.

LATEST EDITION

Python Programming Guide For Beginners - Part 1

What Is Python?

Python is an object-oriented programming language.

- It was created by Guido Rossum in 1989.
- The language is designed for the rapid prototyping of the complex real life applications.
- Python has several interfaces to the OS systems and makes use of libraries.
- Many tech giants such as NASA, Google, Facebook, BitTorrent etc. make use of python for designing sophisticated programs.
- Python language promotes code readability.

Trend Of Python Programming

Python is becoming a popular language because it is widely used in Artificial Intelligence, Machine Learning, Natural Language Generation, Neural Network and various other advanced fields of Data Science.

Fun Fact Of Python Programming

Python is named after the comedy television show Monty Python's Flying Circus. It is not named after the Python snake.

Features Of Python Language

1. **Readable**: Python language is readable.
2. **Language Should Be Easy to Learn**: Learning python is easy as this is an expressive and high level programming language, which means it is easy to understand the language and thus easy to learn.
3. **Cross Platform Programing**: Python is available and can run on various operating systems such as Mac, Windows, Linux, Unix etc. This makes it a cross platform and portable language.
4. **Open Source**: Python is an open source programming language.
5. **Large Standard Library**: Python comes with a large standard library that has some handy codes and functions which we can use while writing code in Python.
6. **Python Supports Exception Handling**:Python language supports exception handling which means coders can write less error prone code and can test various scenarios that can cause an exception later on.
7. **Automatic memory management**: Python supports automatic memory management which means the memory is cleared and freed automatically. You do not have to bother clearing the memory.

Application Of Python Language

1. **Python For Web Development**: Advanced frameworks like Django and Flask are based on Python. With these frameworks you can write server side code and help in the management of databases, write backend programming or mapping urls.
2. **Machine Learning**: One of the reasons of python's popularity i s also due to its application in machine learning.Machine learning is a way to write logic so that a machine can learn and solve a particular problem on its own.
3. **Data Analysis**: Python is used for data analysis and data visualization in the form of charts.
4. **Scripting** – Scripting is writing small programs to automate simple tasks such as sending automated response emails etc. Such types of applications can also be written in Python programming language.
5. **Game development** – You can develop games using Python.
6. **Desktop applications** – You can develop desktop applications in Python using libraries like TKinter or QT.

Learning About Python Basics

How To Print In Python With Example:

Example 1: To print the Welcome to CoderzColumn, use the print () function as follows:

```
In [1]:
print ("Welcome to CoderzColumn")
```

Output:
Welcome to CoderzColumn

What would you do if you wish to print if you want to print the name of five cities, you can write:

```
In [2]:
print("Delhi")
print("Mumbai")
print("Chennai")
print("Kolkata")
print("Ahmedabad")
```

Output:
Delhi
Mumbai
Chennai
Kolkata
Ahmedabad

How to print blank lines

If you want to have multiple blank lines in your code.

```
In [3]:
print (3 * "\n")
```

Output:

Understanding The Basics Of Python Input | Output | Import Statement

Python language makes use of two built in functions to perform the basic input/ outpit tasks.

- print()
- input()

Moreover, you would be learning about the 'import module and its application.

Python's Output use print() function We use the print() function to output data to the standard output device.

Example 1 - Print Function

```
In [4]:
print('This sentence is output to the screen')
```

Output:
This sentence is output to the screen

Example 2 - Print Function

In [5]:

```
a = 5
print('The value of a is', a)
```

Output:
The value of a is 5

Output Formatting

Sometimes we would like to format our output to make it look attractive. This can be done by using the str.format() method.

This method is visible to any string object.

In [6]:

```
x = 5; y = 10
print('The value of x is {} and y is {}'.format(x,y))
```

Output:
The value of x is 5 and y is 10

Python Input

What to do when you want the user to enter the values? The value of variables was defined or hard coded into the source code.

To allow flexibility, we might want to take the input from the user. In Python, we have the input() function to allow this.

The syntax for input() is: input([values])

Note: You must try this on your own and then you'll understand how it lets you enter the values.

In [7]:

```
num = input('Enter a number: ')
```

Output:
Enter a number: 24

List Of Simple Python Programs

Python Program to Add Two Numbers

In [1]:

```
# This program adds two numbers
```

```
num1 = 1.5
num2 = 6.3

# Add two numbers
sum = float(num1) + float(num2)

# Display the sum
print('OUTPUT: The sum of {0} and {1} is {2}'.format(num1, num2, sum))
```

Output:
OUTPUT: The sum of 1.5 and 6.3 is 7.8

Add Two Numbers Provided by The User

In [2]:

```
# Store input numbers
num1 = input('Enter first number: ')
num2 = input('Enter second number: ')

# Add two numbers
sum = float(num1) + float(num2)

# Display the sum
print('The sum of {0} and {1} is {2}'.format(num1, num2, sum))
```


Output:
Enter first number: 23
Enter second number: 34
The sum of 23 and 34 is 57.0

Program To Calculate The Squareroot Of A Positive Number

In [8]:

```
# Python Program to calculate the square root

# Note: change this value for a different result
num = 8

# To take the input from the user
#num = float(input('Enter a number: '))

num_sqrt = num ** 0.5
print('The square root of %0.3f is %0.3f'%(num ,num_sqrt))
```

Output:
The square root of 8.000 is 2.828

Program To Calculate The Squareroot Of Real or Complex numbers

In [9]:

```
# Find square root of real or complex numbers
# Importing the complex math module
import cmath

num = 1+2j

# To take input from the user
#num = eval(input('Enter a number: '))

num_sqrt = cmath.sqrt(num)
print('The square root of {0} is {1:0.3f}+{2:0.3f}j'.format(num
,num_sqrt.real,num_sqrt.imag))
```

Output:
The square root of (1+2j) is 1.272+0.786j

CoderzColumn will bring some more interesting and simple python programs in the next part. Till then **Stay Tuned**

Python Programming Guide For Beginners - Part 2

Learning About Data Types In Python

Every programming language has their own set of Data Types. Therefore, every value in Python has a datatype. Since everything is an object in Python programming, data types are actually classes and variables are instances (objects) of these classes.

- Python Numbers
- Python Lists
- Python Tuples
- Python Strings
- Python Dictionary
- Python Set

Understanding Python Numbers:

There are integers, floating numbers and complex numbers in the category of Python Numbers.These data types are defined as int, float, and complex classes.

You can use the **type()** function to know which class a variable or a value belongs to.

Similarly, the **isinstance()** function is used to check if an object belongs to a particular class.

Important Points For Data Types In Python

Integer : It can be of any length and limited by the memory available.

Floating Point Number: The floating point numbers are accurate up to 15 decimal places.

Complex Number: It is written x + yj, where x is the real part and y is the imaginary part.

Let's see the example to understand the data types in Python.

Program For Data Types

```
In [3]:
a = 5
print(a, "is of type", type(a))

a = 2.0
print(a, "is of type", type(a))

a = 1+2j
print(a, "is complex number?", isinstance(1+2j,complex))
```

Output:
5 is of type <class 'int'>
2.0 is of type <class 'float'>
(1+2j) is a complex number? True

Python List

List is basically an ordered sequence of items. It is one of the most used data type in Python and is very flexible. All the items in a list do not need to be of the same type.

Items separated by commas are enclosed within brackets [ ].

```
In [1]:
a = [1, 2.2, 'python']
```

Program To Understand The Use Of Python List Data Type

```
In [2]:
a = [5,10,15,20,25,30,35,40]

# a[2] = 15
print("a[2] = ", a[2])

# a[0:3] = [5, 10, 15]
print("a[0:3] = ", a[0:3])

# a[5:] = [30, 35, 40]
print("a[5:] = ", a[5:])
```


Output:
a[2] = 15
a[0:3] = [5, 10, 15]
a[5:] = [30, 35, 40]

Python Tuples

Tuple is an ordered sequence of items the same as a list. The only difference is that tuples are immutable. Tuples once created cannot be modified.

Tuples are used to write-protect data and are usually faster than lists as they cannot change dynamically.

It is defined within parentheses () where items are separated by commas.

Program To Understand The Use Of Tuples

```
In [5]:
t = (5,'program', 1+3j)

# t[1] = 'program'
```

```
print("t[1] = ", t[1])

# t[0:3] = (5, 'program', (1+3j))
print("t[0:3] = ", t[0:3])
```

t[1] = program
t[0:3] = (5, 'program', (1+3j))

Python Strings

String is a sequence of Unicode characters.

We can use single quotes or double quotes to represent strings. Multi-line strings can be denoted using triple quotes, ''' or """.

Program To Understand The Use Of Python Strings

```
In [6]:
s = "This is a string"
print(s)
s = '''A multiline
string'''
print(s)
```


Output:
This is a string
A multiline
string

Python Dictionary

Dictionary is an unordered collection of key-value pairs.

Dictionary is generally used when you are dealing with a huge amount of data. Dictionaries are optimized for retrieving data. We must know the key to retrieve the value.

In Python, dictionaries are defined within braces {} with each item being a pair in the form key:value. Key and value can be of any type.

Program To Understand The Application Of Python Dictionary

```
In [8]:
d = {1:'value','key':2}
print(type(d))

print("d[1] = ", d[1]);
```

```
print("d['key'] = ", d['key']);
```

Output:
<class 'dict'>
d[1] = value
d['key'] = 2

Python Set

Set is an unordered collection of unique items. Set is defined by values separated by comma inside braces { }. Items in a set are not ordered.

Program To Understand The Application Of Python Set

```
In [9]:
a = {5,2,3,1,4}

# printing set variable
print("a = ", a)

# data type of variable a
print(type(a))
```


Output:
a = {1, 2, 3, 4, 5}
<class 'set'>

Program To Calculate The Area Of Triangle

The variable a, b and c are three sides of a triangle. Then,

Formula Of Triangle

s = (a+b+c)/2 area = √(s(s-a)*(s-b)*(s-c))

```
In [1]:
# Python Program to find the area of triangle

a = 5
b = 6
c = 7

# Uncomment below to take inputs from the user
# a = float(input('Enter first side: '))
# b = float(input('Enter second side: '))
```

```
# c = float(input('Enter third side: '))

# calculate the semi-perimeter
s = (a + b + c) / 2

# calculate the area
area = (s*(s-a)*(s-b)*(s-c)) ** 0.5
print('The area of the triangle is %0.2f' %area)
```

Output:
The area of the triangle is 14.70

In the above program you can even take the input (the values for 3 sides of the triangle from the user. Taking input from the user is already explained in "Part-1")

A Simple Program To Reverse The Number

```
In [11]:
n=int(input("Enter number: "))
rev=0
while(n>0):
    dig=n%10
    rev=rev*10+dig
    n=n//10
print("Reverse of the number:",rev)
```


Output:
Enter number: 23
Reverse of the number: 32

Python Programming Guide For Beginners - Part 3

Learning About Python Comments

Although comments do not change the outcome of a program, they still play an important role in any programming and not just Python. Comments are the way to improve the readability of a code, by explaining what we have done in code in simple english. In this guide, we will learn about comments in Python and their types.

A comment is text that doesn't affect the outcome of a code, it is just a piece of text to let someone know what you have done in a program or what is being done in a block of code. This is especially helpful when someone else has written a code and you are analysing it for bug fixing or making a change in logic, by reading a comment you can understand the purpose of code much faster then by just going through the actual code.

Types of Comments in Python

There are two types of comments in Python.

1. Single line comment
2. Multiple line comment

Single line comment

In python we use # special character to start the comment. Let's take a few examples to understand the usage.

In [1]:
```
# This is just a comment. Anything written here is ignored by Python
```


Multi-line comment:

To have a multi-line comment in Python, we use triple single quotes at the beginning and at the end of the comment

In [2]:
```
'''
This is a
multi-line
comment
'''
```

Out[2]:
'\nThis is a \multi-line\comment\n'

Simple Program To Understand The Use Of Comments In Python

In [5]:
```
'''
We are writing a simple program here
First print statement.
This is a multiple line comment.
'''
print("Hello Guys")
```

```
# Second print statement
print("How are You all?")

print("Welcome to BeginnersBook") # Third print statement
```

Output
Hello Guys
How are You all?
Welcome to BeginnersBook

Understanding Python Variables

Concept: **Variables** are used to store data, they take memory space based on the type of value we assign to them. Creating variables in Python is simple, you just have to write the variable name on the left side of = and the value on the right side.

Important Points To Understand About Variables - Identifiers

These are the generic rules that you need to follow while creating variables in python.

Variable name is known as an identifier.

1. The name of the variable must always start with either a letter or an underscore (_). For example: _str, str, num, _num are all valid names for the variables.
2. The name of the variable cannot start with a number. For example: 9num is not a valid variable name.
3. The name of the variable cannot have special characters, they can only have alphanumeric characters and underscore (A to Z, a to z, 0-9 or _).
4. Variable name is case sensitive in Python which means num and NUM are two different variables in python.

Simple Program For Understanding Variables - Identifiers In Python

```
In [2]:
num = 100
str = "CoderzColumn"
print(num)
print(str)
```

Output:
100
CoderzColumn

Program For Multiple Assignment - Variables In Python

```
In [4]:
x = y = z = 99
```

```
print(x)
print(y)
print(z)
```

Output:
99
99
99

```
In [5]:
```

```
a, b, c = 5, 6, 7
print(a)
print(b)
print(c)
```

Output:
5
6
7

Understanding Concatenation Operation On The Variables

```
In [7]:
```

```
x = 10
y = 20
print(x + y)

p = "Hello"
q = "World"
print(p + " " + q)
```

Output:
30
Hello World

Python Programming Guide For Beginners - Part 4

Learning The Flow Of Control In Python

1. Python If
2. Python if..else
3. Python if..elif..else
4. Python Nested If
5. Python for loop
6. Python while loop
7. Python pass
8. Python continue
9. Python break

The "if" statements in python are used to show the flow control. It helps the coders to run the code or provide a desired output only when the condition is satisfied. For instance, if you want to display a message on the output screen only when certain code conditions are satisfied. In such cases, you can use the "If" statement to accomplish the programming.

1. Learning The Syntax of "If statement" in Python

The syntax of if statements in Python is pretty simple.
if condition:

    block_of_code

Python Program To Understand – If statement Example


```
In [1]:
flag = True
if flag==True:
   print("Welcome")
   print("To")
   print("CoderzColumn.com")
```

Output:
Welcome
To
CoderzColumn.com

Well, in the above example, we are making use of the variable "flag" and checking its value. While using the Python's 'If' Statement, you are required to check whether the value is "True" or not. The most important point here is to understand is that even when we are comparing the value of the "flag" with "True", the variable is used in the place of the condition. This means if the condition is satisfied the statements below will be printed.

Look at another example to understand the difference.

```
In [3]:
flag = True
if flag:
   print("Welcome")
   print("To")
   print("CoderzColumn.com")
```

Output:
Welcome
To
CoderzColumn.com

```
In [4]:
flag = False
if flag:
   print("You Guys")
   print("are")
   print("Awesome")
```


Now, in the above examples, the statements following the 'If' statement did not get displayed because the condition was not satisfied.

Understanding 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:

```
In [5]:
num = 100
if num < 200:
   print("num is less than 200")
```

Output:
num is less than 200

2. Python If else Statement Example

Why Do We Use 'If-Else' Statements?

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 ‘even number’ if the number is even and ‘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:
  block_of_code_1
else:

      block_of_code_2
```

block_of_code_1: This would execute if the given condition is true

block_of_code_2: This would execute if the given condition is false

If-else example in Python

```
In [6]:
num = 22
if num % 2 == 0:
   print("Even Number")
else:
   print("Odd Number")
```


Output:
Even Number

3. Python 'If elif else' Statement Example

```
if condition:
  block_of_code_1
elif condition_2:
  block_of_code_2
elif condition_3:
  block_of_code_3
..
..
..
else:

      block_of_code_n
```

1. There can be multiple ‘elif’ blocks, however there is only ‘else’ block allowed.
2. Out of all these blocks only one block_of_code gets executed. If the condition is true then the code inside ‘if’ gets executed, if the condition is false then the next

condition(associated with elif) is evaluated and so on. If none of the conditions is true then the code inside ‘else’ gets executed.

```
In [7]:
num = 1122
if 9 < num < 99:
   print("Two digit number")
elif 99 < num < 999:
   print("Three digit number")
elif 999 < num < 9999:
   print("Four digit number")
else:
   print("number is <= 9 or >= 9999")
```

Output:
Four digit number

4. Python 'Nested If else' Statement

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 this is calling the nesting of control statements.


```
In [8]:
num = -99
if num > 0:
   print("Positive Number")
else:
   print("Negative Number")
   #nested if
   if -99<=num:
       print("Two digit Negative Number")
```

Output:
Negative Number
Two digit Negative Number

5. Learning Python 'For Loop'

A loop is used for iterating over a set of statements repeatedly. In Python we have three types of loops for, while and do-while. In this guide, we will learn for loop and the other two loops are covered in the separate tutorials.

Syntax of For loop in Python

```
for <variable> in <sequence>:
  body_of_loop that has set of statements
```

which requires repeated execution

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

```
In [9]:
# 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)
```


Output:
1
4
16
36
121
400

Function range()

In the given example, we have made use of the "For Loop" for iterating the items. However you can even use a **range()** function in for loop to iterate over numbers defined by range().

range(n): generates a set of whole numbers starting from 0 to (n-1). For Instance: range(8) is equivalent to [0, 1, 2, 3, 4, 5, 6, 7]

range(start, stop): generates a set of whole numbers starting from start to stop-1. For Instance: range(5, 9) is equivalent to [5, 6, 7, 8]

range(start, stop, step_size): The default step_size is 1 which is why when we didn't specify the step_size, the numbers generated are having a difference of 1. However by specifying step_size we can generate numbers having the difference of step_size. For instance: range(1, 10, 2) is equivalent to [1, 3, 5, 7, 9]

Understanding range() function Python for loop example using

In [1]:
```
# Program to print the sum of first 5 natural numbers

# variable to store the sum
sum = 0

# iterating over natural numbers using range()
for val in range(1, 6):
   # calculating sum
   sum = sum + val

# displaying sum of first 5 natural numbers
print(sum)
```

Output:
15

For Loop With 'else' Block

In [2]:
```
for val in range(5):
   print(val)
else:
   print("The loop has completed execution")
```


Output:
0
1
2
3
4
The loop has completed execution

Nested For Loop In Python

In [3]:
```
for num1 in range(3):
   for num2 in range(10, 14):
      print(num1, ",", num2)
```

Output:
0 , 10

0 , 11
0 , 12
0 , 13
1 , 10
1 , 11
1 , 12
1 , 13
2 , 10
2 , 11
2 , 12
2 , 13

6. Python "While Loop"

While loop is used to iterate over a block of code repeatedly until a given condition returns false. In the last tutorial, we looked for a loop in Python, which is also used for the same purpose. The main difference is that we use a while loop when we are not certain of the number of times the loop requires execution, on the other hand when we know exactly how many times we need to run the loop, we use it for loop.

Syntax of while loop
while condition:

    body_of_while

Python – While Loop Example

```
In [8]:
num = 1
# loop will repeat itself as long as
# num < 10 remains true
while num < 10:
    print(num)
    #incrementing the value of num
    num = num + 3
```

Output:
1
4
7

Nested While Loop

```
In [9]:
i = 1
j = 5
```

```
while i < 4:
   while j < 8:
       print(i, ",", j)
       j = j + 1
       i = i + 1
```

Output:
1 , 5
2 , 6
3 , 7

7. Python "pass" Statement

The pass statement acts as a placeholder and is usually used when there is no need for code but a statement is still required to make a code syntactically correct. For example we want to declare a function in our code but we want to implement that function in future, which means we are not yet ready to write the body of the function. In this case we cannot leave the body of the function empty as this would raise error because it is syntactically incorrect, in such cases we can use a pass statement which does nothing but makes the code syntactically correct.

Pass statement vs comment You may be wondering that a python comment works similar to the pass statement as it does nothing so we can use comment in place of pass statement. Well, it is not the case, a comment is not a placeholder and it is completely ignored by the Python interpreter while on the other hand pass is not ignored by interpreter, it says the interpreter to do nothing.

Python pass statement example

```
In [5]:
for num in [20, 11, 9, 66, 4, 89, 44]:
   if num%2 == 0:
       pass
   else:
       print(num)
```

Output:
11
9
89

8. Python Continue Statement

The continue statement is used inside a loop to skip the rest of the statements in the body of the loop for the current iteration and jump to the beginning of the loop for the next iteration. The break and continue statements are used to alter the flow of the loop, break terminates the loop when a condition is met and continues to skip the current iteration.

```
In [6]:
# program to display only odd numbers
for num in [20, 11, 9, 66, 4, 89, 44]:
   # Skipping the iteration when number is even
   if num%2 == 0:
       continue
   # This statement will be skipped for all even numbers
   print(num)
```

Output:
11
9
89

9. Python break Statement

The break statement is used to terminate the loop when a certain condition is met. We already learned in previous tutorials (for loop and while loop) that a loop is used to iterate a set of statements repeatedly as long as the loop condition returns true. The break statement is generally used inside a loop along with an if statement so that when a particular condition (defined in an if statement) returns true, the break statement is encountered and the loop terminates.

```
In [1]:
# program to display all the elements before number 88
for num in [11, 9, 88, 10, 90, 3, 19]:
  print(num)
  if(num==88):
     print("The number 88 is found")
     print("Terminating the loop")
     break
```

Output:
11
9
88
The number 88 is found
Terminating the loop

Note:

You would always want to use the break statement with an if statement so that only when the condition associated with ‘if’ is true then only break is encountered. If you do not use it with an ‘if’ statement then the break statement would be encountered in the first iteration of the loop and the loop would always terminate on the first iteration.

Python Programming Guide For Beginners - Part 5

1. Learning About Python Functions
2. What Is The Use Of Function In Python?
3. Syntax of functions in Python
4. What Are The Different Types Of Functions In Python?
5. Default Arguments In Function
6. Python Recursion
7. Why Do We Need To Use Recursion in Programming?
8. What Are The Pros And Cons Of Recursion In Programming?

Learning About Python Functions

This tutorial will help the python beginners to learn functions in python.

Definition:

A function is a block of code that contains one or more Python statements and used for performing specific tasks.

What Is The Use Of Function In Python?

As I mentioned above, a function is a block of code that performs a specific task. Let's discuss what we can achieve in Python by using functions in our code:

1. Code reusability:

Let's say we are writing an application in Python where we need to perform a specific task in several places of our code, assuming that we need to write 10 lines of code to do that specific task. It would be better to write those 10 lines of code in a function and just call the function wherever needed, because writing those 10 lines every time you perform that task is tedious, it would make your code lengthy, less-readable and increase the chances of human errors.

2. Improves Readability:

By using functions for frequent tasks you make your code structured and readable. It would be easier for anyone to look at the code and be able to understand the flow and purpose of the code.

3. Avoid redundancy:

When you no longer repeat the same lines of code throughout the code and use functions in places of those, you actually avoid the redundancy that you may have created by not using functions.

Syntax of functions in Python

Function declaration:

```
def function_name(function_parameters):
    function_body # Set of Python statements

            return # optional return statement
```

Syntax For Calling The Function:

when function doesn't return anything

```
    function_name(parameters)
```

OR

```
# when function returns something
# variable is to store the returned value

    variable = function_name(parameters)
```


Python Function Example

Here we have a **function add()** that adds two numbers passed to it as parameters. Later after function declaration we are calling the function twice in our program to perform the addition.

```
In [1]:
def add(num1, num2):
    return num1 + num2


sum1 = add(100, 200)
sum2 = add(8, 9)
print(sum1)
print(sum2)
```

Output:

```
300
17
```

What Are The Different Types Of Functions In Python?

There are two types of functions in Python:

1. **Built-in functions:** These functions are predefined in Python and we need not to declare these functions before calling them. We can freely invoke them as and when needed.
2. **User defined functions:** The functions which we create in our code are user-defined functions. The add() function that we have created in above examples is a user-defined function.

Default Arguments In Function

We have already learned about the **functions** and its types. Now, it is important to learn how to declare a call a function. Now we will have to see how can we use the **default arguments**.

By using default arguments we can avoid the errors that may arise while **calling a function** without passing all the parameters. Let's take an example to understand this:

In this example we have provided the **default argument** for the second parameter, this default argument would be used when we do not provide the second parameter while calling this function.


```
In [3]:
# default argument for second parameter
def add(num1, num2=1):
    return num1 + num2


sum1 = add(100, 200)
sum2 = add(8)   # used default argument for second param
sum3 = add(100)   # used default argument for second param
print(sum1)
print(sum2)
print(sum3)
```

Output:
300
9
101

Python Recursion

A **function** is said to be a **recursive** if it calls itself. For example, let's say we have a function abc() and in the body of **abc()** there is a call to the **abc()**.

Python Example Of Recursion

In this example we are defining a user-defined function factorial(). This function finds the factorial of a number by calling itself repeatedly until the base case(We will discuss more about the base case later, after this example) is reached.

```
In [5]:
# Example of recursion in Python to
# find the factorial of a given number

def factorial(num):
    """This function calls itself to find
    the factorial of a number"""

    if num == 1:
        return 1
    else:
        return (num * factorial(num - 1))


num = 5
print("Factorial of", num, "is: ", factorial(num))
```

Output:
Factorial of 5 is: 120

Why Do We Need To Use Recursion in Programming?

Explanation: We use recursion to break a big problem into small problems and those small problems into further smaller problems and so on. At the end the solutions of all the smaller subproblems collectively help in finding the solution of the big main problem.

What Are The Pros And Cons Of Recursion In Programming?

Pros of Recursion Recursion makes our program:

1. Easier to write.
2. Readable – Code is easier to read and understand.
3. Reduce the lines of code – It takes less lines of code to solve a problem using recursion.

Cons of Recursion

1. Not all problems can be solved using recursion.
2. If you don't define the base case then the code would run indefinitely.
3. Debugging is difficult in recursive functions as the function is calling itself in a loop and it is hard to understand which call is causing the issue.

4. Memory overhead – Call to the recursive function is not memory efficient.

Python Programming Guide For Beginners – Part 6

As we are learning about Python programming and its basics, now is the time we understand the OOP concept in Python.

Object oriented programming as a discipline has gained a universal following among developers. Python, an in-demand programming language also follows an object-oriented programming paradigm. It deals with declaring Python classes and objects which lays the foundation of OOPs concepts. This article on “object oriented programming python” will walk you through declaring python classes, instantiating objects from them along with the four methodologies of OOPs.

Table Of Content

1. Introduction to Object Oriented Programming in Python
2. Understanding object oriented programming
3. What are Classes and Objects?
4. Object-Oriented Programming methodologies:
5. Inheritance
6. Polymorphism
7. Encapsulation
8. Abstraction

Introduction To OOP Programing In Python

Object Oriented Programming is a way of computer programming using the idea of “objects” to represent data and methods. It is also an approach used for creating neat and reusable code instead of a redundant one. the program is divided into self-contained objects or several mini-programs. Every Individual object represents a different part of the application having its own logic and data to communicate within themselves.

Object-Oriented Programming (OOP)

- It is a bottom-up approach
- Program is divided into objects

- Makes use of Access modifiers public', private', protected'
- It is more secure
- Object can move freely within member functions
- It supports inheritance

What are Classes and Objects?

Definition Of Class: A class is a collection of objects or you can say it is a blueprint of objects defining the common attributes and behavior. Now the question arises, how do you do that?

Well, it logically groups the data in such a way that code reusability becomes easy. I can give you a real-life example- think of an office going 'employee' as a class and all the attributes related to it like 'emp_name', 'emp_age', 'emp_salary', 'emp_id' as the objects in Python. Let us see from the coding perspective how you instantiate a class and an object.

Class is defined under a "Class" Keyword.

Example:

```
class class1(): ## class 1 is the name of the class
```

Objects:

Definition: Objects are an instance of a class. It is an entity that has state and behavior. In a nutshell, it is an instance of a class that can access the data.

Syntax:

```
obj = class1()
```

Here obj is the "object " of class1.

Example - Creating an Object and Class in python:

```
In [3]:
class employee():
    def __init__(self,name,age,id,salary):
        self.name = name
        self.age = age
        self.salary = salary
        self.id = id
emp1 = employee("Dolly",22,1000,1234)
emp2 = employee("Sunny",23,2000,2234)
print(emp1.__dict__)
```

Output:
{'name': 'Dolly', 'age': 22, 'salary': 1234, 'id': 1000}

Learning About Object-Oriented Programming methodologies:

Object-Oriented Programming methodologies deal with the following concepts.

- Inheritance
- Polymorphism
- Encapsulation
- Abstraction

Let us understand the first concept of inheritance in detail.

Inheritance:

Ever heard of this dialogue from relatives “you look exactly like your father/mother” the reason behind this is called ‘inheritance’. From the Programming aspect, It generally means “inheriting or transfer of characteristics from parent to child class without any modification”. The new class is called the derived/child class and the one from which it is derived is called a parent/base class.

Single Inheritance:

Single level inheritance enables a derived class to inherit characteristics from a single parent class.


```
In [16]:
class employee1(): ##Parent Class
    def __init__(self, name, age, salary):
        self.name = name
        self.age = age
        self.salary = salary
class childemployee(employee1):##This is a child class
    def __init__(self, name, age, salary,id):
        self.name = name
        self.age = age
        self.salary = salary
        self.id = id
emp1 = employee1('Dolly',22,1000)
print(emp1.age)
```

Output:
22

Multilevel Inheritance:

Multi-level inheritance enables a derived class to inherit properties from an immediate parent class which in turn inherits properties from his parent class.

Example:

```
In [15]:
class employee():##Super class
    def __init__(self,name,age,salary):
        self.name = name
        self.age = age
        self.salary = salary
class childemployee1(employee):##First child class
    def __init__(self,name,age,salary):
        self.name = name
        self.age = age
        self.salary = salary
class childemployee2(childemployee1):##Second child class
    def __init__(self, name, age, salary):
        self.name = name
        self.age = age
        self.salary = salary
emp1 = employee('Dolly',22,1000)
emp2 = childemployee1('Sunny',23,2000)
print(emp1.age)
print(emp2.age)
```


Output:
22
23

Hierarchical Inheritance:

Hierarchical level inheritance enables more than one derived class to inherit properties from a parent class.

Example:

```
In [14]:
class employee():
    def __init__(self, name, age, salary):      ##Hierarchical Inheritance
        self.name = name
        self.age = age
        self.salary = salary
```

```
class childemployee1(employee):
    def __init__(self,name,age,salary):
        self.name = name
        self.age = age
        self.salary = salary
class childemployee2(employee):
    def __init__(self, name, age, salary):
        self.name = name
        self.age = age
        self.salary = salary
emp1 = employee('Dolly',22,1000)
emp2 = employee('Sunny',23,2000)
print(emp1.age)
print(emp2.age)
```

Output:
22
23

Multiple Inheritance:

Multiple level inheritance enables one derived class to inherit properties from more than one base class.

Example:

```
In [13]:
class employee1():##Parent class
    def __init__(self, name, age, salary):
        self.name = name
        self.age = age
        self.salary = salary
class employee2():##Parent class
    def __init__(self,name,age,salary,id):
         self.name = name
         self.age = age
         self.salary = salary
         self.id = id
class childemployee(employee1,employee2):
    def __init__(self, name, age, salary,id):
        self.name = name
        self.age = age
        self.salary = salary
        self.id = id
emp1 = employee1('Dolly',22,1000)
emp2 = employee2('Sunny',23,2000,1234)
```

```
print(emp1.age)
print(emp2.id)
```

Output:
22
1234

Polymorphism:

You all must have used GPS for navigating the route. Isn't it amazing how many different routes you come across for the same destination depending on the traffic? From a programming point of view this is called 'polymorphism'. It is one such OOP methodology where one task can be performed in several different ways. To put it in simple words, it is a property of an object which allows it to take multiple forms.

Polymorphism is of two types:

- Compile-time Polymorphism
- Run-time Polymorphism
- Compile-time Polymorphism:
- A compile-time polymorphism also called static polymorphism which gets resolved during the compilation time of the program. One common example is "method overloading". Let me show you a quick example of the same.


```
In [23]:
class employee1():
    def name(self):
        print("Dolly Is her name")

    def salary(self):
        print("3000 is her salary")
    def age(self):
        print("22 is her age")
class employee2():
    def name(self):
        print("Sunny is his name")
    def salary(self):
        print("4000 is his salary")
    def age(self):
        print("23 is his age")

def func1(obj):##Method Overloading

        obj.name()
        obj.salary()
        obj.age()
```

```
obj_emp1 = employee1()
obj_emp2 = employee2()
func1(obj_emp1)
func1(obj_emp2)
```

Output:
Dolly Is her name
3000 is her salary
22 is her age
Sunny is his name
4000 is his salary
23 is his age

Run-time Polymorphism:

A **run-time Polymorphism** is also called dynamic polymorphism where it gets resolved into the run time. One common example of Run-time polymorphism is “method overriding”. Let me show you through an example for a better understanding.

Example:


```
In [24]:
class employee():
  def __init__(self,name,age,id,salary):
      self.name = name
      self.age = age
      self.salary = salary
      self.id = id
def earn(self):
       pass
class childemployee1(employee):
  def earn(self):##Run-time polymorphism
     print("no money")
class childemployee2(employee):
  def earn(self):
      print("has money")
c = childemployee1
c.earn(employee)
d = childemployee2
d.earn(employee)
```

Output:
no money
has money

Encapsulation:

In a raw form, encapsulation basically means binding up of data in a single class. Python does not have any private keyword, unlike Java. A class shouldn't be directly accessed but be prefixed in an underscore.

Let me show you an example for a better understanding.

Example:

In [28]:

```
class employee(object):
   def __init__(self):
       self.name = 'Dolly'
       self.age = 22
       self.salary = 1234
object1 = employee()
print(object1.name)
print(object1.age)
print(object1.salary)
```


Output:
Dolly
22
1234

Abstraction:

Suppose you booked a movie ticket from a bookmyshow using net banking or any other process. You don't know the procedure of how the pin is generated or how the verification is done. This is called 'abstraction' from the programming aspect, it basically means you only show the implementation details of a particular process and hide the details from the user. It is used to simplify complex problems by modeling classes appropriate to the problem.

An abstract class cannot be instantiated which simply means you cannot create objects for this type of class. It can only be used for inheriting the functionalities.

Example:

In [30]:

```
from abc import ABC,abstractmethod
class employee(ABC):
   def emp_id(self,id,name,age,salary):     ##Abstraction
```

```
        pass
class childemployee1(employee):
    def emp_id(self,id):
        print("emp_id is 12345")
emp1 = childemployee1()
emp1.emp_id(id)
```

Output:
emp_id is 12345

Python Programming Guide For Beginners - Part 7

Learning Python OOPs Concepts

Table Of Content

1. What Is The Python OOP Concept?
2. How to create 'Class' and 'Objects' in Python
3. Creating Objects of class
4. Example of Class and Objects
5. Python Constructors – default and parameterized
6. Types of constructors in Python
7. What is a constructor?
8. Python – default constructor example

What Is The Python OOP Concept?

Python is an object-oriented programming language. What this means is we can solve a problem in Python by creating objects in our programs. In this guide, we will discuss OOPs terms such as class, objects, methods etc. along with the Object oriented programming features such as inheritance, polymorphism, abstraction, encapsulation.

Understanding The Basic Terms:

Object An object is an entity that has attributes and behaviour. For example, Ram is an object who has attributes such as height, weight, color etc. and has certain behaviours such as walking, talking, eating etc.

Class A class is a blueprint for the objects. For example, Ram, Shyam, Steve, Rick are all objects so we can define a template (blueprint) class Human for these objects. The class can define the common attributes and behaviours of all the objects.

Methods As we discussed above, an object has attributes and behaviours. These behaviours are called methods in programming.

How to create 'Class' and 'Objects' in Python

To define a class in Python you need to use a class that is defined using the keyword class.

Example In this example, we are creating an empty class called DemoClass. This class has no attributes and methods.

The string that we mention in the triple quotes is a docstring which is an optional string that briefly explains the purpose of the class.


```
In [1]:
class DemoClass:
    """This is my docstring, this explains brief about the class"""

# this prints the docstring of the class
print(DemoClass.__doc__)
```

This is my docstring, this explains brief about the class

Creating Objects of class

In this example, we have a class **MyNewClass** that has an attribute num and a function **hello()**. We are creating an object obj of the **class and accessing** the attribute value of the object and calling the method hello() using the object.

```
In [1]:
class MyNewClass:
    """This class demonstrates the creation of objects"""

    # instance attribute
    num = 100

    # instance method
    def hello(self):
        print("Hello World!")
```

In [2]:
```
# creating object of MyNewClass
obj = MyNewClass()

# prints attribute value
print(obj.num)

# calling method hello()
obj.hello()

# prints docstring
print(MyNewClass.__doc__)
```

Output:
100
Hello World!
This class demonstrates the creation of objects

Example of Class and Objects

- Object attributes: name, height, weight
- Object behaviour: eating()

In [3]:
```
class Human:
    # instance attributes
    def __init__(self, name, height, weight):
        self.name = name
        self.height = height
        self.weight = weight

    # instance methods (behaviours)
    def eating(self, food):
        return "{} is eating {}".format(self.name, food)


# creating objects of class Human
ram = Human("Ram", 6, 60)
steve = Human("Steve", 5.9, 56)

# accessing object information
print("Height of {} is {}".format(ram.name, ram.height))
print("Weight of {} is {}".format(ram.name, ram.weight))
print(ram.eating("Pizza"))
```

```
print("Weight of {} is {}".format(steve.name, steve.height))
print("Weight of {} is {}".format(steve.name, steve.weight))
print(steve.eating("Big Kahuna Burger"))
```

Output:
Height of Ram is 6
Weight of Ram is 60
Ram is eating Pizza
Weight of Steve is 5.9
Weight of Steve is 56
Steve is eating Big Kahuna Burger

Python Constructors – default and parameterized

Definition:

A constructor is a special kind of method which is used for initializing the instance variables during object creation. In this guide, we will see what is a constructor, types of it and how to use them in python programming with examples.

Types of constructors in Python

We have two types of constructors in Python.

1. **default constructor** – this is the one, which we have seen in the above example. This constructor doesn't accept any arguments.
2. **a parameterized constructor** – constructor with parameters is known as parameterized constructor.

1. What is a Constructor in Python?

Constructor is used for initializing the instance members when we create the object of a class.

Python – default constructor example

Note: An object cannot be created if we don't have a constructor in our program. This is why when we do not declare a constructor in our program, python does it for us. Let's have a look at the example below.

```
def __init__(self):

        # no body, does nothing.
```

```
In [1]:
class DemoClass:
    # constructor
```

```
    def __init__(self):
        # initializing instance variable
        self.num=100

    # a method
    def read_number(self):
        print(self.num)


# creating object of the class. This invokes constructor
obj = DemoClass()

# calling the instance method using the object obj
obj.read_number()
```

Output:
100

Syntax of constructor declaration

As we have seen in the above example that a constructor always has a name init and the name init is prefixed and suffixed with a double underscore(__).

We declare a constructor using the def keyword, just like methods.

```
def __init__(self):

        # body of the constructor
```


Python – Parameterized constructor example

When we declare a constructor in such a way that it accepts the arguments during object creation then such types of constructors are known as Parameterized constructors. As you can see that with such types of constructors we can pass the values (data) during object creation, which is used by the constructor to initialize the instance members of that object.

```
In [5]:
class DemoClass:
    num = 101

    # parameterized constructor
    def __init__(self, data):
        self.num = data

    # a method
    def read_number(self):
        print(self.num)
```

```
# creating object of the class
# this will invoke parameterized constructor
obj = DemoClass(55)

# calling the instance method using the object obj
obj.read_number()

# creating another object of the class
obj2 = DemoClass(66)

# calling the instance method using the object obj
obj2.read_number()
```

Output:
55
66

Method1 : When we do not declare a constructor


```
In [6]:
class DemoClass:
    num = 101

    # a method
    def read_number(self):
        print(self.num)


# creating object of the class
obj = DemoClass()

# calling the instance method using the object obj
obj.read_number()
```

Output:
101

Method2: When we declare a constructor

```
In [10]:
class DemoClass:
```

```
    num = 101

    # non-parameterized constructor
    def __init__(self):
        self.num = 999

    # a method
    def read_number(self):
        print(self.num)


# creating object of the class
obj = DemoClass()

# calling the instance method using the object obj
obj.read_number()
```

Output:
999

References

- Python Programming Guide For Beginners - Part 1
- Python Programming Guide For Beginners - Part 2
- Python Programming Guide For Beginners - Part 3
- Python Programming Guide For Beginners - Part 4
- Simple Program For Beginners:
- Python Programming Guide For Beginners - Part 5
- Simple Program For Beginners Using List: