accenture technology

Learning and Knowledge Management

Module 2: Python Basics

Module Objectives

At the end of this module, you will be able to:

- Illustrate Python installation
- Demonstrate adding Pydev Plugin in Eclipse IDE
- Explain how to create Python projects
- Describe implementing object orientation concepts and structres in Python programs
- Use Command Line Arguments
- Apply Exceptions and Assertions
- Describe the various String Functions

Topic List

- Python Programming Concepts
- Python Installation
- PyDev Installation in Eclipse
- Getting Started with Python as a Programming Language
- OOP Features of Python
- Data Types and Variables
- Operators and Keywords

Topic List

- Conditional and Looping statements
- Functions and Scope of Variables
- Defining Class and Objects
- Command Line Arguments
- Exceptions And Assertions
- Handling Strings

Topic List

Python Programming Concepts

Features of Python

- High-level ,object-oriented that uses English keywords with easy, and minimal syntax constructs
- Free source and extensively supports almost any platform such as Windows, Mac OS, Linux etc. and databases
- Scalable , extendable which enables programmers to customize their tools to increase efficiency
- Provides extensive built in library that can be easily integrated with web applications
- Enables easy testing and debugging

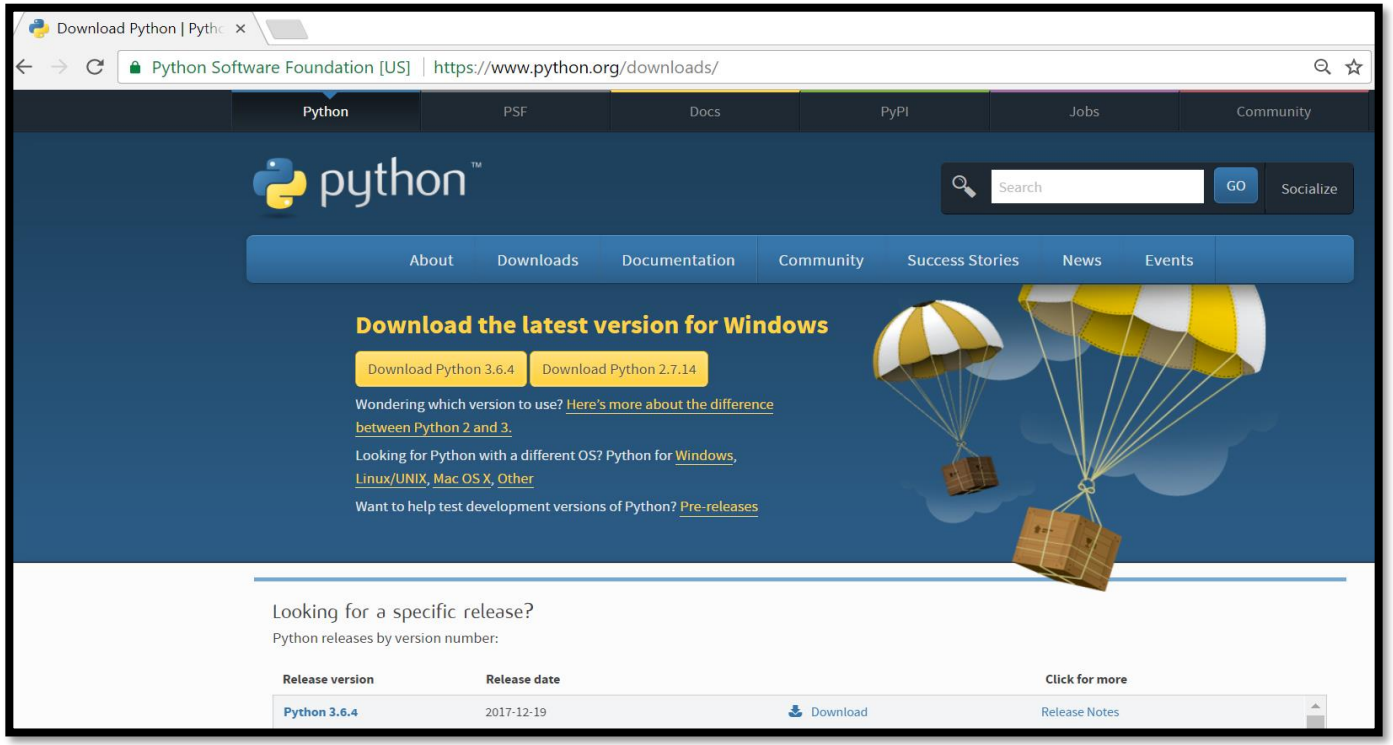
Topic List

Python Installation(1)

How to install Python?

Follow below mentioned steps to install Python

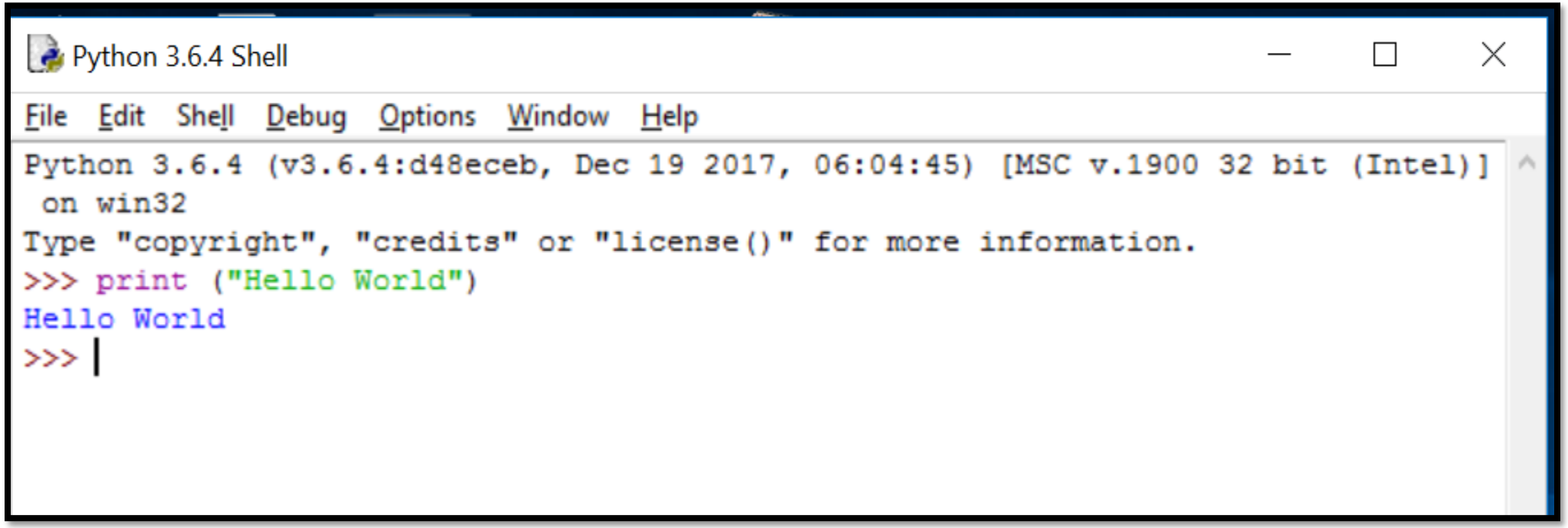
- Visit https://www.python.org/downloads/
- Download (3.6.4) version of Python



Python Installation(2)

Steps to Install Python

- After installing Python, you can see Python menu choice
- Python programming can be done either in **command line** or in **GUI mode**(IDLE)
- Open IDLE and type simple command to print => “Hello World”



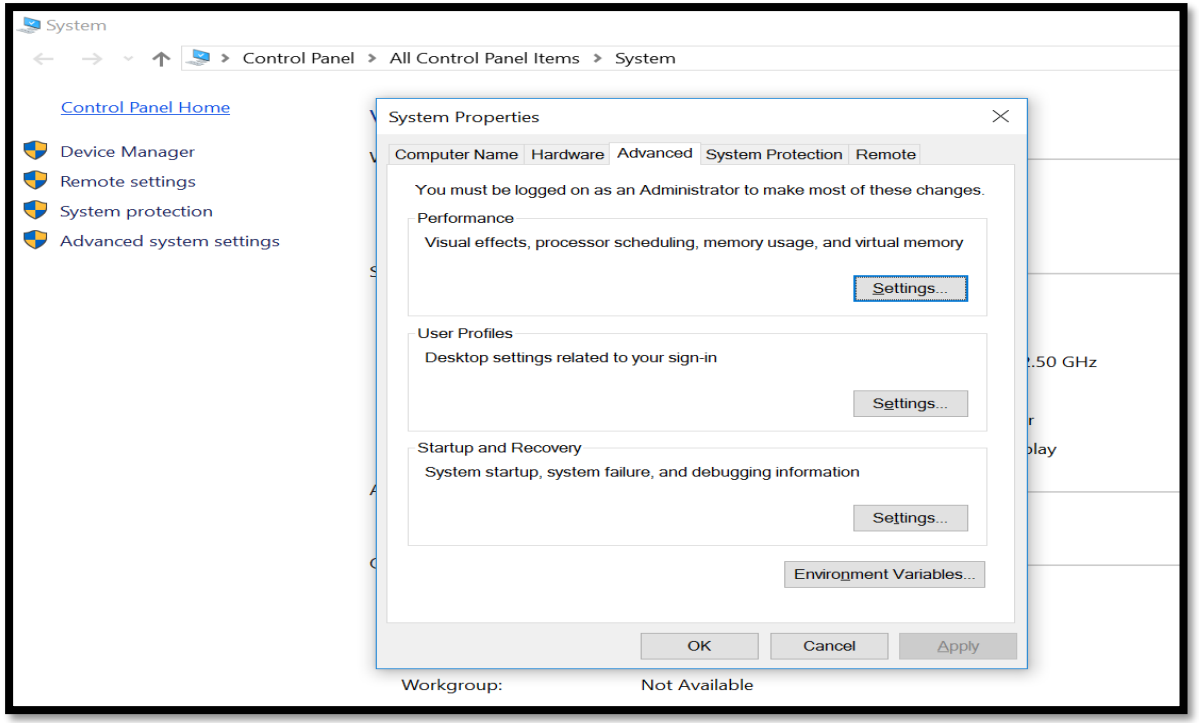
Python Installation(3)

Steps to Install Python

- We also need to add Python Path to Environment Variables

Here is the process to set the Python path:

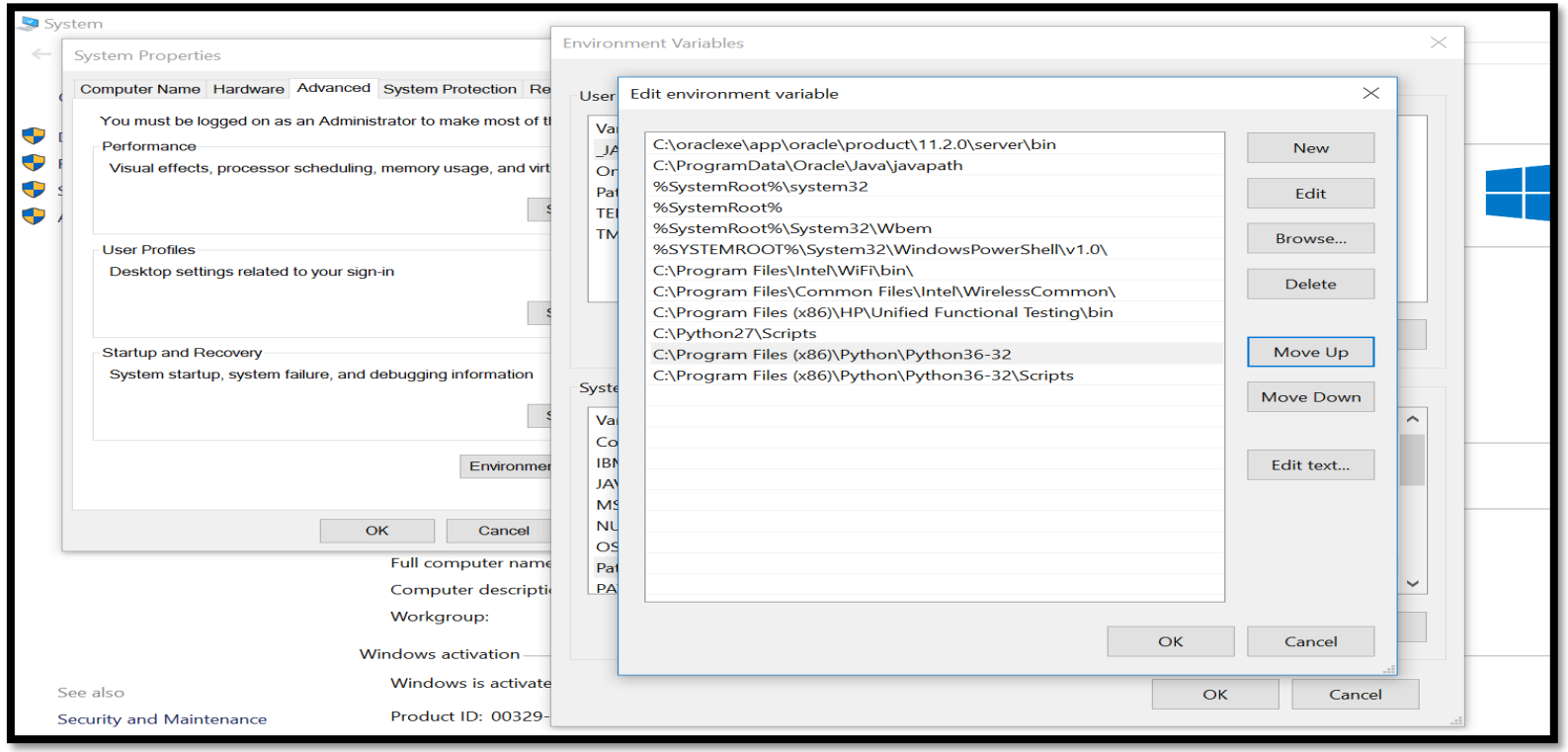
In windows10, go to **Control Panel-> Advance System settings – >Advanced -> Environment Variables**



Python Installation(4)

Steps to Install Python

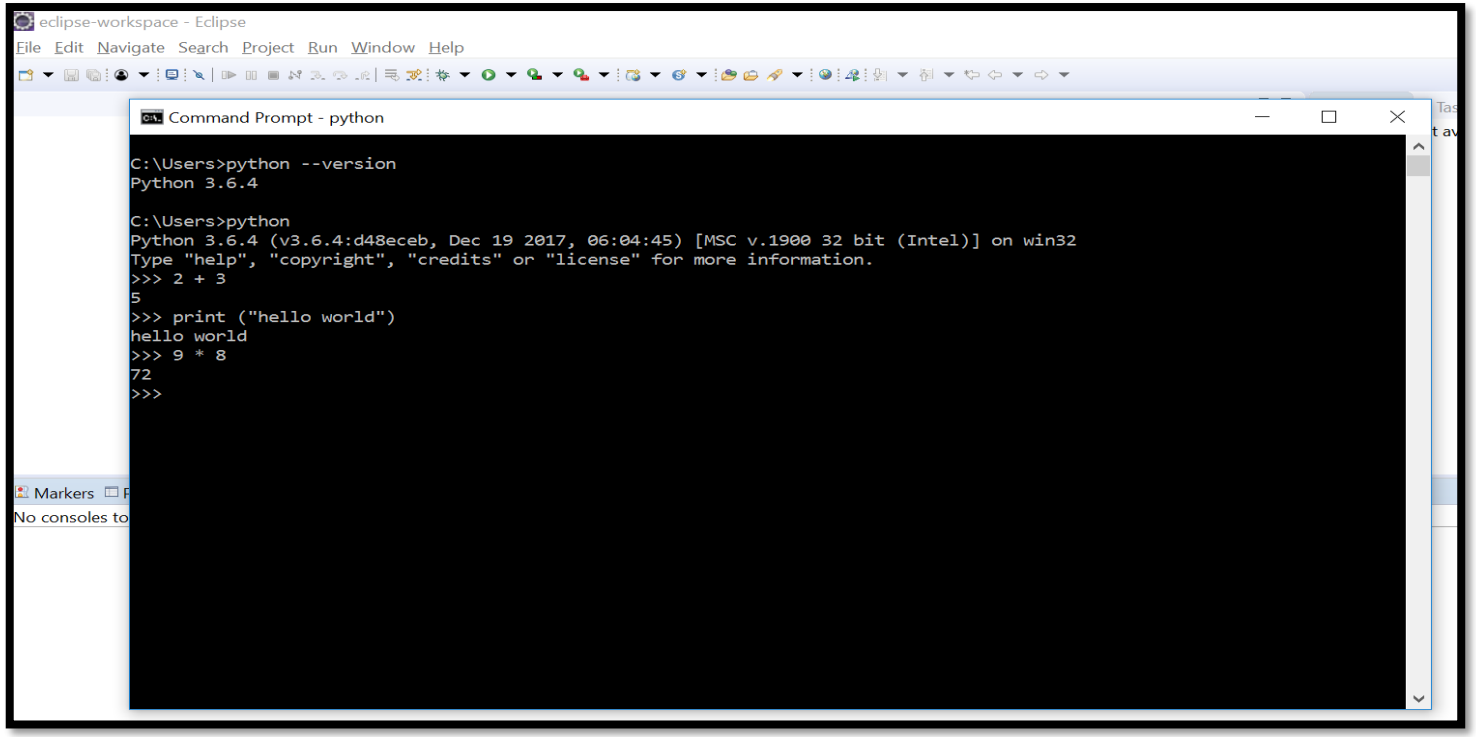
- Then, click **System Variables -> Select Path -> Select Edit -> New**
- Select the path of Python 3 and Scripts folder in the system and then click **OK**



Python Installation(5)

Steps to Install Python

- Now, you can access python script from windows command prompt as below:



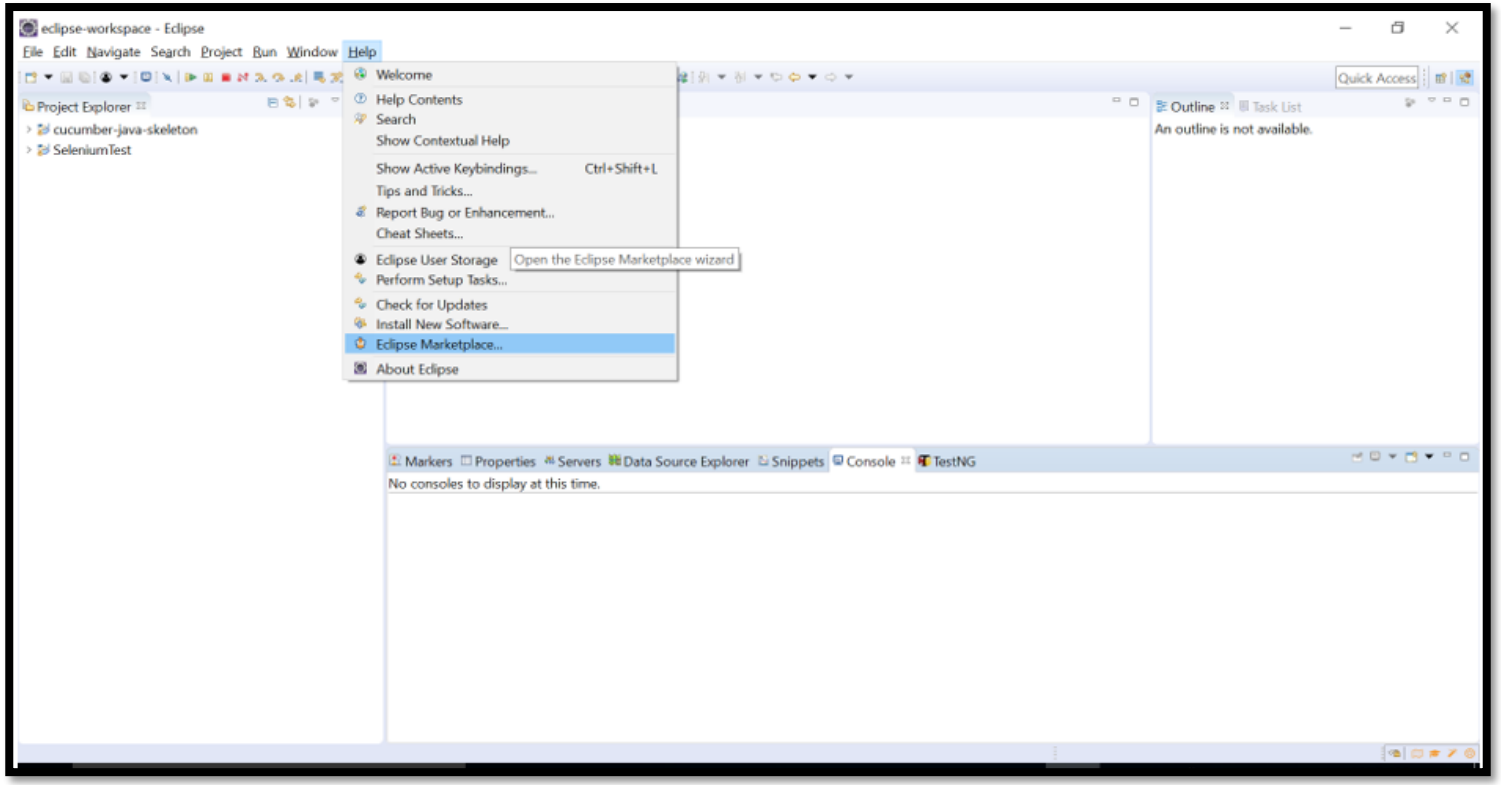
Topic List

PyDev Installation in Eclipse (1)

Steps to install PyDev in Eclipse

Follow below mentioned steps to install Pydev in Eclipse IDE:

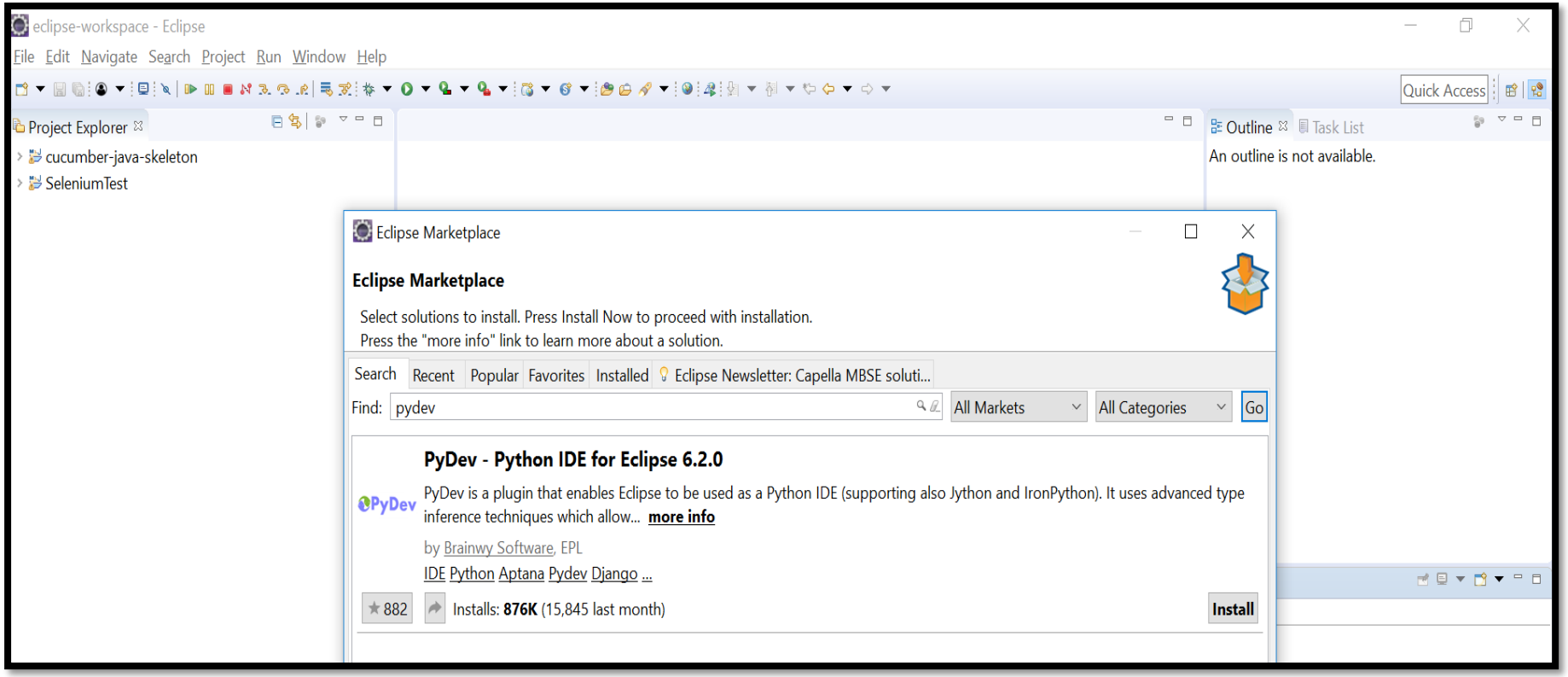
- Open Eclipse and go to **Help-> Eclipse Marketplace**



PyDev Installation in Eclipse (2)

Steps to install PyDev in Eclipse

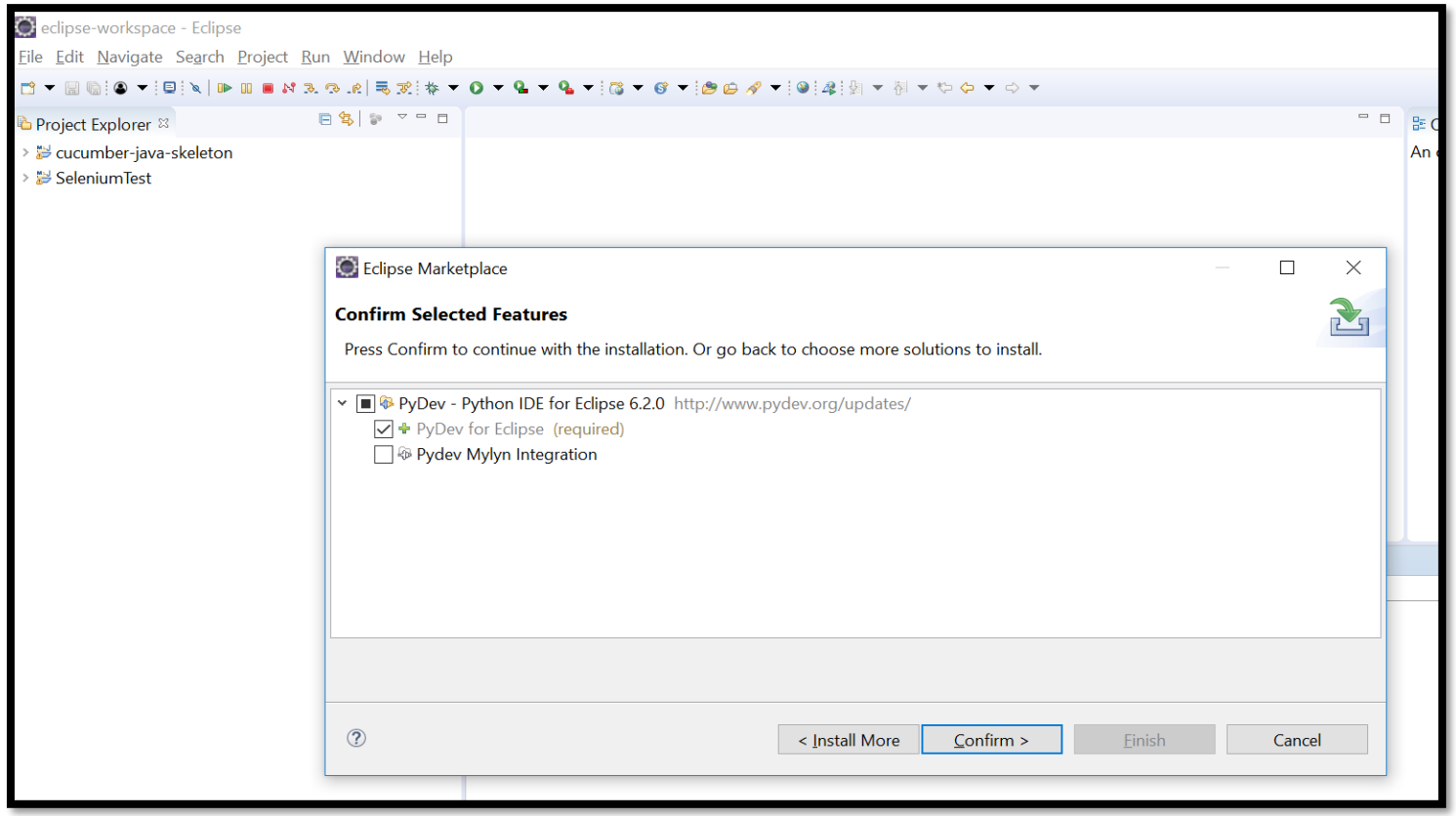
In marketplace, search for **PyDev** and click **Install**



PyDev Installation in Eclipse (3)

Steps to install PyDev in Eclipse

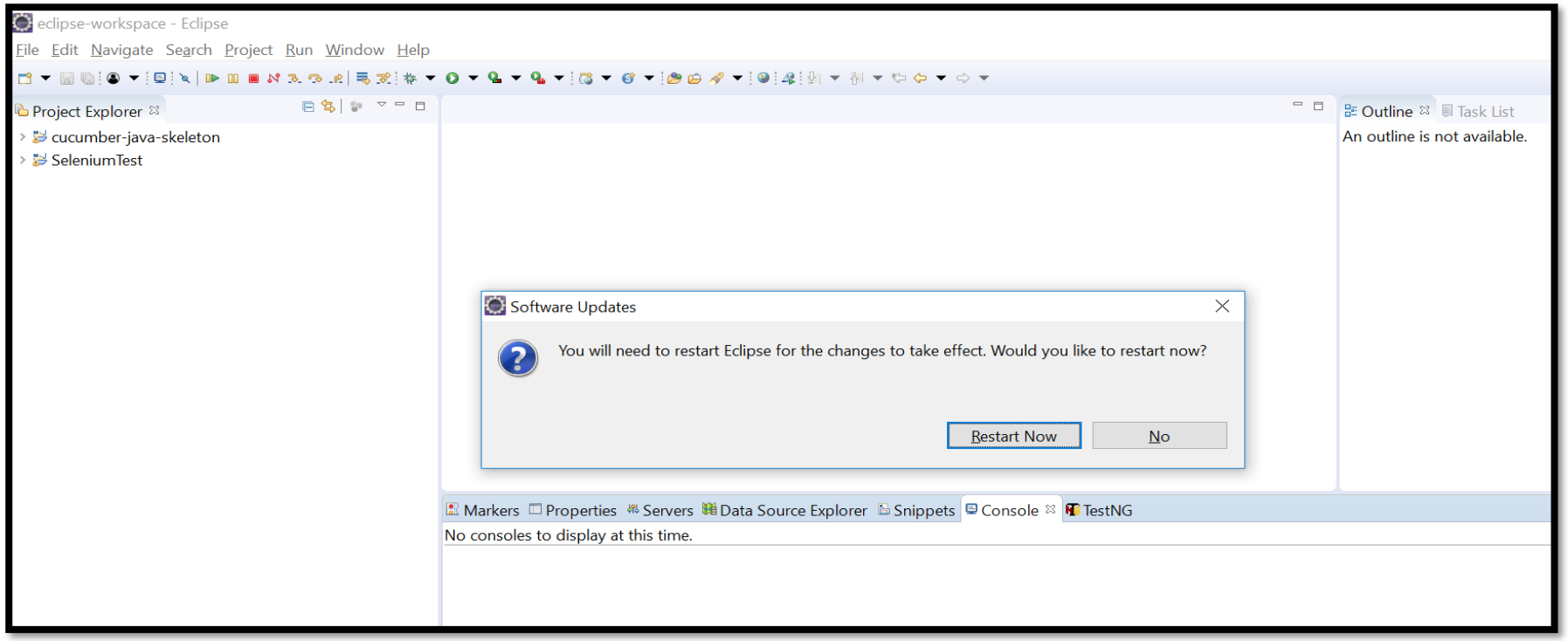
- Select **Pydev for Eclipse** and click **Confirm**



PyDev Installation in Eclipse (4)

Steps to install PyDev in Eclipse

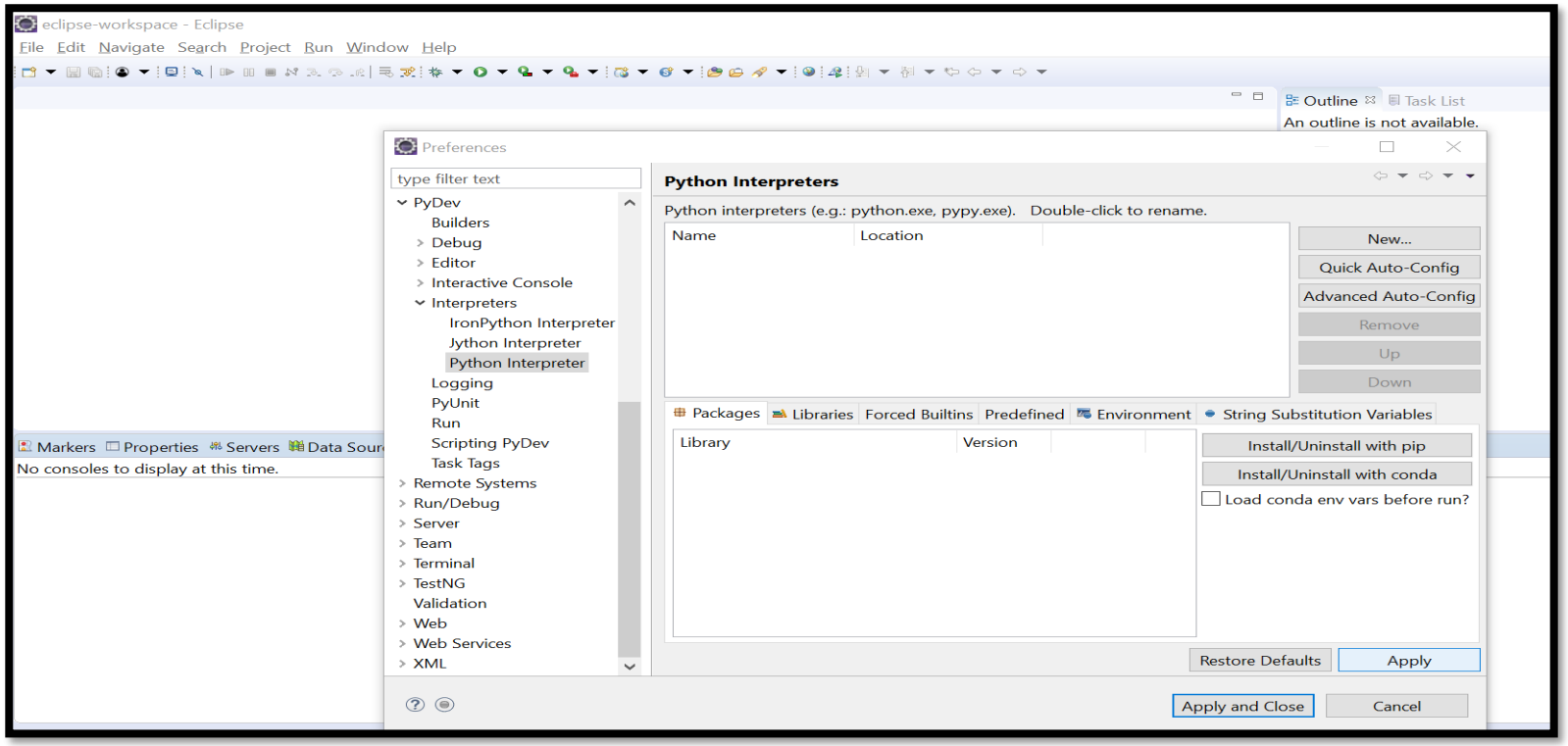
- You would be prompted for Eclipse Restart
- Click **Restart Now**. Pydev is now installed in Eclipse



PyDev Installation in Eclipse (5)

Set Python Interpreter Path After Installation

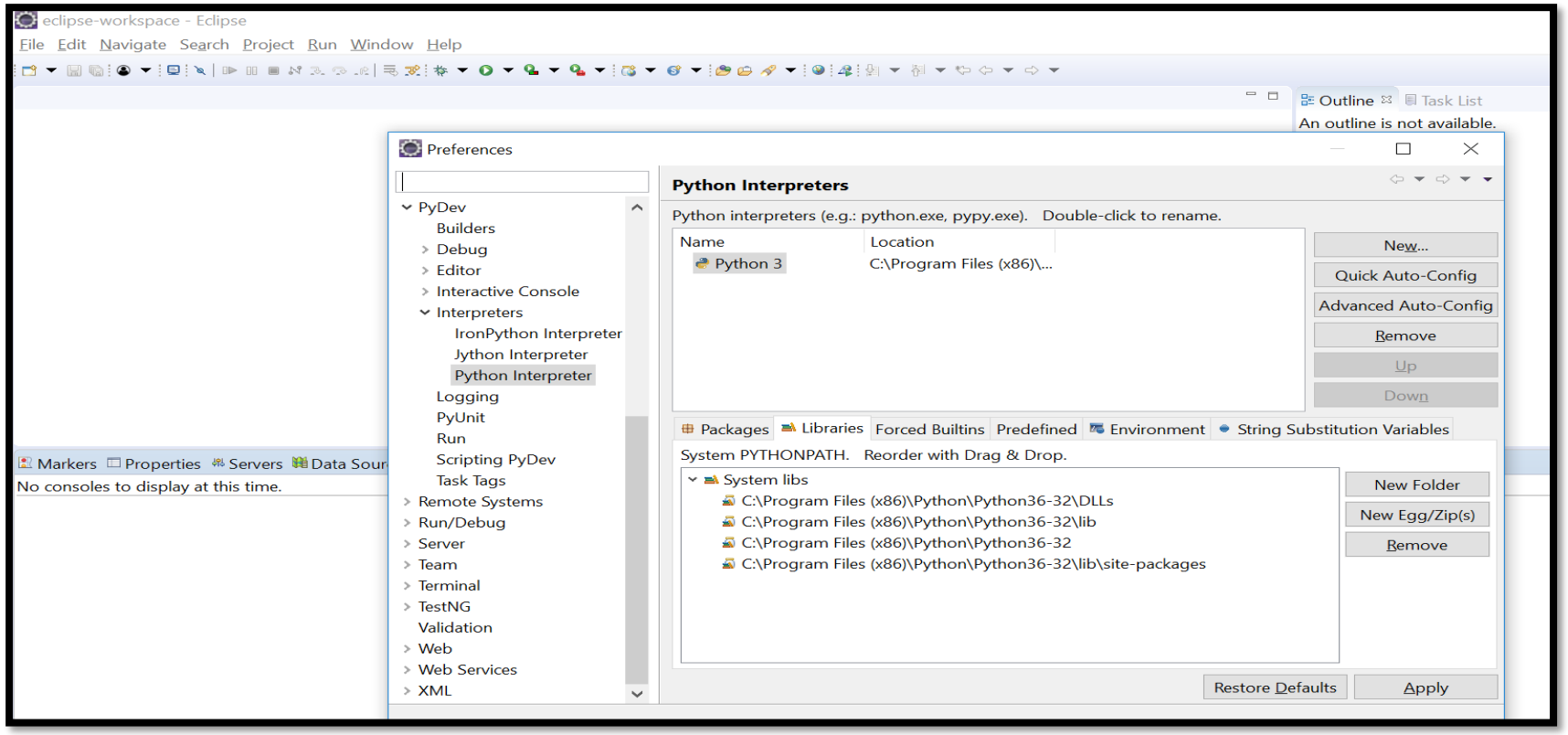
- We need to set Python interpreter path in Eclipse. Go to **Window-> Preferences -> Pydev -> Interpreters->Python Interpreter**
- Provide the path of Python interpreter which we have downloaded



PyDev Installation in Eclipse (6)

Set Python Interpreter Path After Installation

We can see that all required libraries are added.

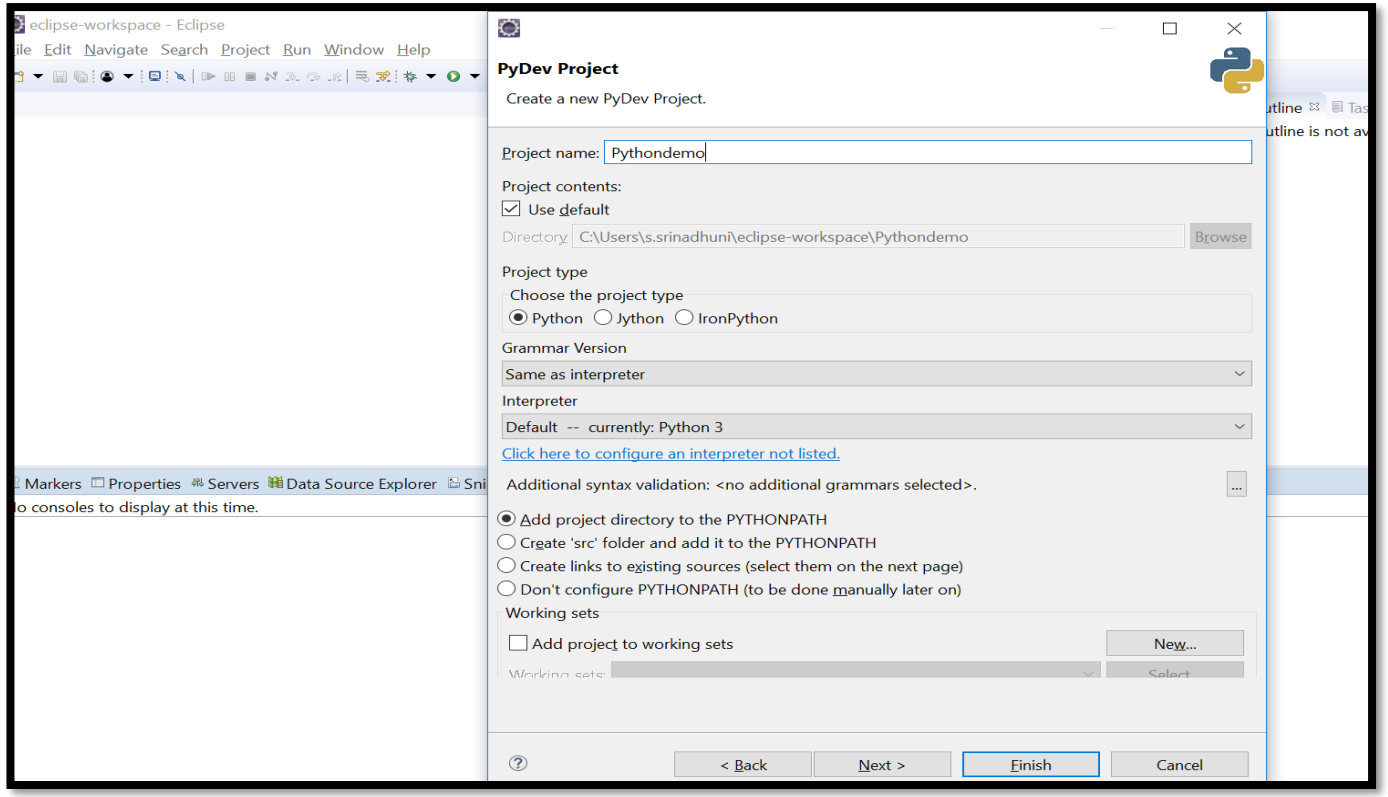


PyDev Installation in Eclipse (7)

Create Python Project

Now lets create Python project in Eclipse:

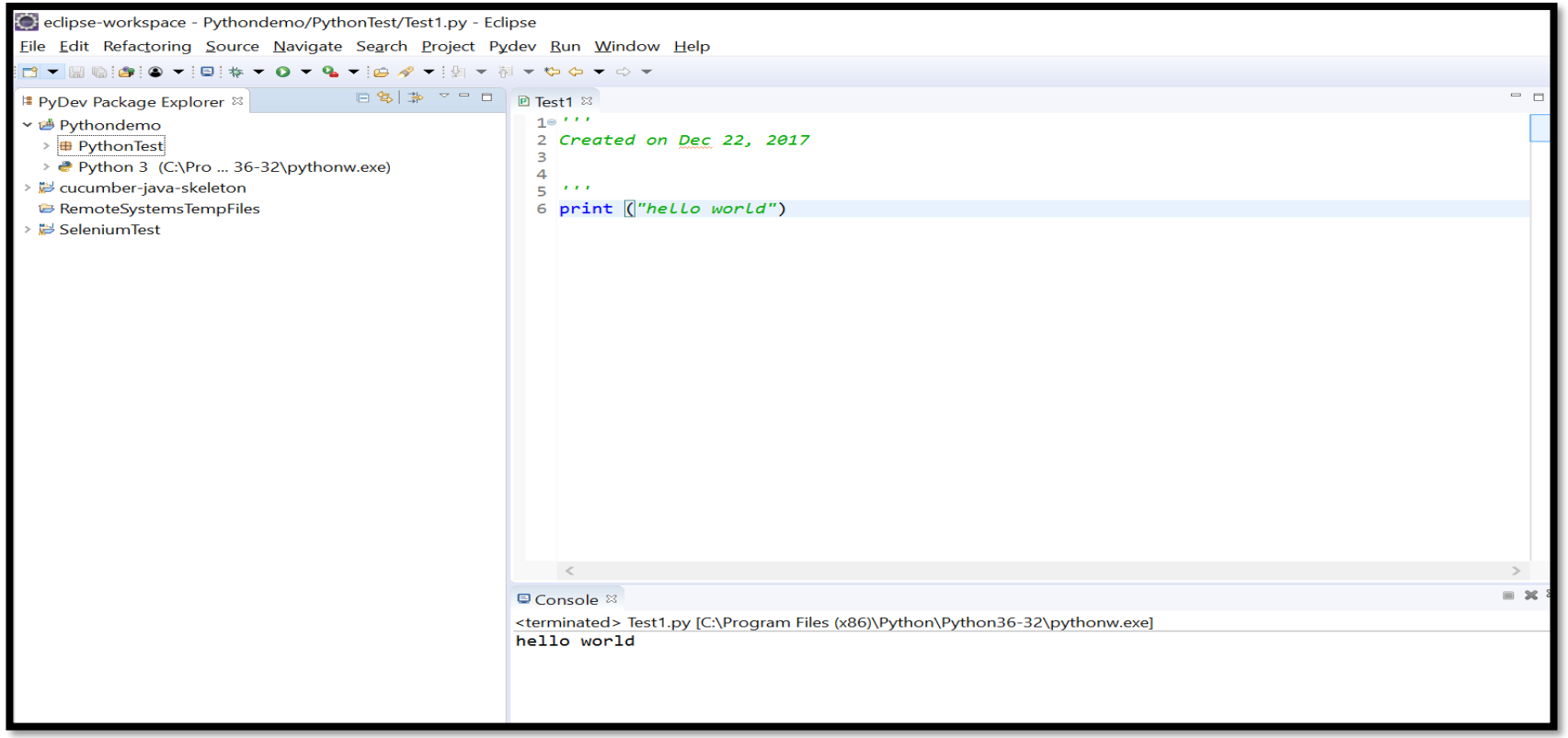
- To create a Python project, navigate to **File -> New -> Pydev Package Or Pydev Module** and provide required details in the respective fields
- Click **Finish**



PyDev Installation in Eclipse (8)

Create Python Project

- Write the required python code and right click
- Execute the code as “**python run**”. Result will be displayed in console



Exercise 2.1: PyDev Plugin in Eclipse IDE

Add PyDev plugin in Eclipse IDE

Refer Exercise 2.1 in Selenium with Python_wbk.doc for the detailed steps.

Topic List

Python as a Programming Language

Facts

- Python is a general purpose, cross-platform programming language created by Guido Van Rossum .
- Python is an Object Oriented Programming Language .
- In Python we can create Classes and Objects .
- Python is interpreted language .
- Python is well appreciated for its syntax and readable code .
- With Python we can do everything from GUI development, Web Application, System Administration task, Financial calculation, Data Analysis, Visualization and list goes on.
- It runs on multiple platforms like Windows, Mac OS X, Linux, Unix and has been ported to the Java and .NET virtual machines .

Topic List

OOP Features of Python (1)

Principles of Object Orientation

Object Orientation

OOP Features of Python (2)

Principles of Object Orientation

Encapsulation

- In an Object Oriented Python program, we can restrict access to methods and variables
- This can prevent the data from being modified by accident and is known as ***encapsulation***

Data Abstraction

- ***Abstraction*** allows us to bundle together some related things and give a name to the bundle (e. g. Functions)
- The tern encapsulation and abstraction are often used as synonyms
- Abstraction can be achieved through Encapsulation

OOP Features of Python (3)

Principles of Object Orientation

Polymorphism

- ***Polymorphism*** in Python refers programming elements such as variables, functions or object to take multiple forms
- In Python , polymorphism facilitates for generic programming than programming in the specific

Inheritance

- Inheritance allows us to extend an existing class to make a more specified class
- Defines a hierarchical relationship among classes
- Allows you to reuse existing code
- Existing class can be called as Parent class and newly derived class can be called as Child class

Topic List

Data Types and Variables (1)

Variables :

A variable is a location in memory used to store some data (values)

Rules for writing Variables :

- Variables can be a combination of letters in lowercase (a to z), or uppercase (A to Z) or digit (0 to 9) or an underscore (_)
- Variable can not start with a digit
- Python Keywords can not be used as variables
- Special symbols like $, %, #, !, @ etc cannot be used in variables
- Variable name can be of any length

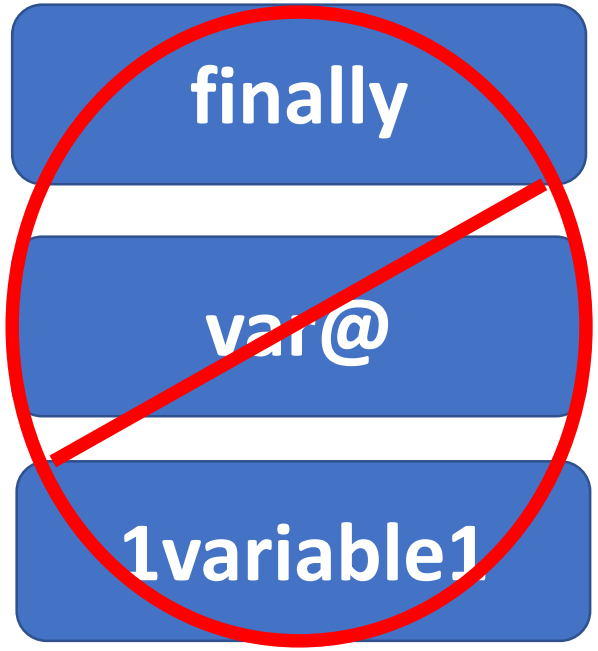
Data Types and Variables (2)

Examples- Dos and Don’ts of Variables

myClass

var_in_python

variable1



Data Types and Variables (3)

Data Types

Every value in Python has a data type

Python Data Types

- Numbers
- String
- List
- Set
- Tuple
- Dictionary

Data Types and Variables (4)

Variable Declaration

Number : int, float and complex data types falls under Number .

Syntax : var_name = value ;

Example : num1=5**;**

List : is an ordered sequence of different data items .

Syntax : var_name = [value1,.,.,.,valueN]

Example : my_list=[1,4.5,'Python']

Tuple : is an ordered sequence of different data items .

Tuples are immutable

Syntax: var_name=(value1,.,.,valueN)

Example : tpl=(96,'Python$',5.7)

Data Types and Variables (5)

Variable Declaration

String : is ordered collection of unique items .

Syntax : var_name = {value1,.,.,.,valueN}

Example : lang = {‘C’,’C++’,’Python’,’Java’}

Set : is an ordered sequence of different data items .

Syntax : var_name = [value1,.,.,.,valueN]

Example : my_list=[1,4.5,’Python’]

Dictionary : is an unordered collection of key-value pairs . Key and value can be of any type .

Syntax: var_name= {key:value1,.,.,.,key:valueN}

Example : dct= {1:”Python”,’key’: 96}

Topic List

Operators and Keywords (1)

Operators and Operands

- Operators are special symbols in that performs logical or arithmetic computations
- Value on which the operator operates is called the ***Operand***
- Following are the operator types in Python :

Arithmetic operators
Relational (Comparison) operators
Boolean (Logical) operators
Bitwise operators
Assignment operators
Special operators

Operators and Keywords (2)

Arithmetic Operators

Operator	Meaning	Example
+	Add two operands or unary plus	Num1 + Num2 +9
-	Subtract right operand from the left or unary minus	Num1 – Num2 -7
*	Multiply two operands	Num1 * Num2
/	Divide left operand by the right operand (always results into float)	Num1 / Num2
%	Modulus - remainder of the division of left operand by the right	Num1 % Num2 (remainder of Num1/Num2)
//	Floor division - division that results into whole number adjusted to the left in the number line	Num1 // Num2
**	Exponent - left operand raised to the power of right	Num1**Num2 (Num1 to the power Num2)

Operators and Keywords (3)

Relational Operators

Operator	Meaning	Example
>	Greater that - True if left operand is greater than right	Num1 > Num2
<	Less that - True if left operand is less than right	Num1 < Num2
==	Equal to - True if both operands are equal	Num1 == Num2
!=	Not equal to - True if operands are not equal	Num1 != Num2
>=	Greater than or equal to - True if left operand is greater than or equal to the right	Num1 >= Num2
<=	Less than or equal to - True if left operand is less than or equal to the right	Num1 <= Num2

Operators and Keywords (4)

Boolean Operators

Operator	Meaning	Example
and	True if both the operands are true	Num1 and Num2
or	True if either of the operands is true	Num1 or Num2
not	True if operand is false (complements the operand)	not Num1

Operators and Keywords (5)

Bitwise Operators

Operator	Meaning	Example
&	Bitwise AND	Num1 & Num2 = 0 (0000 0000)
|	Bitwise OR	Num1 | Num2 = 14 (0000 1110)
~	Bitwise NOT	~Num1 = -11 (1111 0101)
^	Bitwise XOR	Num1 ^ Num2 = 14 (0000 1110)
>>	Bitwise right shift	Num1>> 2 = 2 (0000 0010)
<<	Bitwise left shift	Num1<< 2 = 40 (0010 1000)

Operators and Keywords (6)

Assignment Operators

Operator	Example	Equivatent to
=	Num1 = 5	Num1 = 5
+=	Num1 += 5	Num1 = Num1 + 5
-=	Num1 -= 5	Num1 = Num1 - 5
*=	Num1 *= 5	Num1 = Num1 * 5
/=	Num1 /= 5	Num1 = Num1 / 5
%=	Num1 %= 5	Num1 = Num1 % 5
//=	Num1 //= 5	Num1 = Num1 // 5

Operators and Keywords (7)

Assignment Operators

Operator	Example	Equivatent to
**=	Num1 **= 5	Num1 = Num1 ** 5
&=	Num1 &= 5	Num1 = Num1 & 5
|=	Num1 |= 5	Num1 = Num1 | 5
^=	Num1 ^= 5	Num1 = Num1 ^ 5
>>=	Num1 >>= 5	Num1 = Num1 >> 5
<<=	Num1 <<= 5	Num1 = Num1 << 5

Operators and Keywords (8)

Special Operators

Operator	Meaning	Example
is	True if the operands are identical (refer to the same object)	Num1 is True
is not	True if the operands are not identical (do not refer to the same object)	Num1 is not True

Operators and Keywords (9)

Python Keywords

as	del	false	nonlocal	true
and	elif	global	not	while
assert	else	import	or	with
break	except	in	pass	yield
class	finally	is	raise	if
continue	for	lambda	return	
def	from	none	try	

Topic List

- **Conditional and Looping statements**
- Functions and Scope of Variables
- Defining Class and Objects
- Command Line Arguments
- Exceptions And Assertions
- Handling Strings

Conditional and Looping statements (1)

Conditional Statements

- **if…else**

Syntax

```
if test expression:
        Body of if
else:
        Body of else
```

Example:

```
Num = 9
if Num >= 0:
        print("Positive or Zero")
else:
        print("Negative number")
```

Conditional and Looping statements (2)

Looping Statements

- **for** statement

Syntax

```
for value in sequence:
        Body of for
```

Example:

```
values = [7, 5, 9, 8, 4, 2, 1, 4, 15]
Add=0
for val in values :
        Add=Add + val
        print("Addition is :", Add)
```

- **While** statement

Syntax

```
While test_expression:
        Body of while
```

Example:

```
Num=10
Add=0
i=1
While i<=Num
        Add=Add+i
        i=i+1
        print("Addition is :",Add)
```

Topic List

Conditional and Looping statements

Functions and Scope of Variables

Defining Class and Objects

Command Line Arguments

Exceptions And Assertions

Handling Strings

Functions and Scope of Variables (1)

Functions

A function is a group of related statements that performs a specific task .

Syntax :

```
def
function_name(parameters):
        """docstring"""
        statement(S)
```

Example :

```
def FirstMethod(stmt):
"""Writing the First Function in Python"""
print("Hi All, " + stmt )
```

Functions and Scope of Variables (2)

Types of Functions

Functions can be divided in to two types

Built-in Functions

Functions that are defined or built in to the interpreter and can be called to use in the code

Function	Description
print()	Prints the specified object
str()	Returns informal representation of an object
float()	Returns floating point number from number and string
Input()	Reads and returns a line of string

User Defined Functions

Functions that are defined by users to perform specific actions on objects

```
# Program to illustrate the use of user-defined functions

def AddOfNumbers(num1,num2):
    sum = num1 + num2
    return sum
    num11 = 10
    num22 = 20
    print("The sum is", AddOfNumbers(num11, num22))
```

Functions and Scope of Variables (3)

Scope of Variables

Scope of Variable is part of a program where the variable is recognized

- Parameters with local scope are those variables that are defined inside a function is not accessible from outside.
- Lifetime of a variable is the period throughout which the variable exists in the memory
  - The lifetime of variables inside a function is as long as the function executes.
  - They are destroyed once the function has finished executing. Thus, the function does not remember the value of a variable from its previous calls

Example:

```
def VarScope():
        num = 10
        print("Value inside function:",num)
num = 20
VarScope()
print("Value outside function:", VarScope)
```

Output:

Value of num inside function: 10
Value of num outside function: 20

Topic List

Defining Class and Object (1)

Class Definition

- Class definition begins with the keyword **class** .

Syntax

```
class ClassName :
        <statement-1>
        .
        .
        .
        .
        <statement-N>
```

Example:

```
class MyClass:
"This is my second class"
Num = 10
        def Method(self):
                print('Hello')
# Output: 10
print(MyClass.Num)
# Output:<function MyClass.Method at 0x0000000003079BF8>
print(MyClass.Method)
# Output: 'This is my second class'
print(MyClass.__doc__)
```

Defining Class and Object (2)

Defining Objects

- Object is an instance of a class; collection of variables(data) and functions(methods) that act on those data .

Syntax

```
objectName = className()
```

Example:

```
class Shark:
    def swim(self):
        print("The shark is swimming.")
    def be_awesome(self):
        print("The shark is being awesome.")
# Creating object of class Shark
sammy = Shark()
sammy.swim()
sammy.be_awesome()
```

Output:

The shark is swimming
The shark is being awesome

Exercise 2.2: Defining Objects and Functions in Python

Write a program to create a simple calculator that can Add, Subtract, Multiply and divide two numbers depending upon the input from the user

- To implement this program, define four parameterized functions Add, Subtract, Multiply, Divide respectively

Refer Exercise 2.2 in Selenium with Python_wbk.doc for the detailed steps.

Topic List

Command Line Arguments (1)

sys Module

- **sys** module provides access to command line arguments via **sys.argv**
- sys.argv is the list of command line arguments
- Len(sys.argv) is the number of command line arguments

Following is an example of sys module implementation:

```
import sys
program_name = sys.argv[0]
args = sys.argv[1:]
countofargs = len(args)
```

Command Line Arguments (2)

Reading Arguments from Command Line

- Example code snippet to read arguments from command line is shown below:

```
import sys
for i in sys.argv:
        print "Argument: ", i
```

```
len(sys.argv) ->  Checks the number of arguments entered
len(sys.argv) != 3  - >  Checks whether you have entered at least three elements
```

Command Line Arguments (3)

Reading Arguments from Command Line

```
import sys
if len (sys.argv) != 2 :
        print "Usage: python demo.py "
        sys.exit (1)
```

```
Execution
>>python demo.py demo script
Argument: demo.py
Argument: demo
Argument: script
```

Topic List

Conditional and Looping statements

Functions and Scope of Variables

Defining Class and Objects

Command Line Arguments

Exceptions And Assertions

Handling Strings

Exceptions and Assertions (1)

Exceptions

- Exceptions are errors detected at the time of program execution.
  - Example: Lets say you are driving, road is clear and you reach the destination place successfully.

Exceptions and Assertions (2)

Exceptions

- Accident = Exception

Exceptions and Assertions (3)

Exception, An Example

Validate whether given number is a integer.

```
n=int(input("Enter a Number to validate :"))
Enter a Number to validate :12.3
Traceback (most recent call last):
 File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: '12.3'
```

Exceptions and Assertions (4)

Exception Handling

- Example code snippets of exception handling

```
while True:
    try:
        n = input(" Enter a Number to validate : ")
        n = int(n)
        break
    except ValueError:
        print(“Not a valid integer! Please try again ...")
print(“Successfully entered an integer!")
```

```
Num1=input("Enter a Number1:")
Num2=input("Enter a Number2:")
try:
  val=(Num1) / int(Num2)
except ZeroDivisionError as e:
  print ('ZeroDivisionError Exception')

except TypeError as e:
  print('Type Error Exception')
```

Exceptions and Assertions (5)

Try-except-finally

- Example of implementing Try-except-finally is shown below
  - Even if the division by zero exception is not handled, code will always execute the finally block.

```
def demo_final():
  try:
    f=open("c:\\python\\demo.txt")
    x=10/0
    except FileNotFoundError as e:
      print("Except block")
    finally:
      print("Final block - Cleaning up file")
      f.close()

demo_final()
```

Exceptions and Assertions (6)

Built-in-Exceptions

Exception	Description
EOFError	Raised when the input() functions hits end-of-file condition.
MemoryError	Raised when an operation runs out of memory.
SyntaxError	Raised by parser when syntax error is encountered.
TypeError	Raised when a function or operation is applied to an object of incorrect type.
ValueError	Raised when a function gets argument of correct type but improper value
ZeroDivisionError	Raised when second operand of division or modulo operation is zero.

Exceptions and Assertions (7)

Assertions

- “Assert” is a python keyword.
- Assert statements are a convenient way to insert debugging assertions into a program
- Syntax:

  1. assert<condition>

  2. assert<condition>,<error message>

- If<conditon> is False, raise an AssertionError exception
  - Example:

```
def get_age(age):
   print(" Your age is :",age)

get_age(20)
```

Exceptions and Assertions (8)

- But assume if the code snippet is

```
def get_age(age):
   print(" Your age is :",age)

get_age(-1)
```

Age cannot be Negative!

Exceptions and Assertions (9)

- Assertion Example : Code snippet

```
def get_age(age):
  assert age > 0, " Age can't be Negative!"
  print(" Your age is :",age)

get_age(-1)
```

Exercise 2.3: Illustrate Exception Handling in Python

Write a program to handle divide by zero exception

Refer Exercise 2.3 in Selenium with Python_wbk.doc for the detailed steps.

Topic List

Conditional and Looping statements

Functions and Scope of Variables

Defining Class and Objects

Command Line Arguments

Exceptions And Assertions

Handling Strings

Handling Strings (1)

About Strings

String is a sequence of characters.

- A String literal in python can be represented as **“Demo Python”** or **‘Demo Python’**
- Index starts at 0.
  - Example :

```
>>> val="This is a sample statement for demo"
>>> op=val[3]
>>> print(op)
s
```

Operators

In Operator: The word **in** is a boolean operator that takes two strings and returns True if the first appears as a substring in the second

- Example :

```
>>> 'A' in "Accenture"
True
>>> 'p' in "Accenture"
False
```

Handling Strings (2)

String Methods

- Python has a function called **dir** which lists the methods available for an object.

  - Example:

```
>>> orgname="Accenture"
>>> type(orgname)
<class 'str'>
>>> dir(orgname)
['__add__', '__class__', '__contains__', '__delattr__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__getitem__', '__getnewargs__', '__gt__',
'__hash__', '__init__', '__init_subclass__', '__iter__', '__le__', '__len__', '__lt__', '__mod__', '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '
__rmod__', '__rmul__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__', 'capitalize', 'casefold', 'center', 'count', 'encode', 'endswith', 'expandtabs', 'find', 'fo
rmat', 'format_map', 'index', 'isalnum', 'isalpha', 'isdecimal', 'isdigit', 'isidentifier', 'islower', 'isnumeric', 'isprintable', 'isspace', 'istitle', 'isupper', 'join', 'lj
ust', 'lower', 'lstrip', 'maketrans', 'partition', 'replace', 'rfind', 'rindex', 'rjust', 'rpartition', 'rsplit', 'rstrip', 'split', 'splitlines', 'startswith', 'strip', 'swap
case', 'title', 'translate', 'upper', 'zfill']
```

Handling Strings (3)

String Methods

Method	Description
capitalize()	Capitalizes first letter of string
index(str, beg=0, end=len(string))	Same as find(), but raises an exception if str not found
Len(str)	Returns the length of the string
join(seq)	Merges (concatenates) the string representations of elements in sequence seq into a string, with separator string.
replace(old, new [, max])	Replaces all occurrences of old in string with new or at most max occurrences if max given.
rfind(str, beg=0,end=len(string))	Same as find(), but search backwards in string

Handling Strings (4)

String Methods

Method	Description
lstrip()	Removes all leading whitespace in string.
upper()	Converts lowercase letters in string to uppercase.
isnumeric()	Returns true if a unicode string contains only numeric characters and false otherwise.
isalnum()	Returns true if string has at least 1 character and all characters are alphanumeric and false otherwise.
count(str, beg= 0,end=len(string))	Counts how many times str occurs in string or in a substring of string if starting index beg and ending index end are given.
decode(encoding='UTF-8',errors='strict')	Decodes the string using the codec registered for encoding. encoding defaults to the default string encoding.

Handling Strings (5)

String Method-Examples

- Code snippet with **find ()** method to search for the position of one string within another.

```
>>> demostr=" This is a sample demo for strings"
>>> index=demostr.find('a')
>>> print(index)
9
```

- Code snippet with Startswith() method

```
>>> demostr='This is a sample demo for strings'
>>> demostr.startswith('sample')
False
>>> demostr.startswith('This')
True
```

Module Summary

Now, you should be able to:

- Illustrate Python installation
- Demonstrate adding Pydev Plugin in Eclipse IDE
- Explain how to create Python projects
- Describe implementing object orientation concepts and structres in Python programs
- Use Command Line Arguments
- Apply Exceptions and Assertions
- Describe the various String Functions

Reference

- https://www.python-course.eu/python3_exception_handling.php
- https://www.tutorialspoint.com/
- https://www.programiz.com/python-programming/exceptions

Thank You