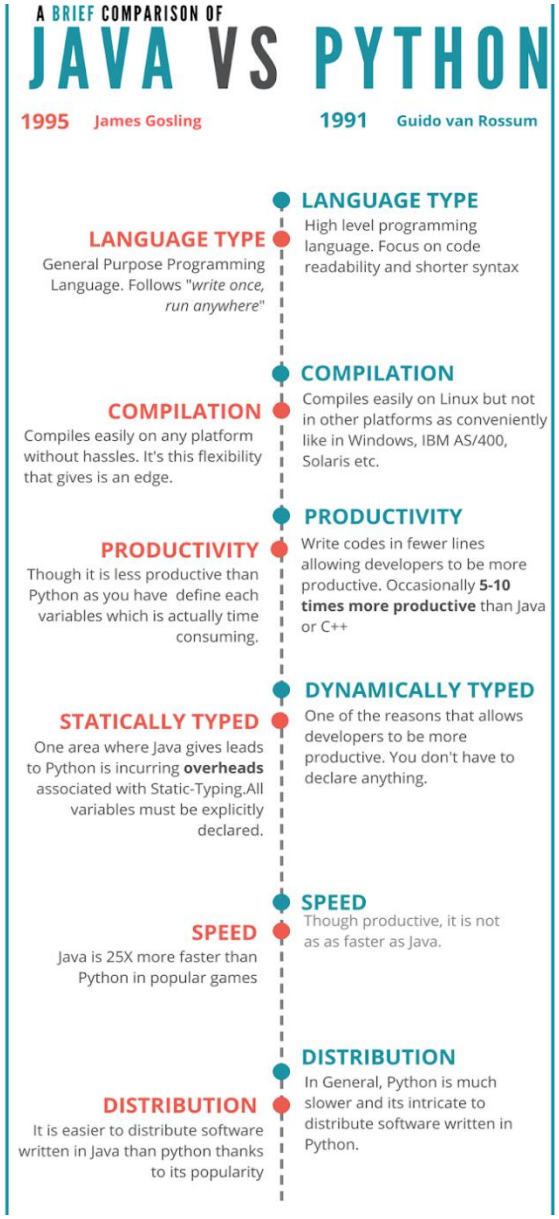
Python

Εισαγωγή

Python vs Java



"Hello, World"

- C

```
#include <stdio.h>

int main(int argc, char ** argv)
{
   printf("Hello, World!\n");
}
```

- Java

```
public class Hello
{
   public static void main(String argv[])
   {
      System.out.println("Hello, World!");
   }
}
```

- now in Python

```
print "Hello, World!"
```

File I/O in Java:

```
// get current directory
File dir = new File(".");
File fin=new File(dir.getCanonicalPath()
        + File.separator + "Code.txt");

FileInputStream fis =
            new FileInputStream(fin);

//Construct the BufferedReader object
BufferedReader in = new BufferedReader
         (new InputStreamReader(fis));

String aLine = null;
while ((aLine = in.readLine()) != null)
{//Process each line, here we count
   empty lines
    if (aLine.trim().length() == 0) {
    }
}
// do not forget to close the buffer
reader
in.close();
```

File I/O in Python:

```
myFile = open("/home/xiaoran/Desktop/
test.txt")

print myFile.read();
```

Advantages of Python over Java

Features	PYTHON	JAVA
Typing	Dynamically-typed	Statically-typed
Syntax	Semicolon is not needed to end statement	Error when semicolon is missed
Ease of Use	Codes are shorter therefore easier to use	Not easy when compared with Python
Productivity	More productive as codes can be written in fewer lines	Less productive as variables have to be defined

Γιατί Python?

https://www.python.org/

python

Scientific Computing

NumPy

SciPy

pandas

$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$

matplotlib

Machine Learning

scikit learn

DEAP

Deep Learning

TensorFlow

Keras

theano

mxnet

Microsoft Cognitive Toolkit

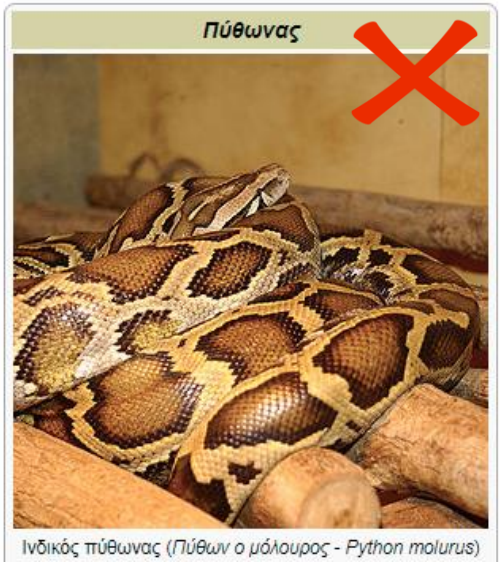
Caffe

Caffe2

Text mining

Natural Language Analyses with NLTK

gensim



Ινδικός πύθωνας (*Πύθων ο μόλουρος - Python molurus*)

Monty Pythons

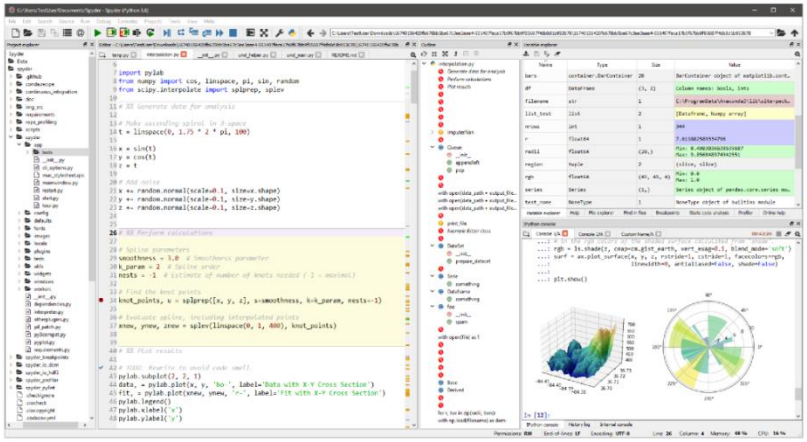
Γιατί Spyder?

https://www.spyder-ide.org/

- Spyder, the Scientific Python Development Environment, is a free integrated development environment (IDE) that is included with Anaconda.

- Spyder is a powerful scientific environment written in Python, for Python, and designed by and for scientists, engineers and data analysts.

- It offers a combination of the advanced editing, analysis, debugging, and profiling functionality of a comprehensive development tool with the data exploration, interactive execution, deep inspection, and visualization capabilities of a scientific package.



- Spyder offers built-in integration with many popular scientific packages, including NumPy, SciPy, Pandas, IPython, QtConsole, Matplotlib, SymPy, and more.

- Beyond its many built-in features, its abilities can be extended even further via its plugin system and API.

- Furthermore, Spyder can also be used as a PyQt5 extension library, allowing developers to build upon its functionality and embed its components, such as the interactive console, in their own PyQt software.

Components

Core building blocks of a powerful IDE

Editor

Work efficiently in a multi-language editor with a function/class browser, code analysis tools, automatic code completion, horizontal/vertical splitting, and go-to-definition.

IPython Console

Harness the power of as many IPython consoles as you like within the flexibility of a full GUI interface; run your code by line, cell, or file; and render plots right inline.

Variable Explorer

Interact with and modify variables on the fly: plot a histogram or timeseries, edit a dateframe or Numpy array, sort a collection, dig into nested objects, and more!

Profiler

Find and eliminate bottlenecks to unchain your code's performance.

Debugger

Trace each step of your code's execution interactively.

Help

Instantly view any object's docs, and render your own.

Εισαγωγή στην Python

Comments

Use a hash character # at the beginning of the text to write a comment in Python. Hash works with a single line, use # on the next line to comment again.

Code:

```
# I am just a comment
# Begin with # to write a comment
# It is so easy to write a comment in Python
```

Output:

Define the variables of different data types (1/2)

Code:

```
# Set String value (str data type) to a variable
a_str = "I am a string data type value"
print(a_str)

# Set Integer value (int data type) to a variable
a_int = 41
print(a_int)

# Set Float value (float data type) to a variable
a_float = 53.5
print(a_float)

# Set Complex value (complex data type) to a variable
a_complex = 5j
print(a_complex)

# Set List value (list data type) to a variable
a_list = ["Asia", "America", "Africa"]
print(a_list)

# Set Tuple value (tuple  data type) to a variable
a_tuple = ("Asia", "America", "Africa")
print(a_tuple)

# Set Range value (Range data type) to a variable
a_range = range(11)
print(a_range)

# Set Dictionary value (dict data type) to a variable
a_dict = {"Employee_id": 70923, "Employee_name": "Jonathan"}
print(a_dict)

# Set SET value (set data type) to a variable
a_set = {"Asia", "America", "Africa"}
print(a_set)

# Set frozenset value (frozenset data type) to a variable
a_frozenset = frozenset({"Asia", "America", "Africa"})
print(a_frozenset)
```

```

# Set Boolean value (bool data type) to a variable
a_bool = True
print(a_bool)

# Set Bytes value (bytes data type) to a variable
a_bytes = b"Hello"
print(a_bytes)

# Set Bytearray value (bytearray data type) to a variable
a_bytearray = bytearray(5)
print(a_bytearray)

# Set memoryview value (memoryview data type) to a variable
a_memoryview = memoryview(bytes(5))
print(a_memoryview)

# Use type() function to return class type of the argument(object)
passed as parameter.
print(type(a_str))
print(type(a_int))
print(type(a_float))
print(type(a_complex))
print(type(a_list))
print(type(a_tuple))
print(type(a_range))
print(type(a_dict))
print(type(a_set))
print(type(a_frozenset))
print(type(a_bool))
print(type(a_bytes))
print(type(a_bytearray))
print(type(a_memoryview))
```

Define the variables of different data types (2/2)

Output:

```
I am a string data type value
41
53.5
5j
['Asia', 'America', 'Africa']
('Asia', 'America', 'Africa')
range(0, 11)
{'Employee_id': 70923, 'Employee_name': 'Jonathan'}
{'Africa', 'America', 'Asia'}
frozenset({'Africa', 'America', 'Asia'})
True
b'Hello'
bytearray(b'\x00\x00\x00\x00\x00')
<memory at 0x00000271C43CDD08>
<class 'str'>
<class 'int'>
<class 'float'>
<class 'complex'>
<class 'list'>
<class 'tuple'>
<class 'range'>
<class 'dict'>
<class 'set'>
<class 'frozenset'>
<class 'bool'>
<class 'bytes'>
<class 'bytearray'>
<class 'memoryview'>
```

Assign values to multiple variables in one line

Code:

```
# Assigning Multiple values to Multiple Variables in single line
a_continent, b_continent, c_continent = "Asia", "America", "Africa"
print(a_continent)
print(b_continent)
print(c_continent)

# You can also assign same value to multiple variables in single line
a_continent = b_continent = c_continent = "Asia"
print(a_continent, b_continent, c_continent)
```

Output:

```
Asia
America
Africa
Asia Asia Asia
```

Type casting

In Python 3 the default data type of input value is a string (str). For example please go through the mentioned code and output given below:

Code: When Typecast is not used

```
# File name: type_cast_not_used.py
a = input()
b = a + 3
print(b)
```

Output:

```
3
Traceback (most recent call last):
  File "type_cast_not_used.py", line 2, in <module>
    b = a + 3
TypeError: can only concatenate str (not "int") to str
```

Explanation:

When you execute the above python script and entered 3 as input. Now we assume that a+3 will be calculated as 3 + 3 and print will return 6. But, it is not the case as input value will be treated as string type and string can only concatenate string type. Thus, TypeError will be returned.

Code: When Typecast is used

```
a = int(input())
b = a + 3
print(b)
```

Output:

```
3
6
```

Explanation:

When you execute the above python script and enter 3 as input. Sring type will be converted to int data type because we have used the int typecast in front of input() function. Thus, **a** variable will contain integer value and, b will perform the calculation correctly. The final output will be 6.

String Literal (1/6)

Literal refers to a specific value or a fixed value or a constant that is used in the program. Normally when literal consists of a single character, it refers to character literal. When literal consists of more than one character, it refers to as String literal. But, in the case of Python language, there is no character data type, so all single characters are strings with length one.

Code:

```
1. a_string_variable = 'Python Programming Course'
```

In the above code, the **'Python Programming Course'** is a string literal and **a_string_variable** is a string variable. In short, the string variable points to a string literal. Different string variables can point to a single literal but one string variable can point to only one literal at a given time.

String Literal (2/6)

String literal can be placed inside single quotes or double-quotes. The more formal way is to say that String literal is represented using single or double-quote delimiters.

Assigning string literal to a variable using single quotes as delimiters.

Code:

```
single_quote_variable = 'Hi I am placed inside single quotes'
```

Assigning string literal to a variable using double quotes as delimiters.

Code:

```
double_quote_variable = "Hi I am placed inside double quotes"
```

Combination of Single quote and Double quotes to represent String Literal

In python programming, there is no difference between string literal when placed inside single or double-quotes. It means there is no semantic difference between these two representations, it differs only in syntax.

Code:

```
single_quote_variable = 'Hi I am placed inside single quotes'

double_quote_variable = "Hi I am placed inside double quotes"
```

String Literal (3/6)

Code:

```
1. print('Place " double quotes when using single quote as delimiter')
2. print("Place ' single quotes when using double quote as delimiter")
3. a_double_inside_single = 'Place "Watch me" double quotes when using
single quote as delimiter'
4. b_single_inside_double = "Place 'Watch me' single quotes when using
double quote as delimiter"
5. print(a_double_inside_single)
6. print(b_single_inside_double)
```

Output:

```
1.  Place " double quotes when using single quote as delimiter
2.  Place ' single quotes when using double quote as delimiter
3.  Place "Watch me" double quotes when using single quote as delimiter
4.  Place 'Watch me' single quotes when using double quote as delimiter
```

String Literal (4/6)

Backslash character has special interpretation when placed inside Single quote and Double quotes.

Before explaining the importance of Backslash character. Think of a scenario when you need to place a single quote in string literal delimited by single quotes. Similarly, how will you place a double quote in string literal delimited by double quotes?

Code:

```
print('Place ' single quote when using single quote as delimiter')
print("Place " double quotes when using double quote as delimiter")
```

Output:

```
    print('Place ' single quote when using single quote as
delimiter')
                          ^
SyntaxError: invalid syntax
```

Explanation:

As soon as you place, the single quote character inside the string literal delimited by single quotes, the placed single quote character will act as the delimiter and the remaining string will not be delimited.

So, here comes the backslash character to the rescue. When a single quote is preceded by a backslash, it instructs the Python Interpreter to ignore the special meaning of a single quote and consider it as an ordinary character. The same goes for the double-quotes.

String Literal (5/6)

Code:

```
print('Place \' single quote with backslash when using single quote as delimiter')
print("Place \" double quotes with backslash when using double quote as delimiter")
```

Output:

```
Place ' single quote with backslash when using single quote as delimiter
Place " double quotes with backslash when using double quote as delimiter
```

Wait, it does not end here. Backslash has another purpose too. As we see, the backslash hides the special meaning of a single quote and double quote. Similarly, some characters when preceded by a backslash have special meaning. In the below code we are using \n i.e when n is preceded by a backslash it refers to new-line.

Code:

```
print("I am a message and \nI will print in two separate lines")
print(r"I am a message and \nI will print in two separate lines")
```

Output:

```
I am a message and
I will print in two separate lines
I am a message and \nI will print in two separate lines
```

The first print function will print in two separate lines because of the new-line character. But in the second case, r hides the special meaning of character preceded with a backslash.

String Literal (6/6)

You can use triple quotes to enclose strings with more than one line

Triple quotes can be a combination of either single quotes or double quotes

Code:

```
print('''I am a string literal
... has more than one
... line
....placed inside triple single quotes ''')
print("""I am a string literal
... has more than one
... line
....placed inside triple double quotes """)
```

Output:

```
I am a string literal
... has more than one
... line
....placed inside triple single quotes
I am a string literal
... has more than one
... line
....placed inside triple double quotes
```

Print() Function (1/6)

Print function with Single and Double Quotes

You can print any text if placed inside single and double quotes inside the print() function.

Code: When using single and Double quotes

```
print('Demo print: Enclosed inside single quotes')
print("Demo print: Enclosed inside double quotes")
```

Output:

```
Demo print: Enclosed inside single quotes
Demo print: Enclosed inside double quotes
```

Print() Function (2/6)

Printing Variables using print() function

Now, we will discuss how to print variables without a format specifier. As you we discussed in thumb rule, you can simply use any number of argument inside a print function. To print a variable without format specifier just print the name of the variable.

Code: Printing Variable without Format String

```
1. x = 5
2. print("Let's print Variable x",x, 'See it is working')
```

Output:

```
1.  Let's print Variable x 5 See it is working
```

Print() Function (3/6)

Performing mathematics inside print() function

We can also perform mathematic operations inside the print() function. The mathematics expression will evaluate to a numeric data value, thus should not be placed inside quotes.

Code: Performing maths operation inside print() function

```
x = 5
y = 10
z = 10
print("Let's perform addition", x + y + z)
print("Let's perform Subtraction", x - y, z - x)
print("Let's perform Multiplication", x * y * z)
print("Let's perform Division", z / x)
```

Output:

```
Let's perform addition 25
Let's perform Subtraction -5 5
Let's perform Multiplication 500
Let's perform Division 2.0
```

Print() Function (4/6)

Printing using f-String

Let's take an example to understand the need for the format string. If you want to display the value of a variable inside a string placed between a single or double-quotes. Please go through the following code and output.

Code: Trying to print

```
1. x = 105
2. y = "Python 3 Program"
3. print("x y")
4. print(f"{x} {y}")
```

Output:

```
1.  x y
2.  105 Python 3 Program
```

Print() Function (5/6)

Code: Assigning format string to a variable and use it to print

```
1. x = 1209
2. y = "The value of x is"
3. z = f"{y} {x}"
4. print(f"{x}")
5. print(f"{y}")
6. print(z)
7. print(f"x value is : {x}")
8. print(f"The value y is : {y}")
```

Output:

```
1.  1209
2.  The value of x is
3.  The value of x is 1209
4.  x value is : 1209
5.  The value y is : The value of x is
```

Print() Function (6/6)

string.format()

1. **Where string refers to a string variable or string placed using single or double quotes**
2. **Dot format .format() is the syntax to call the function**

Input type: format() takes input as any Data type.

Return type: string.format() function returns a string with substituted values used for curly brackets positions.

```
1. a = "I am a {} programmer with {} years of
experience".format("Python", 10)
2. print(a)
3. print("Learn {} {} by practise".format("Python", "Programming"))
4.
5. format_string = "{} {} {} {}"
6. print(format_string.format(1, 3, 4, 9))
7. print(format_string.format("I am", "a", "Python", "Programmer"))
8. print(format_string.format("I have", 10, "years of experience", "In
Python"))
9. print(format_string)
```

Output:

```
1.  I am a Python programmer with 10 years of experience
2.  Learn Python Programming by practise
3.  1 3 4 9
4.  I am a Python Programmer
5.  I have 10 years of experience In Python
6.  {} {} {} {}
```

Input function (1/2)

You can simply use input() function for input data, the data entered using input is treated as a string.

Code: How to use the input function

```
print("Enter your age")
a=input()
print("Entered age is {}".format(a))
print("The type of value entered is {}".format(type(a)))
```

Output:

```
Enter your age
34
Entered age is 34
The type of value entered is <class 'str'>
```

Explanation:

The type returned as a string in the output for the age data because input data is treated as a string.

Input function (2/2)

You can also print message while input

Code: Print message while input

```
a = int(input("Enter your age: "))
print("Entered age is %d" % a)
```

Output:

```
Enter your age: 345
Entered age is 345
```

Operators (1/2)

Arithmetic operators in Python

Operator	Name	Example
+	Addition	a + b
-	Subtraction	a - b
*	Multiplication	a * b
/	Division	a / b
%	Modulus	a % b
**	Exponentiation	a ** b
//	Floor division	a // b

Comparison operators in Python

Operator	Name	Example
==	Equal	a == b
!=	Not equal	a != b
>	Greater than	a > b
<	Less than	a < b
>=	Greater than or equal to	a >= b
<=	Less than or equal to	a <= b

Assignment operators in Python

Operator	Example	Equivalent to
=	a = 5	a = 5
+=	a += 3	a= a + 3
-=	a -= 3	a = a - 3
*=	a *= 3	a = a * 3
/=	a /= 3	a= a / 3
%=	a %= 3	a = a % 3
//=	a //= 3	a = a // 3
**=	a **= 3	a = a ** 3
&=	a &= 3	a = a & 3
|=	a |= 3	a = a | 3
^=	a^= 3	a= a ^ 3
>>=	a >>= 3	a = a >> 3
<<=	a <<= 3	a = a << 3

Operators (2/2)

Logical operators in Python

Operator	Description	Example
AND	Returns True if both statements are true	a < 4 and a < 11
OR	Returns True if one of the statements is true	a < 7 or x < 3
NOT	Reverse the result, returns False if the result is true	not(a < 1 and a < 5)

Identity operators in Python

Operator	Description	Example
is	Returns true if both variables are the same object	a is b
is not	Returns true if both variables are not the same object	a is not b

Membership operators in Python

Operator	Description	Example
in	Returns True if a sequence with the specified value is present in the object	a in b
not in	Returns True if a sequence with the specified value is not present in the object	a not in b

Bitwise operators in Python

Operator	Name	Description
&	AND	Sets each bit to 1 if both bits are 1
|	OR	Sets each bit to 1 if one of two bits is 1
^	XOR	Sets each bit to 1 if only one of two bits is 1
~	NOT	Inverts all the bits
<<	Zero fill left shift	Shift left by pushing zeros in from the right and let the leftmost bits fall off
>>	Signed right shift	Shift right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off

Boolean Operators

Printing Boolean Value using comparison operators

Code:

```
a = 3
b = 4
print(a == b)
print(a != b)
print(a > b)
print(a < b)
print(a >= b)
print(a <= b)
```

Output:

```
False
True
False
True
False
True
```

Printing Boolean Value using Identity operators

Code:

```
a = 3
b = 4
print(a is b)
print(a is not b)
```

Output:

```
False
True
```

Printing Boolean Value using logical operators

Code:

```
a = 3
b = 4
print(a < b and a < 10)
print(a > b or a < 5)
print(not(a > b or a < 5))
```

Output:

```
True
True
False
```

Printing Boolean Value using Membership operators

Code:

```
a = [2, 3, 4, 5]
b = 4
c = 1
print(b in a)
print(c not in a)
```

Output:

```
True
True
```

Flow Control

Code:

```
a = 6
b = 4
c = 4
if a > b:
    print("{} is greater than {}".format(a, b))

if a > c:
    print("{} is less than {}".format(c, a))

if b == c:
    print("{} is equal to {}".format(b, c))

if a != c:
    print("{} is not equal to {}".format(a, c))

if a >= b:
    print("{} is greater than equal to {}".format(a, b))

if a > c:
    print("{} is less than equal to {}".format(c, a))
```

Output:

```
6 is greater than 4
4 is less than 6
4 is equal to 4
6 is not equal to 4
6 is greater than equal to 4
4 is less than equal to 6
```

Python Programming with if-else block

We will make a program to find the lowest integer number out of two unique integers.

Code:

```
a = int(input("Enter first integer: "))
b = int(input("Enter Second integer: "))

if a < b:
    print("{} is the lowest integer".format(a))
else:
    print("{} is the lowest integer".format(b))
```

Output:

```
Enter first integer: 4
Enter Second integer: 3
3 is the lowest integer
```

Python Programming with if-elif-else block

Code:

```
a = b = 4
c = 5
if a > b:
    print("{} is greater than {}".format(a, b))
elif b == c:
    print("{} is equal to {}".format(b, c))
else:
    print("Neither a is greater than b Nor b is equal to c ")
```

Output:

```
Neither a is greater than b Nor b is equal to c
```

Flow Control (1/3)

Python Programming with Nested if

We will make a program to find the greatest number out of three unique integer numbers.

Test Cases:

```
# a b c
# 3 4 5
# 3 5 4
# 4 3 5
# 4 5 3
# 5 3 4
# 5 4 3
```

Code:

```
a = int(input("Enter first integer: "))
b = int(input("Enter Second integer: "))
c = int(input("Enter Third integer: "))
if a > b:
    if a > c:
        print("{} is the greatest element".format(a))
    else:
        print("{} is the greatest element".format(c))

elif b > c:
    print("{} is the greatest element".format(b))

else:
    print("{} is the greatest element".format(c))
```

Output:

```
Enter first integer: 3
Enter Second integer: 4
Enter Third integer: 5
5 is the greatest element
```

Flow Control (2/3)

Python Programming with for loop

You need a loop construct when you need to perform the operation which is to be repeated over a list or range of data. In this section, we will discuss the possible uses of for loop in Python 3 programming.

Programming for loop over a list of Data

Code:

```
a = [0, 1, 2, 3, 4]
for i in a:
    a[i] += 1

print("List after increment: {}".format(a))
```

Output:

```
List after increment: [1, 2, 3, 4, 5]
```

Programming for loop over a range of numbers

We will also use the range data type, which we discussed in an earlier article **Data Types available in Python**

Code:

```
a = ["one", "two", "three", "four", "five"]
for i in range(0, 5):
    print("Value of i at {} step is {}".format(a[i], i + 1))
```

Output:

```
Value of i at one step is 1
Value of i at two step is 2
Value of i at three step is 3
Value of i at four step is 4
Value of i at five step is 5
```

Flow Control (3/3)

Python Programming using while loop

Code:

```
a = ["one", "two", "three", "four", "five"]
i = 0
while i < 5:
    print(a[i])
    i += 1
```

Output:

```
one
two
three
four
five
```

Functions (1/2)

Programming function with Zero Argument

Code:

```
def greater_function():
    print("a is greater than b")


def smaller_function():
    print("a is smaller than b")


a = 3
b = 7
if a > b:
    greater_function()
else:
    smaller_function()
```

Output:

```
a is smaller than b
```

Functions (2/2)

Programming function that accepts arguments

We will make a program with basic calculator functionalities that provide addition, multiplication, subtraction and division functions.

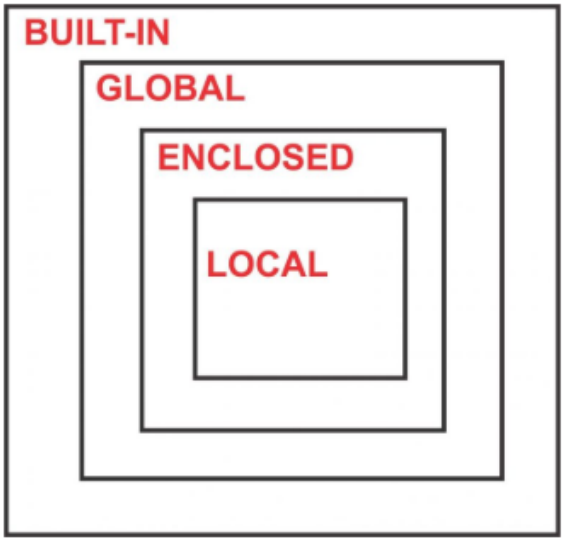
Code:

```
def addition_fun(a, b):
    return a + b


def multiply_fun(a, b):
    return a * b


def subtraction_fun(a, b):
    return a - b


def division_fun(a, b):
    return a / b


x = int(input("Enter any integer value: "))
y = int(input("Enter any integer value: "))
print("Result after Addition is {}".format(addition_fun(x, y)))
print("Result after Multiplication is {}".format(multiply_fun(x, y)))
print("Result after Subtraction is {}".format(subtraction_fun(x, y)))
print("Result after Division is {}".format(division_fun(x, y)))
```

Output:

```
Enter any integer value: 10
Enter any integer value: 5
Result after Addition is 15
Result after Multiplication is 50
Result after Subtraction is 5
Result after Division is 2.0
```

Scope of a Variable (1/5)



- **Local(L)**: Defined inside a function/class or enclosed within a nested loop or conditional construct.
- **Enclosed(E)**: Defined inside enclosing functions (Nested functions)
- **Global(G)**: Defined at the topmost level
- **Built-in(B)**: Reserved Keywords in Python built-in functions/modules/classes

Scope of a Variable (2/5)

Program to explain the local scope of a variable

Code:

```
var_scope = 'I am a variable with global scope'


def local_scope_func():
    var_scope = "I am a variable with local scope"
    print(var_scope)


local_scope_func()
```

Output:

```
I am a variable with local scope
```

Explanation:

In the above program, though we have defined two variables with the same name, one is at the topmost level and the other is inside the function. We can clearly see after executing our program that the variable inside the function is printed. It means the function searches the variable inside the local boundary first, If it is not found then only it will access the global variable.

Scope of a Variable (3/5)

Program to explain the local and global scope of a variable

Code:

```
var_scope = 'I am a variable with global scope'


def local_scope_func():
    var_scope = "I am a variable with local scope"
    print(var_scope)


local_scope_func()
print(var_scope)
```

Output:

```
I am a variable with local scope
I am a variable with global scope
```

Explanation:

The output for the first line is clearly understood as discussed above. The print(var_scope) does not have access to the variable defined inside the function local_scope_fun() so it will access the global variable **var_scope** declared at the topmost level without any doubt.

Scope of a Variable (4/5)

Program to explain the local, enclosed and global scope of a variable

Code:

```
var_a = 'global variable'

def Function_at_level_one():
    var_a = 'variable at level one'
    def Function_at_level_two():
        # var_a = 'variable at level two'
        nonlocal var_a
        print(var_a)
    Function_at_level_two()

Function_at_level_one()
print(var_a)
```

Output:

```
variable at level one
global variable
```

Explanation:

In this program, we have used keyword **nonlocal**, which is used to access the variable declared in the outer function.

Scope of a Variable (5/5)

Program to explain the local, enclosed, global and Built-in scope of a variable

Code:

```
from math import pi


# pi = 'global pi variable'

def outer():
    # pi = 'outer pi variable'
    def inner():
        # pi = 'inner pi variable'
         print(pi)

    inner()


outer()
```

Output:

```
3.141592653589793
```

Explanation:

In this program, we have imported built-in variable **pi** from the math package. When we execute the function, it first searches the **pi** variable inside **inner()**, when it is not found it searches inside the **outer()** function, when it is not found, it searches the variable at the global level. When it is not found at global level, it uses the built-in **pi** variable to print

File handling (1/3)

Name of the Function	Role of Function
open	It is use to open the given file.
close	Closes the file.
read	Reads the contents of the file. You can assign the result to a variable.
readline	Reads just one line of a text file.
truncate	Empties the file.
write('stuff')	Writes "stuff" to the file.
seek(0)	Moves the read/write location to the beginning of the file.

File handling (2/3)

Program Implemetation: Open | Truncate | Write | Close

Code:

```
1. from sys import argv
2.
3. program_name, filename = argv
4. print("We are going to erase the content of the given file first, if
the file exists")
5. print("Opening the File....")
6. destination = open(filename, 'w')
7. print("Erasing the content of the file....")
8. destination.truncate()
9. print("Now Enter the data you want: In three lines")
10. Line1 = input("Line 1: ")
11. Line2 = input("Line 2: ")
12. Line3 = input("Line 3: ")
13. print("These three lines will be written")
14. destination.write(Line1)
15. destination.write("\n")
16. destination.write(Line2)
17. destination.write("\n")
18. destination.write(Line3)
19. destination.write("\n")
20. print("Now the file is set to close")
21. destination.close()
```

Output:

```
1.  We are going to erase the content of the given file first, if the
    file exists
2.  Opening the File....
3.  Erasing the content of the file....
4.  Now Enter the data you want: In three lines
5.  Line 1: I love Python Programming
6.  Line 2: I love to code using Python
7.  Line 3: I am good at Python Programming
8.  These three lines will be written
9.  Now the file is set to close
```

File handling (3/3)

Program Implementation: Copy data of One File to Another

Code:

```
from sys import argv
from os.path import exists

script, from_file, to_file = argv

print(f"Copying from {from_file} to {to_file}")

# we could do these two on one line, how?
input_file = open(from_file)
input_data = input_file.read()

print(f"The input file is {len(input_data)} bytes long")

print(f"Does the output file exist? {exists(to_file)}")
print("Ready, hit RETURN to continue, CTRL-C to abort.")
input()

output_file = open(to_file, 'w')
output_file.write(input_data)

print("File is Copied Successfully")
```

Output:

```
Copying from sample.txt to sample_copy.txt
The input file is 86 bytes long
Does the output file exist? False
Ready, hit RETURN to continue, CTRL-C to abort.

File is Copied Successfully
```

Important Built in Functions

1. type() function in Python
2. abs() function in Python
3. all () function in Python
4. any () function in Python
5. bin () function in Python
6. bool () function in Python
7. float () function in Python
8. int () function in Python
9. str () function in Python
10. len () function in Python
11. list () function in Python
12. sorted () function in Python
13. reversed () function in Python
14. round () function in Python
15. iter () function in Python
16. next () function in Python
17. range () function in Python
18. sum () function in Python
19. min () function in Python
20. max () function in Python

Working with list (1/7)

STORE ITEMS, TRAVERSE LIST, ACCESS USING INDEX

The following code demonstrates how to store items in a list, how to traverse a list, how to access elements of a list using index.

Code:

```
Nordic_Country_List = ["FINLAND", "ICELAND", "NORWAY", "DENMARK",
"SWEDEN", "FAROE ISLANDS"]
Asian_Country_List = ["CHINA", "INDIA", "INDONESIA", "PAKISTAN",
"BANGLADESH", "JAPAN", "PHILIPPINES", "VIETNAM"]

for i in Nordic_Country_List:
    print(i, end=" ")
print("\n")
for i in Asian_Country_List:
    print(i, end=" ")
print("\n")
#Lets calculate the size of the List
#Then we access element of List using index
Nordic_length = len(Nordic_Country_List)
Asian_length = len(Asian_Country_List)

for i in range(Nordic_length):
    print(Nordic_Country_List[i], end=" ")
print("\n")

for i in range(Asian_length):
    print(Asian_Country_List[i], end=" ")
```

Output:

```
FINLAND ICELAND NORWAY DENMARK SWEDEN FAROE ISLANDS

CHINA INDIA INDONESIA PAKISTAN BANGLADESH JAPAN PHILIPPINES VIETNAM

FINLAND ICELAND NORWAY DENMARK SWEDEN FAROE ISLANDS

CHINA INDIA INDONESIA PAKISTAN BANGLADESH JAPAN PHILIPPINES VIETNAM
```

Working with list (2/7)

Append, Insert, Extend, Index, Remove, Sort, Reverse, Pop.

Syntax:

```
list.append(elem)
-- adds a single element to the end of the list. Common error: does not return
the new list, just modifies the original.
list.insert(index, elem)
-- inserts the element at the given index, shifting elements to the right.
list.extend(list2)
-- adds the elements in list2 to the end of the list. Using + or += on a list
is similar to using extend().
list.index(elem)
-- searches for the given element from the start of the list and returns its
index. Throws a ValueError if the element does not appear (use "in" to check
without a ValueError).
list.remove(elem)
-- searches for the first instance of the given element and removes it (throws
ValueError if not present)
list.sort()
-- sorts the list in place (does not return it). (The sorted() function shown
later is preferred.)
list.reverse()
 -- reverses the list in place (does not return it)
list.pop(index)
 -- removes and returns the element at the given index. Returns the rightmost
element if index is omitted (roughly the opposite of append()).
```

Working with list (3/7)

Code:

```
Asian_Country_List = ["CHINA", "INDIA", "INDONESIA", "PAKISTAN",
"BANGLADESH", "JAPAN", "PHILIPPINES", "VIETNAM"]


#append element at end of list
Asian_Country_List.append('Thailand')
print(Asian_Country_List)
#insert element at index 0
Asian_Country_List.insert(0, 'Malaysia')
print(Asian_Country_List)
#add list of elements at end
Asian_Country_List.extend(['North Korea', 'South Korea'])
print(Asian_Country_List)

print(Asian_Country_List.index('INDONESIA'))

#search and remove that element
Asian_Country_List.remove('BANGLADESH')
print(Asian_Country_List)

#Remove third element in the list
Asian_Country_List.pop(2)
print(Asian_Country_List)

#sorts the list in place (does not return it)
Asian_Country_List.sort()
print(Asian_Country_List)

#reverses the list in place (does not return it)
Asian_Country_List.reverse()
print(Asian_Country_List)
```

Output:

```
['CHINA', 'INDIA', 'INDONESIA', 'PAKISTAN', 'BANGLADESH', 'JAPAN',
'PHILIPPINES', 'VIETNAM', 'Thailand']
['Malaysia', 'CHINA', 'INDIA', 'INDONESIA', 'PAKISTAN', 'BANGLADESH',
'JAPAN', 'PHILIPPINES', 'VIETNAM', 'Thailand']
['Malaysia', 'CHINA', 'INDIA', 'INDONESIA', 'PAKISTAN', 'BANGLADESH',
'JAPAN', 'PHILIPPINES', 'VIETNAM', 'Thailand', 'North Korea', 'South
Korea']
3
['Malaysia', 'CHINA', 'INDIA', 'INDONESIA', 'PAKISTAN', 'JAPAN',
'PHILIPPINES', 'VIETNAM', 'Thailand', 'North Korea', 'South Korea']
['Malaysia', 'CHINA', 'INDONESIA', 'PAKISTAN', 'JAPAN',
'PHILIPPINES', 'VIETNAM', 'Thailand', 'North Korea', 'South Korea']
['CHINA', 'INDONESIA', 'JAPAN', 'Malaysia', 'North Korea',
'PAKISTAN', 'PHILIPPINES', 'South Korea', 'Thailand', 'VIETNAM']
['VIETNAM', 'Thailand', 'South Korea', 'PHILIPPINES', 'PAKISTAN',
'North Korea', 'Malaysia', 'JAPAN', 'INDONESIA', 'CHINA']
```

Working with list (4/7)

Empty List

One popular trick is to use an empty list [] and use an "append" or "extend" method to add elements to the list.

Code:

```
Nordic_Country_List = ["FINLAND", "ICELAND", "NORWAY", "DENMARK",
"SWEDEN", "FAROE ISLANDS"]
Asian_Country_List = ["CHINA", "INDIA", "INDONESIA", "PAKISTAN",
"BANGLADESH", "JAPAN", "PHILIPPINES", "VIETNAM"]

Country_List = []
for i in range(0, 5):
    Country_List.append(Asian_Country_List[i])

print(Country_List)

Country_List_2 = []
Country_List_2.extend(Nordic_Country_List)
print(Country_List_2)

Country_List_3 = []
Country_List_3.extend(["CHINA", "INDIA", "INDONESIA"])
print(Country_List_3)
```

Output:

```
['CHINA', 'INDIA', 'INDONESIA', 'PAKISTAN', 'BANGLADESH']
['FINLAND', 'ICELAND', 'NORWAY', 'DENMARK', 'SWEDEN', 'FAROE
ISLANDS']
['CHINA', 'INDIA', 'INDONESIA']
```

Working with list (5/7)

List Slices

Slices work on lists just as with strings, and can also be used to change sub-parts of the list.

Code:

```
1. Nordic_Country_List = ["FINLAND", "ICELAND", "NORWAY", "DENMARK",
"SWEDEN", "FAROE ISLANDS"]
2. Asian_Country_List = ["CHINA", "INDIA", "INDONESIA", "PAKISTAN",
"BANGLADESH", "JAPAN", "PHILIPPINES", "VIETNAM"]
3.
4. #Replacing first two items in the list with "GREENLAND"
5. Nordic_Country_List[0:1] = ["GREENLAND"]
6. print(Nordic_Country_List)
7.
8. #Replacing first three items in the list with TWO ITEMS "GREENLAND"
and "IRELAND"
9. Nordic_Country_List[0:2] = ["GREENLAND", "IRELAND"]
10. print(Nordic_Country_List)
11.
12. #Replacing first four items in the list with TWO ITEMS "GREENLAND"
and "IRELAND"
13. Nordic_Country_List[0:4] = ["GREENLAND", "IRELAND"]
14. print(Nordic_Country_List)
```

Output:

```
1. ['GREENLAND', 'ICELAND', 'NORWAY', 'DENMARK', 'SWEDEN', 'FAROE
   ISLANDS']
2. ['GREENLAND', 'IRELAND', 'NORWAY', 'DENMARK', 'SWEDEN', 'FAROE
   ISLANDS']
3. ['GREENLAND', 'IRELAND', 'SWEDEN', 'FAROE ISLANDS']
```

Working with list (6/7)

Negative indexes in List

We can use negative indexes to access the list elements from the end. -1 to access the last element, -2 to access second last and so on.

Using Positive index 0 to n-1 from first to the last element. Using Negative index -1 to -n from the last element to the first.

In the below code, we will demonstrate, how to initialize a tuple and traverse the list using a for loop.

```
a = [1, 2, 3, 4, 5, 6]

a_length = -len(a)

for i in range(-1, a_length-1, -1):
    print(a[i])
```

Output:

```
6
5
4
3
2
1
```

Working with list (7/7)

Multidimensional List

We can use nest lists within lists. With the help of a single-dimensional list, we can use the concept of an array. Similarly using the concept of a Multidimensional list we can use the concept of Multidimensional array.

Code:

```
a = [1, 2, 3] # 1Dimensional LIST of order 1 by 3
b = [5, 6, 6]  # 1Dimensional LIST of order 1 by 3
c = [102, 5, -7] # 1Dimensional LIST of order 1 by 3

x1 = [[1, 2, 4], [2, 4, 5], [0, 0, 0]]  # 2Dimensional LIST of order 3 by 3
x2 = [[5, 6, 7], [11, 14, 45], [90, 87, 73]]  # 2Dimensional LIST 3 by 3
x3 = [[7, 11, 19], [15, 154, 415], [190, 807, 723]] # 2Dimensional LIST 3 by 3


d = [a, b, c] # 2Dimensional LIST of order 3 by 3
#D1 D2
D1 = len(d)
D2 = len(d[0])

for i in range(D1):
    for j in range(D2):
        print(d[i][j], end=" ")
    print("\t")

print("\n")
x = [x1, x2, x3] # 3Dimensional LIST of order 3 by 3 by 3
#X1 X2
X1 = len(x)
X2 = len(x[0])
X3 = len(x[0][0])

for i in range(X1):
    for j in range(X2):
        for k in range(X3):
            print(x[i][j][k], end=" ")
        print("\t")
    print("\n")
```

Output:

```
1 2 3
5 6 6
102 5 -7


1 2 4
2 4 5
0 0 0


5 6 7
11 14 45
90 87 73


7 11 19
15 154 415
190 807 723
```

Working with Sets (1/5)

Initializing and Traversing a Set

In the below code, we will demonstrate, how to initialize a set and traverse the set using for loop. Sets can be created using the function set().

Code:

```
a = {1, 2, 3}

for i in a:
    print(i)
```

Output:

```
1
2
3
```

How to add elements to a Set

Sets are mutable. But since they are unordered, indexing has no meaning.

We cannot access or change an element of a set using indexing or slicing. Sets does not support it.

We can add a single element using the **add()** method and multiple elements using the update() method. The **update()** method can take tuples, lists, strings or other sets as its argument. In all cases, duplicates are avoided.

Code:

```
a = {1, 2, 3}

for i in a:
    print(i)

a.add(4)
print(a)

#We are adding list as well as set to a set
#We are also trying to add duplicate element 1 to existing set
a.update([5, 6, 7], {1, 9, 10})
print(a)
```

Output:

```
1
2
3
{1, 2, 3, 4}
{1, 2, 3, 4, 5, 6, 7, 9, 10}
```

Working with Sets (2/5)

Remove elements from a Set

A particular item can be removed from the set using methods, **discard()** and **remove()**.

The only difference between the two is that, while using discard() if the item does not exist in the set, it remains unchanged. But remove() will raise an error in such condition.

Similarly, we can remove and return an item using the **pop()** method.

Set being unordered, there is no way of determining which item will be popped. It is completely arbitrary.

We can also remove all items from a set using **clear()**.

Code:

```
a_set = {1, 2, 3, 4, 5, 6}
print(a_set)

a_set.discard(3)
print(a_set)

a_set.remove(5)
print(a_set)

a_set.pop()
print(a_set)

a_set.clear()
print(a_set)
```

Output:

```
{1, 2, 3, 4, 5, 6}
{1, 2, 4, 5, 6}
{1, 2, 4, 6}
{2, 4, 6}
set()
```

Working with Sets (3/5)

Set Membership Test

We can test if an item exists in a set or not, using the keyword in

Code:

```
a_set = {1, 2, 3, 4, 5, 6}

print(1 in a_set)
print(101 in a_set)
```

Output:

```
True
False
```

Working with Sets (4/5)

Set Operations similar to Mathematics

Set Union operation

Union of two sets X and Y is a set of all unique elements from both sets.

A union is performed using **|** operator. The same can be accomplished using the method **union()**.

Code:

```
a_set = {1, 2, 3, 4, 5, 6}
b_set = {1, 7, 8, 9, 10}

c_set = a_set | b_set
print(c_set)

c_set = a_set.union(b_set)
print(c_set)
```

Output:

```
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
```

Set Intersection operation

The intersection of two sets X and Y is a set of elements that are common in both sets.

The intersection is performed using **&** operator. The same can be accomplished using the method **intersection()**.

Code:

```
a_set = {1, 2, 3, 4, 5, 6}
b_set = {1, 7, 8, 9, 10}

c_set = a_set & b_set
print(c_set)

c_set = a_set.intersection(b_set)
print(c_set)
```

Output:

```
{1}
{1}
```

Working with Sets (5/5)

Set Difference operation

A difference of two sets X and Y refers to (X – Y) is a set of elements that are only in X but not in Y. Similarly, Y – X is a set of the element in Y but not in X.

The difference is performed using – minus operator. The same can be accomplished using the method **difference()**.

Code:

```
a_set = {1, 2, 3, 4, 5, 6}
b_set = {1, 7, 8, 9, 10}

c_set = a_set - b_set
print(c_set)

c_set = a_set.difference(b_set)
print(c_set)

d_set = b_set - a_set
print(d_set)

d_set = b_set.difference(a_set)
print(d_set)
```

Output:

```
{2, 3, 4, 5, 6}
{2, 3, 4, 5, 6}
{8, 9, 10, 7}
{8, 9, 10, 7}
```

Set Symmetric Difference operation

Symmetric Difference of two sets X and Y is a set of elements in both X and Y except those that are common in both.

The symmetric difference is performed using ^ operator. The same can be accomplished using the method **symmetric_difference()**.

Code:

```
a_set = {1, 2, 3, 4, 5, 6}
b_set = {1, 7, 8, 9, 10}

c_set = a_set ^ b_set
print(c_set)

c_set = a_set.symmetric_difference(b_set)
print(c_set)
```

Output:

```
{2, 3, 4, 5, 6, 7, 8, 9, 10}
{2, 3, 4, 5, 6, 7, 8, 9, 10}
```

Working with Tuples (1/7)

Difference b/w List & Tuple

List	Tuple
Elements of a list are Mutable	Elements of a tuple are immutable.
When you need to change data over time, list is used.	When you need to use data that does not changes over time, tuple is used.
Traversing items in a List is slow as compared to Tuple.	Traversing items in a Tuple is fast as compared to List.
Elements of list are enclosed in square bracket.	Elements of a tuple are enclosed in parenthesis.

Working with Tuples (2/7)

Initializing and Traversing a Tuple using index & Loop

In the below code, we will demonstrate, how to initialize a tuple and traverse the tuple using a for loop.

Code:

```
a = (1, 2, 3, 4, 5, 6)

a_length = len(a)
for i in range(a_length):
    print(a[i])
```

Output:

```
1
2
3
4
5
6
```

Working with Tuples (3/7)

Negative indexes in tuples

Similar to list and strings we can use negative indexes to access the tuple elements from the end. -1 to access the last element, -2 to access second last and so on.

Using Positive index 0 to n-1 from first to the last element. Using Negative index -1 to -n from last element to the first.

In the below code, we will demonstrate, how to initialize a tuple and traverse the tuple using a for loop.

Code:

```
a = (1, 2, 3, 4, 5, 6)

a_length = -len(a)

for i in range(-1, a_length-1, -1):
    print(a[i])
```

Output:

```
6
5
4
3
2
1
```

Working with Tuples (4/7)

Using Nested tuples

Code:

```
a = (1, 2, 3) # 1Dimensional TUPLE of order 1 by 3
b = (5, 6, 6)  # 1Dimensional TUPLE of order 1 by 3
c = (102, 5, -7) # 1Dimensional TUPLE of order 1 by 3

x1 = ((1, 2, 4), (2, 4, 5), (0, 0, 0))  # 2Dimensional TUPLE of order 3 by 3
x2 = ((5, 6, 7), (11, 14, 45), (90, 87, 73))  # 2Dimensional TUPLE 3 by 3
x3 = ((7, 11, 19), (15, 154, 415), (190, 807, 723)) # 2Dimensional TUPLE 3 by 3


d = (a, b, c) # 2Dimensional TUPLE of order 3 by 3
#D1 D2
D1 = len(d)
D2 = len(d[0])

for i in range(D1):
    for j in range(D2):
        print(d[i][j], end=" ")
    print("\t")

print("\n")
x = [x1, x2, x3] # 3Dimensional TUPLE of order 3 by 3 by 3
#X1 X2
X1 = len(x)
X2 = len(x[0])
X3 = len(x[0][0])

for i in range(X1):
    for j in range(X2):
        for k in range(X3):
            print(x[i][j][k], end=" ")
        print("\t")
    print("\n")
```

Output:

```
1 2 3
5 6 6
102 5 -7


1 2 4
2 4 5
0 0 0


5 6 7
11 14 45
90 87 73


7 11 19
15 154 415
190 807 723
```

Working with Tuples (5/7)

Changing & Deleting Elements of a Tuple

We cannot change the elements of a tuple because tuple itself is immutable. However, we can change the elements of nested items that are mutable. For example, in the following code, we are changing the element of the list which is present inside the tuple. List items are mutable that's why it is allowed.

Tuple elements are immutable which also means that we cannot delete the elements of a tuple. However, deleting entire tuple is possible.

Code:

```
a = (1, 2, 3, ["I", "Love", "Python"], 4, 5, 6)

print(a)

a[3][0], a[3][1], a[3][2] = "You", "Hate", "Python"
print(a)

del a
#You will get error, as the tuple is deleted
print(a)
```

Output:

```
(1, 2, 3, ['I', 'Love', 'Python'], 4, 5, 6)
(1, 2, 3, ['You', 'Hate', 'Python'], 4, 5, 6)
Traceback (most recent call last):
  File "practise.py", line 10, in <module>
    print(a)
NameError: name 'a' is not defined
```

Working with Tuples (6/7)

Slicing in Tuple

Slices work on Tuple just as with strings and lists, and can also be used to change sub-parts of the Tuple.

Code:

```
Nordic_Country_Tuple = ("FINLAND", "ICELAND", "NORWAY", "DENMARK", "SWEDEN", "FAROE ISLANDS")

print(Nordic_Country_Tuple[0:1])

print(Nordic_Country_Tuple[0:2])

print(Nordic_Country_Tuple[0:4])

# displaying entire tuple
print(Nordic_Country_Tuple[:])

print(Nordic_Country_Tuple[4:-1])
```

Output:

```
('FINLAND',)
('FINLAND', 'ICELAND')
('FINLAND', 'ICELAND', 'NORWAY', 'DENMARK')
('FINLAND', 'ICELAND', 'NORWAY', 'DENMARK', 'SWEDEN', 'FAROE ISLANDS')
('SWEDEN',)
```

Working with Tuples (7/7)

Membership Test in Tuples

in: Checks whether an element exists in the specified tuple.
not in: Checks whether an element does not exist in the specified tuple.

Code:

```
1. Nordic_Country_Tuple = ("FINLAND", "ICELAND", "NORWAY", "DENMARK",
"SWEDEN", "FAROE ISLANDS")
2.
3. print("FINLAND" in Nordic_Country_Tuple)
4.
5. print("ICELAND" in Nordic_Country_Tuple)
6.
7. print("FAROE ISLANDS" not in Nordic_Country_Tuple)
8.
9. print("DENMARK" not in Nordic_Country_Tuple)
```

Output:

```
1.  True
2.  True
3.  False
4.  False
```

Working with Modules (1/5)

What is a module?

A module is a Python script file, that consists of functions, classes, and data (variables or immutable data). At, current moment forget about classes and just focus on functions and data, that we have already discussed.

Why do we need modules?

It is cumbersome to write a large number of lines of codes in a single file. Also, when we like to use functions written by other programmers we need to import their files. In order to group related code and data, we simply write related codes in separate Python script.

Based on our current Python script requirement, we simply import individual functions or complete modules (Python script file), to use in our program.

How to import the module in our Python Script?

We would like to explain things to you from scratch. So, instead of importing the existing module, we will build a module/Python script and then we will import in another Python Program or Script.

Working with Modules (2/5)

Creating a calculator.py Module

The below code is an aggregation of four different functions, that provide basic operations of a calculator such as addition, multiplication, subtraction, division

Code:

```
def add(a, b):
    c = a + b
    return c;


def sub(a, b):
    c = a - b
    return c;


def mul(a, b):
    c = a * b
    return c;


def div(a, b):
    c = a / b
    return c;
```

Working with Modules (3/5)

How to import module in a Python Script/Program

- In order to import a Python module, the module must be placed in the current directory or the default path where python is installed.
- Now there are two methods to access the functions in the module with or without a dot. operator.

Without Using dot operator

- If you want to import all the functions from calculator.py in your current program use
  - from calculator import *
- If you want to import specific function from caluclator.py in your current program use
  - from calculator import add

Code:

```
#import all the functions from calculator.py
from calculator import *

print(add(3, 4))
print(mul(3, 4))
print(sub(3, 4))
print(div(3, 4))
```

Output:

```
7
12
-1
0.75
```

Working with Modules (4/5)

Code:

```
#import specific function from caluclator.py
from calculator import add

print(add(91, 909))

#We will get error if we will use any other function
print(mul(91, 909))
```

Output:

```
1000
Traceback (most recent call last):
  File "practise.py", line 7, in <module>
    print(mul(91, 909))
NameError: name 'mul' is not defined
```

Working with Modules (5/5)

Using dot operator

- If you want to import all the functions from calculator.py in your current program use
  - import calculator

Code:

```
#import all function from caluclator.py
import calculator

#We need to access function using the dot operator
print(calculator.add(91, 909))

#We need to access function using the dot operator
print(calculator.mul(91, 909))
```

Output:

```
1000
82719
```

Working With Classes (1/7)

What is a class?

We all have heard about Object-Oriented Programming. Implementation of a Class in any programming language helps us to achieve the concept of the OOPs. A class consists of functions and data, which works as a blueprint, whenever we need to use the associated function and data in a given class, we create an object, which contains its local copy of function and data. So, we are able to access the function and data using **object_name** and dot operator.

Why Class is needed when the concept of function is already there?

There are many important concepts we realize through the implementation of classes such as Data Abstraction, Data Hiding, Inheritance, Polymorphism. You need to study Object-oriented programming if you want to go in detail.

One important fundamental: Whenever we import a module, which does not contain any class i.e. a simple python script with function and data. We import the function itself, so any other python program that wants to execute this module at the same time can produce unexpected results. Because two python programs will simultaneously share the resources of the given module.

But in the case of Class, we are free to do so, because class gives access to its function and data only through an object. And each object has its own local copy of function and data. Though some data can be shared between classes if we want to, which is further implementation of the OOPs concept.

Working With Classes (2/7)

Writing a class:

Simply imagine you are writing a single function or group of functions using data.

PseudoCode:

```
class class_name(object)


    def __init__(self, arg1, arg2, arg3, .., .., argN):
        self.arg1 = arg1
        self.arg2 = arg2
        self.arg3 = arg3
        self.argN = argN

    def function_name_1(self):
        print("I AM {} Color".format(self.arg1))

    def function_name_2(self):
        print("I AM {} Color".format(self.arg2))

    def function_name_3(self):
        print("I AM {} Color".format(self.arg3))

    def function_name_N(self):
        print("I AM {} Color".format(self.argN))
```

Please note: In the above code **class, object, self, def, __init__** are reserved keyword and you need to remember these keywords for writing a class.

class_name, function_name_1, function_name_2, function_name_3, function_name_4 can be named anything (based on your choice) in compliance with permitted namespace in Python.

What is the use of self keyword: In the above code you can see self keyword repeating a lot of times. We are creating an object that contains a local copy of data variables, thus we use the self keyword where each object refers to its own data and associated function.

Working With Classes (3/7)

How to access it:

We need to create an object for a given class. When creating an object we can pass arguments to initialize data. Then we will access all the data and function using:

PseudoCode:

```
#Creating an Object & initializing with N arguments
first_object = class_name(arg1, arg2, arg3, .., .., argN)

#Calling function_name_1
first_object.function_name_1()

#Calling function_name_2
first_object.function_name_2()

#Calling function_name_3
first_object.function_name_3()

#Calling function_name_N
first_object.function_name_N()
```

Working With Classes (4/7)

Implementation of Calculator Python Program using Class:

Initializing Parameters while creating an object

Code:

```
class Calculator(object):

    def __init__(self, a, b):
        self.a = a
        self.b = b

    def add(self):
        return self.a + self.b

    def sub(self):
        return self.a - self.b

    def mul(self):
        return self.a * self.b

    def div(self):
        return self.a / self.b


calc_obj = Calculator(99, 3)
print("Addition result is {}".format(calc_obj.add()))
print("Multiplication  result is {}".format(calc_obj.mul()))
print("Division  result is {}".format(calc_obj.div()))
print("Subtraction  result is {}".format(calc_obj.sub()))
```

Output:

```
Addition result is 102
Multiplication  result is 297
Division  result is 33.0
Subtraction  result is 96
```

Encapsulation, IS-A (Inheritance), HAS-A (Composition)

Encapsulation

Encapsulation: When we want to hide some data from other objects or we want to restrict access to some methods and variables. The concept of Encapsulation helps us to prevent direct manipulation or modification of data. Only, the function has access to private data. We hide attributes by making them private. In Python, we hide it by placing underscore as the prefix for the attribute, i.e single " _ " or double " __"

In the below code, we have demonstrated that private class data can only be manipulated using class functions and not using an object.

Code:

```
class Color:

    def __init__(self):
        self.__color_name = "Green"

    def Color_Type(self):
        print("Color Type is : {}".format(self.__color_name))

    def Color_Type_1(self, price):
         self.__color_name = "Red"

c = Color()
c.Color_Type()

# changing the Color
c.__color_name = "Blue"
c.Color_Type()

# Setting color using Function
c.Color_Type_1(1000)
c.Color_Type()
```

Output:

```
Color Type is : Green
Color Type is : Green
Color Type is : Red
```

Working With Classes (5/7)

IS-A Relationship (Inheritance)

IS-A Relationship: Salmon is a Fish. It means Salmon inherit properties from a Fish. When a child class inherits the property of a parent class. We refer to it as a Parent-Child relationship.

There are three ways in which parent and child class can interact:

- **Actions on the child imply an action on the parent: Implicit behavior**
- **Actions on the child override the action on the parent: Override Explicitly**
- **Actions on the child alter the action on the parent: Alter Before or After**

Actions on the child imply an action on the parent: Implicit behavior

Implicit actions that happen when you define a function in the parent but not in the child.

Code:

```
class Parent(object):
        def function_parent(self):
            print("I am parent")


class Child(Parent):
        pass

a = Parent()
b = Child()

a.function_parent()
b.function_parent()
```

Output:

```
I am parent
I am parent
```

Actions on the child override the action on the parent: Override Explicitly

Code:

```
class Parent(object):
        def function(self):
            print("I am parent")


class Child(Parent):
    def function(self):
        print("I am Child")

a = Parent()
b = Child()

a.function()
b.function()
```

Output:

```
I am parent
I am Child
```

Working With Classes (6/7)

Actions on the child alter the action on the parent: Alter Before or After

The third way to use inheritance is a special case of overriding where you want to alter the behavior before or after the Parent class's version runs. You first override the function just like in the last example, but now we will use a Python built-in function named **super** to get the Parent version to call. Here's the example of doing that so you can make sense of this description:

Code:

```
class Parent(object):
        def function(self):
            print("I am parent")


class Child(Parent):
    def function(self):
        print("I am CHILD, BEFORE PARENT is called")
        super(Child, self).function()
         print("I am CHILD, AFTER PARENT is called")


a = Parent()
b = Child()

a.function()
b.function()
```

Output:

```
I am parent
I am CHILD, BEFORE PARENT is called
I am parent
I am CHILD, AFTER PARENT is called
```

Demonstration of super for initialization and function calling from Child class

Using super() with __init__

Code:

```
class Cube(object):

        def __init__(self, arg):
            self.arg = arg

        def CubeFunction(self):
            self.CubeResult = self.arg * self.arg * self.arg


class CubeAdd(Cube):

         def __init__(self, arg, arg1):
             super().__init__(arg)
             self.arg1 = arg1
         def CubeAddFunction(self):
             super(CubeAdd, self).CubeFunction()
             self.CubeAddResult = self.CubeResult + self.arg1



b = CubeAdd(2,1)
b.CubeAddFunction()
print(b.CubeAddResult)

b = CubeAdd(3,1)
b.CubeAddFunction()
print(b.CubeAddResult)

b = CubeAdd(4,1)
b.CubeAddFunction()
print(b.CubeAddResult)
```

Output:

```
9
28
65
```

Working With Classes (7/7)

HAS-A Relationship (Composition)

HAS-A Relationship: A dog is a mammal and Parrot is a Bird. Dog inherits features from Mammal parent class and Parrot inherits features from Bird parent class. Sometimes, one class does not inherit all the features from a class. But both dog and mammal have a mouth. It means they have some common properties but do not inherit all. So, we achieve this functionality through composition, we directly call a function from another class, without inheriting all the featured from a class.

The following code demonstrate the use of Composition.

Code:

```
class Cube(object):

        def __init__(self, arg):
            self.arg = arg

        def CubeFunction(self):
            self.CubeResult = self.arg * self.arg * self.arg


class CubeAdd(Cube):

         def __init__(self, arg, arg1):
             self.arg1 = arg1
             self.cube = Cube(arg)

         def CubeAddFunction(self):
             self.cube.CubeFunction()
             self.CubeAddResult = self.cube.CubeResult + self.arg1


b = CubeAdd(2,1)
b.CubeAddFunction()
print(b.CubeAddResult)

b = CubeAdd(3,1)
b.CubeAddFunction()
print(b.CubeAddResult)

b = CubeAdd(4,1)
b.CubeAddFunction()
print(b.CubeAddResult)
```

Output:

```
9
28
65
```

Βασικές βιβλιοθήκες

NumPy is used for working with arrays.

NumPy

```
import numpy

arr = numpy.array([1, 2, 3, 4, 5])

print(arr)
```

Create a 0-D array with value 42

```
import numpy as np

arr = np.array(42)

print(arr)
```

Create a 1-D array containing the values 1,2,3,4,5:

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5])

print(arr)
```

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5])

print(arr)

print(type(arr))
```

Create a 2-D array containing two arrays with the values 1,2,3 and 4,5,6:

```
import numpy as np

arr = np.array([[1, 2, 3], [4, 5, 6]])

print(arr)
```

Use a tuple to create a NumPy array:

```
import numpy as np

arr = np.array((1, 2, 3, 4, 5))

print(arr)
```

Create a 3-D array with two 2-D arrays, both containing two arrays with the values 1,2,3 and 4,5,6:

```
import numpy as np

arr = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])

print(arr)
```

NumPy

Get the first element from the following array:

```
import numpy as np

arr = np.array([1, 2, 3, 4])

print(arr[0])
```

Get the second element from the following array.

```
import numpy as np

arr = np.array([1, 2, 3, 4])

print(arr[1])
```

Get third and fourth elements from the following array and add them.

```
import numpy as np

arr = np.array([1, 2, 3, 4])

print(arr[2] + arr[3])
```

Access the 2nd element on 1st dim:

```
import numpy as np

arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])

print('2nd element on 1st dim: ', arr[0, 1])
```

Access the 5th element on 2nd dim:

```
import numpy as np

arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])

print('5th element on 2nd dim: ', arr[1, 4])
```

Access the third element of the second array of the first array:

```
import numpy as np

arr = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]])

print(arr[0, 1, 2])
```

NumPy

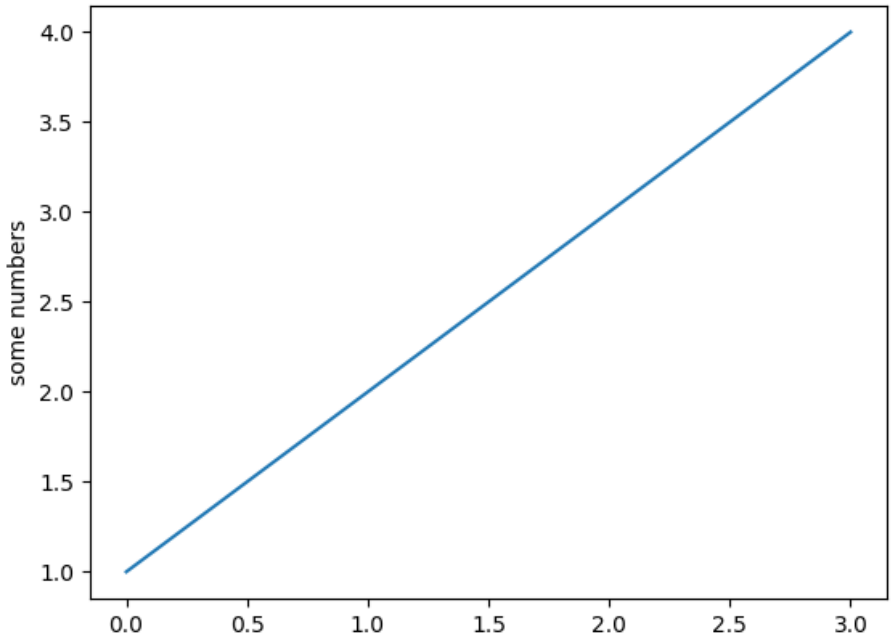
https://numpy.org/

https://www.w3schools.com/python/numpy_intro.asp

Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python.

matplotlib

```
import matplotlib.pyplot as plt
plt.plot([1, 2, 3, 4])
plt.ylabel('some numbers')
plt.show()
```

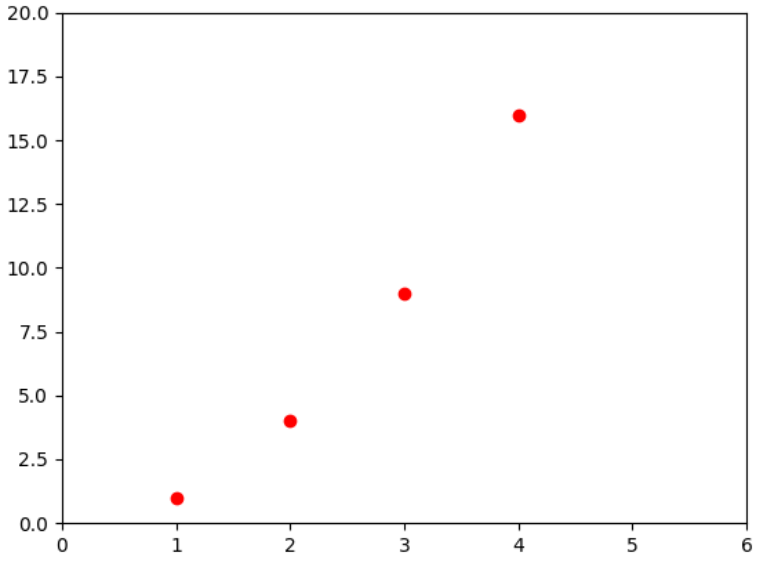


matplotlib

Formatting the style of your plot

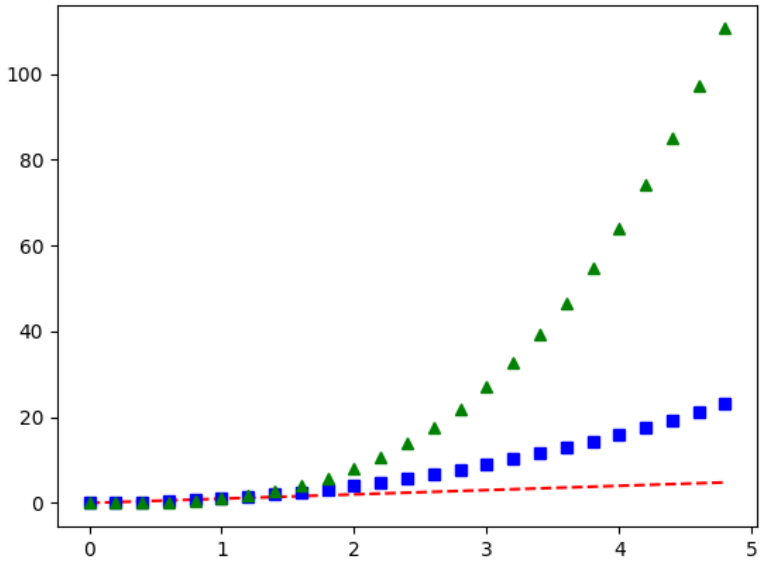
For every x, y pair of arguments, there is an optional third argument which is the format string that indicates the color and line type of the plot. The letters and symbols of the format string are from MATLAB, and you concatenate a color string with a line style string. The default format string is 'b-', which is a solid blue line. For example, to plot the above with red circles, you would issue

```
plt.plot([1, 2, 3, 4], [1, 4, 9, 16], 'ro')
plt.axis([0, 6, 0, 20])
plt.show()
```



matplotlib

```
import numpy as np

# evenly sampled time at 200ms intervals
t = np.arange(0., 5., 0.2)

# red dashes, blue squares and green triangles
plt.plot(t, t, 'r--', t, t**2, 'bs', t, t**3, 'g^')
plt.show()
```



matplotlib

https://matplotlib.org/index.html

https://www.datacamp.com/community/tutorials/matplotlib-tutorial-python

https://realpython.com/python-matplotlib-guide/

SciPy library

Fundamental library for scientific computing

It provides numerical routines, such as routines for numerical integration, interpolation, optimization, linear algebra, and statistics.

SciPy

Numpy VS SciPy

Numpy:

- Numpy is written in C and use for mathematical or numeric calculation.
- It is faster than other Python Libraries
- Numpy is the most useful library for Data Science to perform basic calculations.
- Numpy contains nothing but array data type which performs the most basic operation like sorting, shaping, indexing, etc.

SciPy:

- SciPy is built in top of the NumPy
- SciPy is a fully-featured version of Linear Algebra while Numpy contains only a few features.
- Most new Data Science features are available in Scipy rather than Numpy.

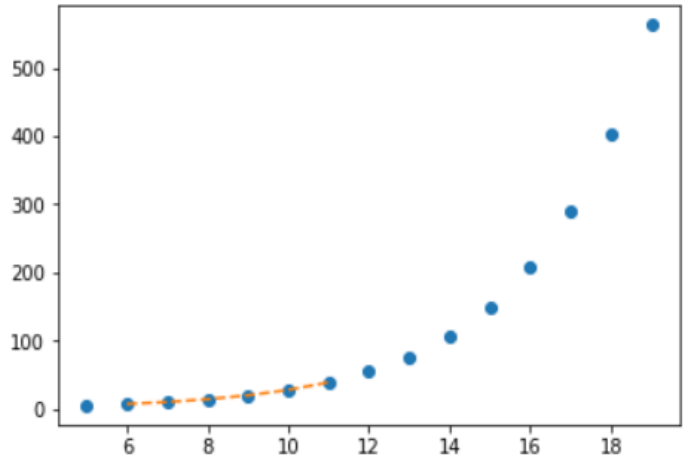
- Interpolation - scipy.interpolate
- Optimization and fit - scipy.optimize
- Statistics and random numbers - scipy.stats
- Numerical Integration - scipy.integrate
- Fast Fourier transforms - scipy.fftpack
- Signal Processing - scipy.signal
- Image manipulation – scipy.ndimage

SciPy

EXAMPLE:

```
import matplotlib.pyplot as plt
from scipy import interpolate
x = np.arange(5, 20)
y = np.exp(x/3.0)
f = interpolate.interp1d(x, y)x1 = np.arange(6, 12)
y1 = f(x1)   # use interpolation function returned by `interp1d`
plt.plot(x, y, 'o', x1, y1, '--')
plt.show()
```

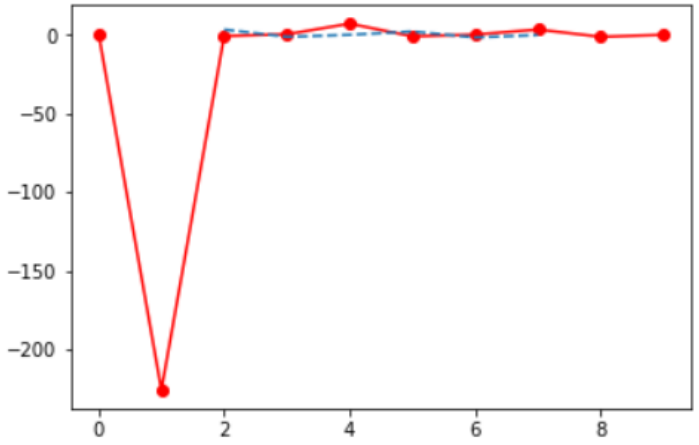
OUTPUT:



EXAMPLE:

```
from scipy import interpolate
import matplotlib.pyplot as plt
x = np.arange(0,10)
y = np.arange(10,25)
x1, y1 = np.meshgrid(x, y)
z = np.tan(xx+yy)
f = interpolate.interp2d(x, y, z, kind='cubic')
x2 = np.arange(2,8)
y2 = np.arange(15,20)
z2 = f(xnew, ynew)
plt.plot(x, z[0, :], 'ro-', x2, z2[0, :], '--')
plt.show()
```

OUTPUT:



SciPy

https://docs.scipy.org/doc/

https://www.guru99.com/scipy-tutorial.html

https://www.edureka.co/blog/scipy-tutorial/

It is a fast, powerful, flexible and easy to use open source data analysis and manipulation tool.

pandas

script.py

```
1  dict = {"country": ["Brazil", "Russia", "India", "China",
   "South Africa"],
2         "capital": ["Brasilia", "Moscow", "New Dehli",
   "Beijing", "Pretoria"],
3         "area": [8.516, 17.10, 3.286, 9.597, 1.221],
4         "population": [200.4, 143.5, 1252, 1357, 52.98] }
5
6  import pandas as pd
7  brics = pd.DataFrame(dict)
8  print(brics)
```

IPython Shell

```
    area    capital       country  population
0   8.516   Brasilia        Brazil      200.40
1  17.100     Moscow        Russia      143.50
2   3.286  New Dehli         India     1252.00
3   9.597    Beijing         China     1357.00
4   1.221   Pretoria  South Africa       52.98

In [1]:
```

pandas

Creating DataFrame from dict of ndarray/lists: To create DataFrame from dict of narray/list, all the narray must be of same length. If index is passed then the length index should be equal to the length of arrays. If no index is passed, then by default, index will be range(n) where n is the array length.

```
# Python code demonstrate creating
# DataFrame from dict narray / lists
# By default addresses.

import pandas as pd

# intialise data of lists.
data = {'Name':['Tom', 'nick', 'krish', 'jack'],
        'Age':[20, 21, 19, 18]}

# Create DataFrame
df = pd.DataFrame(data)

# Print the output.
print(df)
```

Output:

	Name	Age
0	Tom	20
1	nick	21
2	krish	19
3	jack	18

pandas

Dealing with Rows and Columns

A Data frame is a two-dimensional data structure, i.e., data is aligned in a tabular fashion in rows and columns. We can perform basic operations on rows/columns like selecting, deleting, adding, and renaming.

Column Selection: In Order to select a column in Pandas DataFrame, we can either access the columns by calling them by their columns name.

```
# Import pandas package
import pandas as pd

# Define a dictionary containing employee data
data = {'Name':['Jai', 'Princi', 'Gaurav', 'Anuj'],
        'Age':[27, 24, 22, 32],
        'Address':['Delhi', 'Kanpur', 'Allahabad', 'Kannauj'],
        'Qualification':['Msc', 'MA', 'MCA', 'Phd']}

# Convert the dictionary into DataFrame
df = pd.DataFrame(data)

# select two columns
print(df[['Name', 'Qualification']])
```

Output:

	Name	Qualification
0	Jai	Msc
1	Princi	MA
2	Gaurav	MCA
3	Anuj	Phd

Python For Data Science *Cheat Sheet*

Pandas Basics

Learn Python for Data Science Interactively at www.DataCamp.com

Pandas

The **Pandas** library is built on NumPy and provides easy-to-use **data structures** and **data analysis** tools for the Python programming language.

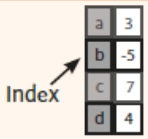
Use the following import convention:

```
>>> import pandas as pd
```

Pandas Data Structures

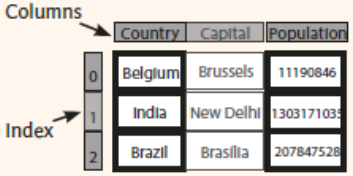
Series

A **one-dimensional** labeled array capable of holding any data type




```
>>> s = pd.Series([3, -5, 7, 4], index=['a', 'b', 'c', 'd'])
```

DataFrame



A **two-dimensional** labeled data structure with columns of potentially different types

```
>>> data = {'Country': ['Belgium', 'India', 'Brazil'],
            'Capital': ['Brussels', 'New Delhi', 'Brasília'],
            'Population': [11190846, 1303171035, 207847528]}

>>> df = pd.DataFrame(data,
                      columns=['Country', 'Capital', 'Population'])
```

Asking For Help

```
>>> help(pd.Series.loc)
```

Selection

Also see NumPy Arrays

Getting

```
>>> s['b']
 -5

>>> df[1:]
   Country     Capital  Population
1   India  New Delhi  1303171035
2  Brazil   Brasília   207847528
```

Get one element

Get subset of a DataFrame

Selecting, Boolean Indexing & Setting

By Position

```
>>> df.iloc[[0],[0]]
'Belgium'
>>> df.iat([0],[0])
'Belgium'
```

Select single value by row & column

By Label

```
>>> df.loc[[0], ['Country']]
'Belgium'
>>> df.at([0], ['Country'])
'Belgium'
```

Select single value by row & column labels

By Label/Position

```
>>> df.ix[2]
Country      Brazil
Capital    Brasília
Population  207847528
```

Select single row of subset of rows

```
>>> df.ix[:,'Capital']
0     Brussels
1    New Delhi
2     Brasília
```

Select a single column of subset of columns

```
>>> df.ix[1,'Capital']
'New Delhi'
```

Select rows and columns

Boolean Indexing

```
>>> s[~(s > 1)]
>>> s[(s < -1)  |  (s > 2)]
>>> df[df['Population']>1200000000]
```

Series s where value is not >1
s where value is <-1 or >2
Use filter to adjust DataFrame

Setting

```
>>> s['a'] = 6
```

Set index a of Series s to 6

Dropping

```
>>> s.drop(['a', 'c'])
>>> df.drop('Country', axis=1)
```

Drop values from rows
Drop values from colu

Sort & Rank

```
>>> df.sort_index()
>>> df.sort_values(by='Country')
>>> df.rank()
```

Sort by labels along
Sort by the values
Assign ranks to ent

Retrieving Series/DataFrame Informatio

Basic Information

```
>>> df.shape
>>> df.index
>>> df.columns
>>> df.info()
>>> df.count()
```

(rows,columns)
Describe index
Describe DataFrame colum
Info on DataFrame
Number of non-NA values

Summary

```
>>> df.sum()
>>> df.cumsum()
>>> df.min()/df.max()
>>> df.idxmin()/df.idxmax()
>>> df.describe()
>>> df.mean()
>>> df.median()
```

Sum of values
Cummulative sum of v
Minimum/maximum v
Minimum/Maximum in
Summary statistics
Mean of values
Median of values

Applying Functions

```
>>> f = lambda x: x*2
>>> df.apply(f)
>>> df.applymap(f)
```

Apply function
Apply function element-wis

Data Alignment

Internal Data Alignment

NA values are introduced in the indices that don't ov

```
>>> s3 = pd.Series([7, -2, 3], index=['a', 'c'
>>> s + s3
a    10.0
b     NaN
c     5.0
d     7.0
```

Arithmetic Operations with Fill Methods

You can also do the Internal data alignment yourself the help of the fill methods:

```
>>> s.add(s3, fill_value=0)
a    10.0
b    -5.0
c     5.0
d     7.0
>>> s.sub(s3, fill_value=2)
>>> s.div(s3, fill_value=4)
>>> s.mul(s3, fill_value=3)
```

I/O

Read and Write to CSV

```
>>> pd.read_csv('file.csv', header=None, nrows=5)
>>> df.to_csv('myDataFrame.csv')
```

Read and Write to Excel

```
>>> pd.read_excel('file.xlsx')
>>> df.to_excel('dir/myDataFrame.xlsx', sheet_name='Sheet1')
```

Read multiple sheets from the same file

```
>>> xlsx = pd.ExcelFile('file.xls')
>>> df = pd.read_excel(xlsx, 'Sheet1')
```

Read and Write to SQL Query or Database Table

```
>>> from sqlalchemy import create_engine
>>> engine = create_engine('sqlite:///:memory:')
>>> pd.read_sql("SELECT * FROM my_table;", engine)
>>> pd.read_sql_table('my_table', engine)
>>> pd.read_sql_query("SELECT * FROM my_table;", engine)
```

`read_sql()` is a convenience wrapper around `read_sql_table()` and `read_sql_query()`

```
>>> df.to_sql('myDf', engine)
```

DataCamp
Learn Python for Data Science Interactively

pandas

https://pandas.pydata.org/

https://towardsdatascience.com/a-quick-introduction-to-the-pandas-python-library-f1b678f34673

https://www.learnpython.org/en/Pandas_Basics

https://www.geeksforgeeks.org/python-pandas-dataframe/

https://www.dataquest.io/blog/pandas-cheat-sheet/

https://pandas.pydata.org/Pandas_Cheat_Sheet.pdf

https://www.datacamp.com/community/blog/python-pandas-cheat-sheet

1. Supervised learning
2. Unsupervised learning
3. Model selection and evaluation
4. Inspection
5. Visualizations
6. Dataset transformations
7. Dataset loading utilities
8. Computing with scikit-learn

It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, k-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.

Python For Data Science *Cheat Sheet*

Scikit-Learn

Learn Python for data science Interactively at www.DataCamp.com

Scikit-learn

Scikit-learn is an open source Python library that implements a range of machine learning, preprocessing, cross-validation and visualization algorithms using a unified interface.

A Basic Example

```
>>> from sklearn import neighbors, datasets, preprocessing
>>> from sklearn.model_selection import train_test_split
>>> from sklearn.metrics import accuracy_score
>>> iris = datasets.load_iris()
>>> X, y = iris.data[:, :2], iris.target
>>> X_train, X_test, y_train, y_test= train_test_split(X, y, random_state=33)
>>> scaler = preprocessing.StandardScaler().fit(X_train)
>>> X_train = scaler.transform(X_train)
>>> X_test = scaler.transform(X_test)
>>> knn = neighbors.KNeighborsClassifier(n_neighbors=5)
>>> knn.fit(X_train, y_train)
>>> y_pred = knn.predict(X_test)
>>> accuracy_score(y_test, y_pred)
```

Loading The Data

Also see NumPy & Pandas

Your data needs to be numeric and stored as NumPy arrays or SciPy sparse matrices. Other types that are convertible to numeric arrays, such as Pandas DataFrame, are also acceptable.

```
>>> import numpy as np
>>> X = np.random.random((10,5))
>>> y = np.array(['M','M','F','F','M','F','M','M','F','F','F'])
>>> X[X < 0.7] = 0
```

Training And Test Data

```
>>> from sklearn.model_selection import train_test_split
>>> X_train, X_test, y_train, y_test = train_test_split(X,
                                                        y,
                                                        random_state=0)
```

Create Your Model

Supervised Learning Estimators

Linear Regression

```
>>> from sklearn.linear_model import LinearRegression
>>> lr = LinearRegression(normalize=True)
```

Support Vector Machines (SVM)

```
>>> from sklearn.svm import SVC
>>> svc = SVC(kernel='linear')
```

Naive Bayes

```
>>> from sklearn.naive_bayes import GaussianNB
>>> gnb = GaussianNB()
```

KNN

```
>>> from sklearn import neighbors
>>> knn = neighbors.KNeighborsClassifier(n_neighbors=5)
```

Unsupervised Learning Estimators

Principal Component Analysis (PCA)

```
>>> from sklearn.decomposition import PCA
>>> pca = PCA(n_components=0.95)
```

K Means

```
>>> from sklearn.cluster import KMeans
>>> k_means = KMeans(n_clusters=3, random_state=0)
```

Model Fitting

Supervised learning	
`>>> lr.fit(X, y)`	Fit the model to the data
`>>> knn.fit(X_train, y_train)`	
`>>> svc.fit(X_train, y_train)`	
Unsupervised Learning	
`>>> k_means.fit(X_train)`	Fit the model to the data
`>>> pca_model = pca.fit_transform(X_train)`	Fit to data, then transform it

Prediction

Supervised Estimators	
`>>> y_pred = svc.predict(np.random.random((2,5)))`	Predict labels
`>>> y_pred = lr.predict(X_test)`	Predict labels
`>>> y_pred = knn.predict_proba(X_test)`	Estimate probability of a label
Unsupervised Estimators	
`>>> y_pred = k_means.predict(X_test)`	Predict labels in clustering algos

Preprocessing The Data

Standardization

```
>>> from sklearn.preprocessing import StandardScaler
>>> scaler = StandardScaler().fit(X_train)
>>> standardized_X = scaler.transform(X_train)
>>> standardized_X_test = scaler.transform(X_test)
```

Normalization

```
>>> from sklearn.preprocessing import Normalizer
>>> scaler = Normalizer().fit(X_train)
>>> normalized_X = scaler.transform(X_train)
>>> normalized_X_test = scaler.transform(X_test)
```

Binarization

```
>>> from sklearn.preprocessing import Binarizer
>>> binarizer = Binarizer(threshold=0.0).fit(X)
>>> binary_X = binarizer.transform(X)
```

Encoding Categorical Features

```
>>> from sklearn.preprocessing import LabelEncoder
>>> enc = LabelEncoder()
>>> y = enc.fit_transform(y)
```

Imputing Missing Values

```
>>> from sklearn.preprocessing import Imputer
>>> imp = Imputer(missing_values=0, strategy='mean', axis=0)
>>> imp.fit_transform(X_train)
```

Generating Polynomial Features

```
>>> from sklearn.preprocessing import PolynomialFeatures
>>> poly = PolynomialFeatures(5)
>>> poly.fit_transform(X)
```

Evaluate Your Model's Performance

Classification Metrics

Accuracy Score

```
>>> knn.score(X_test, y_test)
```
Estimator score method

```
>>> from sklearn.metrics import accuracy_score
>>> accuracy_score(y_test, y_pred)
```
Metric scoring functions

Classification Report

```
>>> from sklearn.metrics import classification_report
>>> print(classification_report(y_test, y_pred))
```
Precision, recall, f1-score and support

Confusion Matrix

```
>>> from sklearn.metrics import confusion_matrix
>>> print(confusion_matrix(y_test, y_pred))
```

Regression Metrics

Mean Absolute Error

```
>>> from sklearn.metrics import mean_absolute_error
>>> y_true = [3, -0.5, 2]
>>> mean_absolute_error(y_true, y_pred)
```

Mean Squared Error

```
>>> from sklearn.metrics import mean_squared_error
>>> mean_squared_error(y_test, y_pred)
```

R^2 Score

```
>>> from sklearn.metrics import r2_score
>>> r2_score(y_true, y_pred)
```

Clustering Metrics

Adjusted Rand Index

```
>>> from sklearn.metrics import adjusted_rand_score
>>> adjusted_rand_score(y_true, y_pred)
```

Homogeneity

```
>>> from sklearn.metrics import homogeneity_score
>>> homogeneity_score(y_true, y_pred)
```

V-measure

```
>>> from sklearn.metrics import v_measure_score
>>> metrics.v_measure_score(y_true, y_pred)
```

Cross-Validation

```
>>> from sklearn.cross_validation import cross_val_score
>>> print(cross_val_score(knn, X_train, y_train, cv=4))
>>> print(cross_val_score(lr, X, y, cv=2))
```

Tune Your Model

Grid Search

```
>>> from sklearn.grid_search import GridSearchCV
>>> params = {"n_neighbors": np.arange(1,3),
              "metric": ["euclidean", "cityblock"]}
>>> grid = GridSearchCV(estimator=knn,
                        param_grid=params)
>>> grid.fit(X_train, y_train)
>>> print(grid.best_score_)
>>> print(grid.best_estimator_.n_neighbors)
```

Randomized Parameter Optimization

```
>>> from sklearn.grid_search import RandomizedSearchCV
>>> params = {"n_neighbors": range(1,5),
              "weights": ["uniform", "distance"]}
>>> rsearch = RandomizedSearchCV(estimator=knn,
                                 param_distributions=params,
                                 cv=4,
                                 n_iter=8,
                                 random_state=5)
>>> rsearch.fit(X_train, y_train)
>>> print(rsearch.best_score_)
```

DataCamp
Learn Python for Data Science Interactively

scikit-learn

https://scikit-learn.org/stable/

https://appdividend.com/2019/02/01/python-scikit-learn-tutorial-for-beginners-with-example/

https://dev.to/duomly/what-is-scikit-learn-a-beginner-guide-to-popular-machine-learning-python-library-1f06

https://www.dataquest.io/blog/sci-kit-learn-tutorial/

SciPy is a Python-based ecosystem of open-source software for mathematics, science, and engineering.

These are some of the core packages:

NumPy
Base N-dimensional array package

SciPy library
Fundamental library for scientific computing

Matplotlib
Comprehensive 2-D plotting

IP[y]: IPython

IPython
Enhanced interactive console

SymPy
Symbolic mathematics

pandas
Data structures & analysis

Deep learning

Περισσότερα

Examples

https://www.tutorialgateway.org/python-programming-examples/

https://www.w3schools.com/python/python_examples.asp

https://www.sanfoundry.com/python-problems-solutions/

https://www.pythonforbeginners.com/code-snippets-source-code/python-code-examples

https://www.geeksforgeeks.org/python-programming-examples/

Further reading

https://docs.python.org/3/tutorial/

https://www.learnpython.org/

https://www.w3schools.com/python/default.asp

https://www.pythonforbeginners.com/

https://pythonbaba.com/string-literals-in-python/

https://towardsdatascience.com/tagged/python