

# Python

# What & Why Of python

- Python is a general purpose, interpreted, interactive , object oriented, high level programming language.
- It was first introduced in 1991 by Guido Van Rossum, a Dutch programmer.
- The language places strong emphasis on code reliability and simplicity so that programmers can develop applications rapidly.



**Trivia :** The name Python was inspired from the series Monty Python's flying circus.

Large  
Standard  
Library

High Level

Open  
Source

Simple

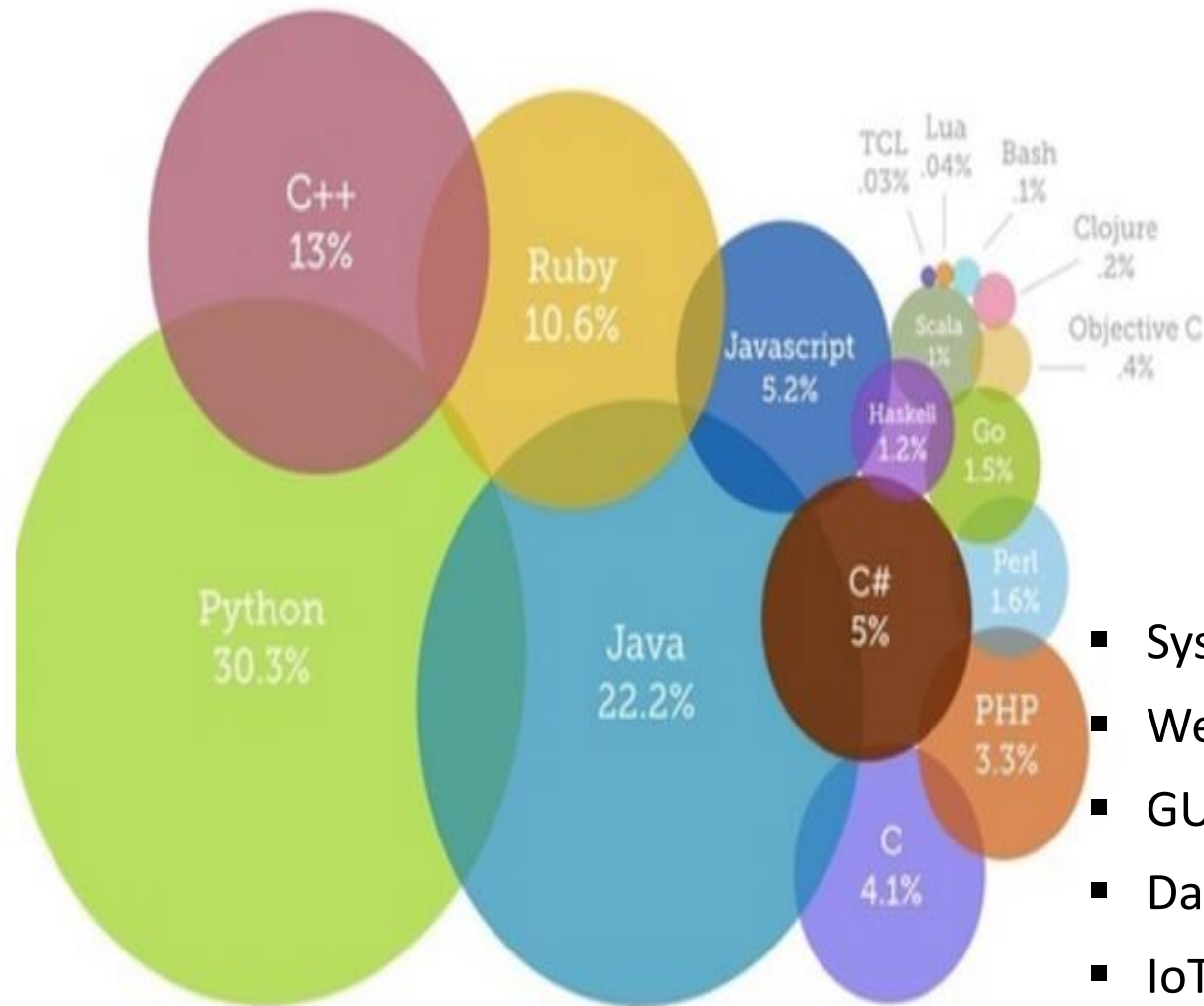
GUI  
Programming

Expressive  
language

# Features of python

- **Easy-to-learn, read, and maintain**
  - Few keywords, simple structure, and a clearly defined syntax .
- **Easy-to-use**
  - Interactive programming experience
- **A broad standard library**
  - Portable library and cross-platform compatible on UNIX, Windows and Mac
- **Open source**
  - Free to use and distribute
- **Portable**
  - Supports a wide variety of hardware platforms . Same code can be run on different platforms without any modification.
- **Object-Oriented**
  - An object-oriented language, from the ground up with support for advanced notions as well .

# WHERE is python USED ?



Most Popular coding languages of 2015 : Source  
[blogs.codeeval.com](http://blogs.codeeval.com)

© DIPTARKO DAS SHARMA



## django

- Systems Programming.
- Web Application Development.
- GUI and Scientific programming
- Data Analytics
- IoT
- Machine Learning



# Installing python

- Version to Install : Python 3.9.0
- OS Needed :Not supported on windows 7 or earlier Versions.
- OS can be 32 Bit or 64 bit. Both types of installable supported.
- Standard Hardware configurations(1 GB RAM)
- How to download :-
  - Ensure you have admin rights to download and install executables.
  - Go to **[www. Python.org/downloads](https://www.python.org/downloads)**

# Installing python



- Defaults to Windows 64 Bit Download.
- For other Versions , click on the “View the full list of downloads option”.

# Installing python

- Double-click the icon labeling the file python-3.9.0.exe. An Open File - Security Warning pop-up window will appear.



- Click Run, A Python 3.9.0 Setup pop-up window will appear. (Please refer next slide for the screenshot)
- Ensure that the Install launcher for all users (recommended) and the Add Python 3.9 to PATH checkboxes at the bottom are checked.
- If the Python Installer finds an earlier version of Python installed on your computer, the Install Now message will instead appear as Upgrade Now (and the checkboxes will not appear).



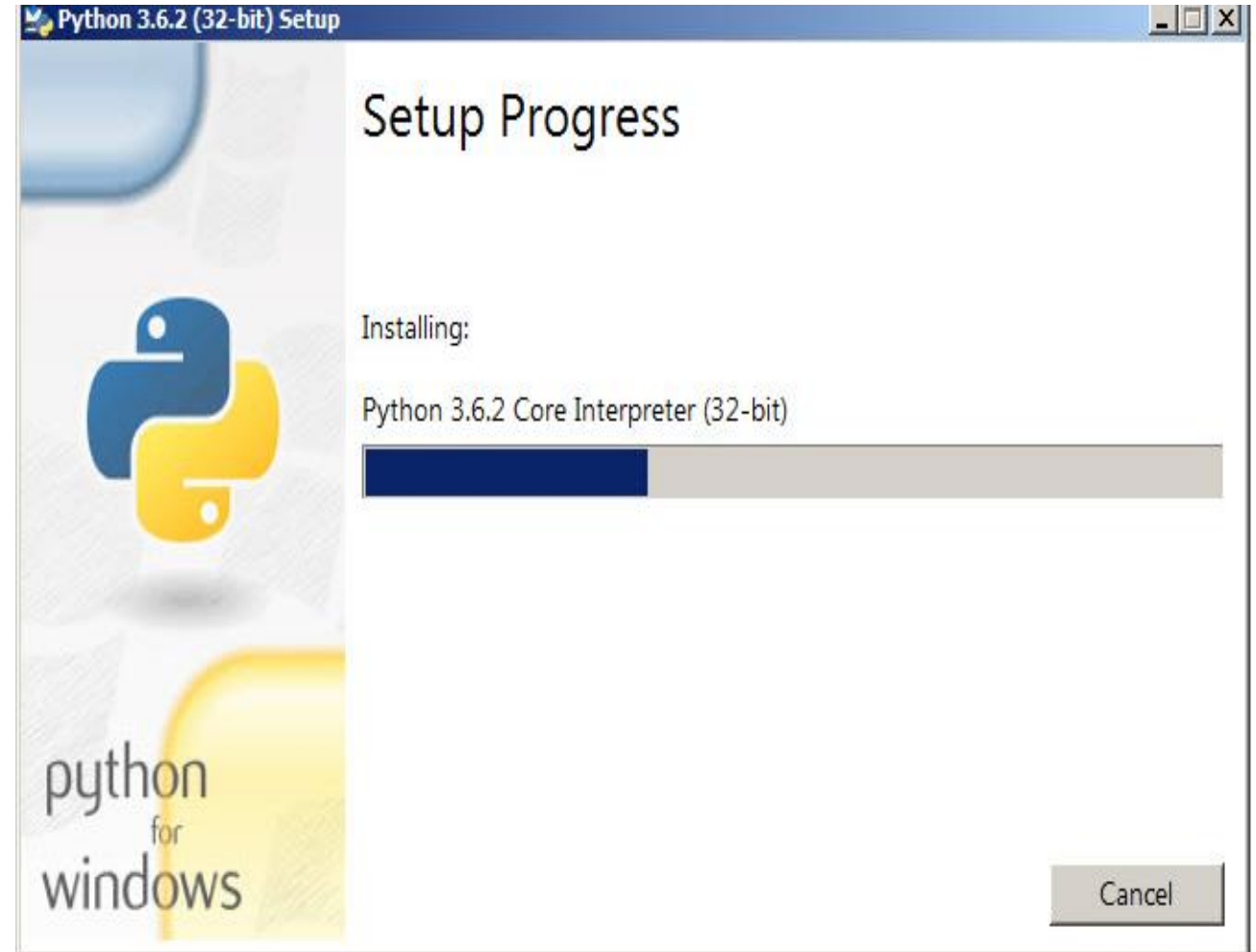
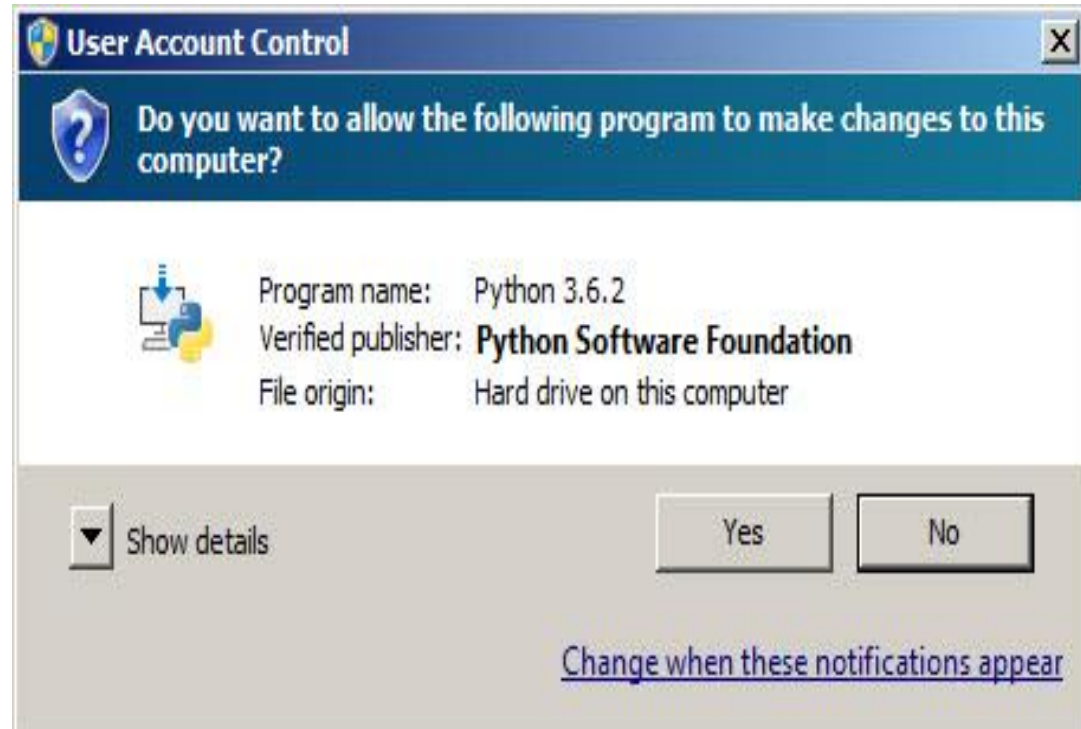
# Installing python



- Highlight the Install Now (or Upgrade Now) message, and then click it. A User Account Control pop-up window will appear, posing the question Do you want to allow the following program to make changes to this computer?(Please ref screenshot in next slide)



# Installing python



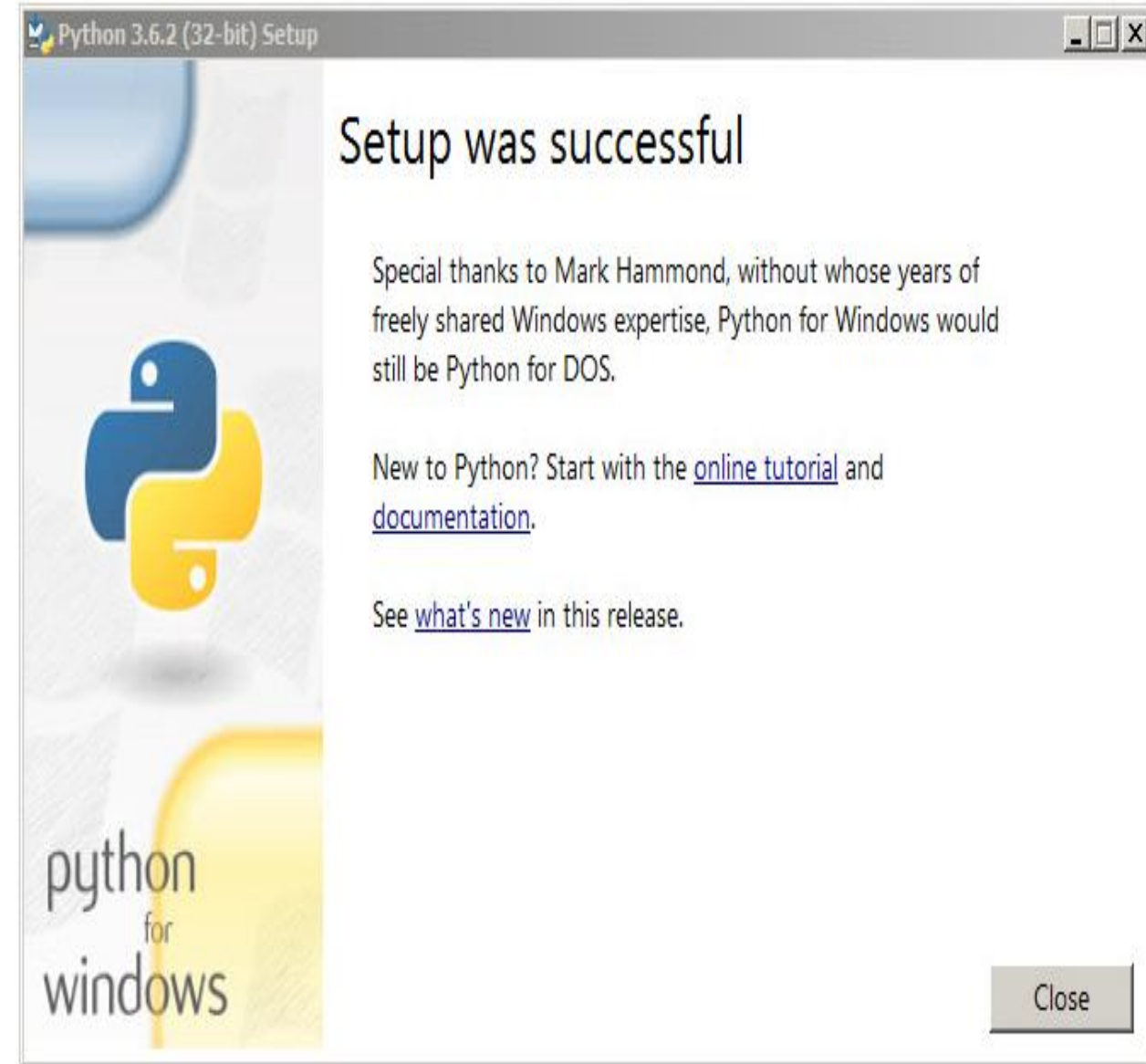
- Click the Yes button. A new Python 3.9.0 Setup pop-up window will appear with a Setup Progress message and a progress bar.
- During installation, it will show the various components it is installing and move the progress bar towards completion. Soon, a new Python 3.9.0 Setup pop-up window will appear with a Setup was successful message.

# Installing python

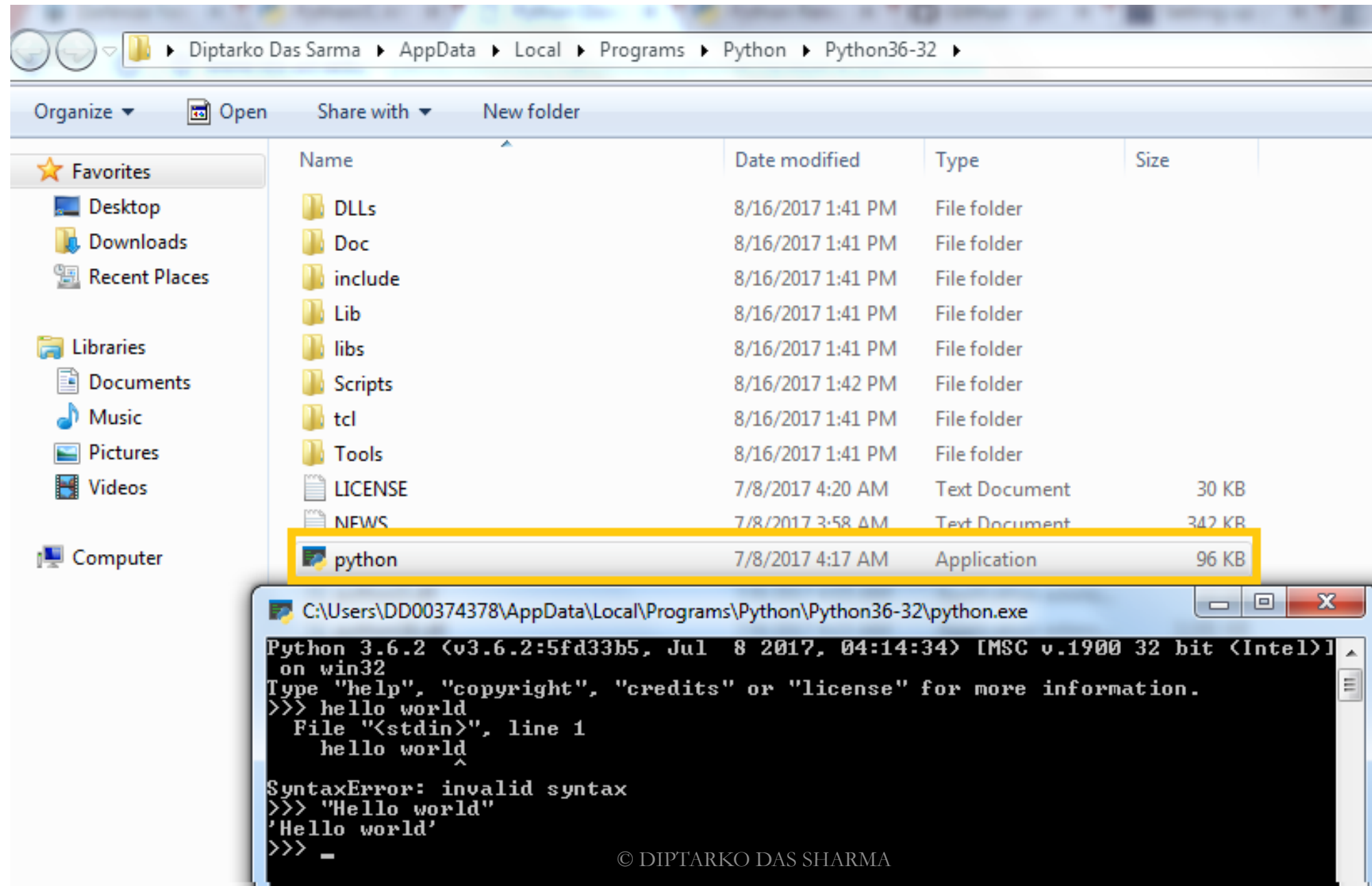
- Click the Close button. Python should now be installed.

## Verifying the installation:

- Go to the following path :-  
C:\Users\<<Username>>\AppData\Local\Programs.
- You should be able to locate the Python Folder there, under the full path  
C:\Users\<<Username>>\AppData\Local\Programs\Python\Python36-32
- Double click the Python icon(Please refer the Screen shot in next slide)
- A pop-up window with the title C:\Users\<<Username>>\AppData\Local\Programs\Python\Python36-32 appears, and inside the window; on the first line is the text Python 3.6.2 ... (notice that it should also say 32 bit). Inside the window, at the bottom left, is the prompt >>>: Here, type "Hello world", and you get the same outputted.

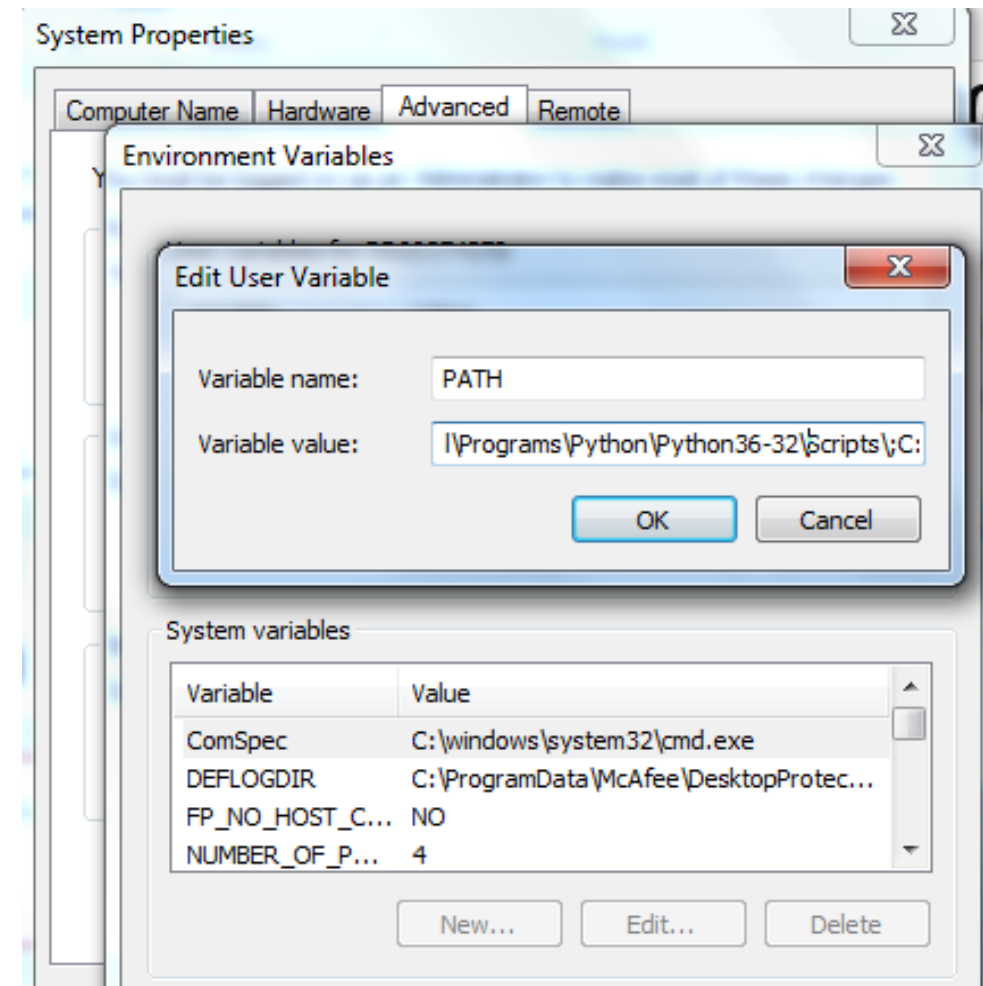
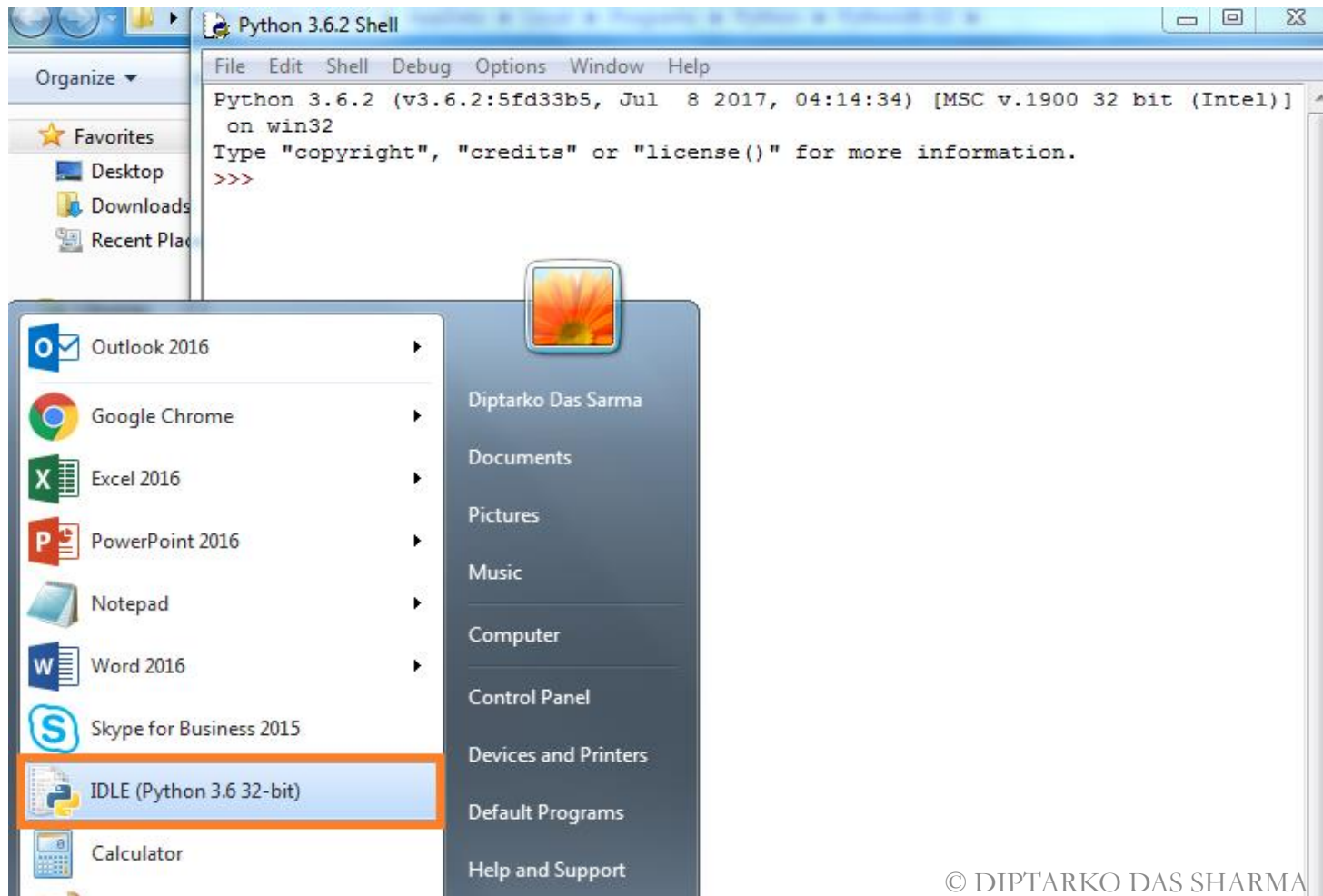


# Installing python



# Installing python

- Also, go to the Start menu , check for the Idle Icon, and open it.
- Just type in the text at the prompt, “Hello World”, and ensure that the correct output is obtained.
- Go to System environment variables, and ensure that the Python path is correctly visible in the same.



# Python Environment Setup

## Setting up PATH:-

- PATH is a variable where the OS stores data related to the folders or directories which it must search to run a particular file or executable.
- This variable is named as PATH in Unix(Unix is Case-Sensitive), PATH or Path in Windows.
- Since we are concentrating on Windows OS, we **will only check the Path in Windows.**
- Usually, at the time of Installing Python, the PATH gets sets by default(Ref the previous slides on installation) . Post installation, the path will usually have the path where the Python installation has been downloaded.
- This is the Path from where the python interpreter is invoked, whenever you run Python.
- It is also the path where your Default IDE , **Idle** is Installed.
- The Path will also specify the path , where the OS will search for the Python Programs/Modules which you have written and run those files.



# Python interpreters

- Suppose you are in Germany, and you find yourself stranded because of the lack of knowledge of the language.



- Out comes a friend who knows German, and accompanies you throughout your stay in Germany.

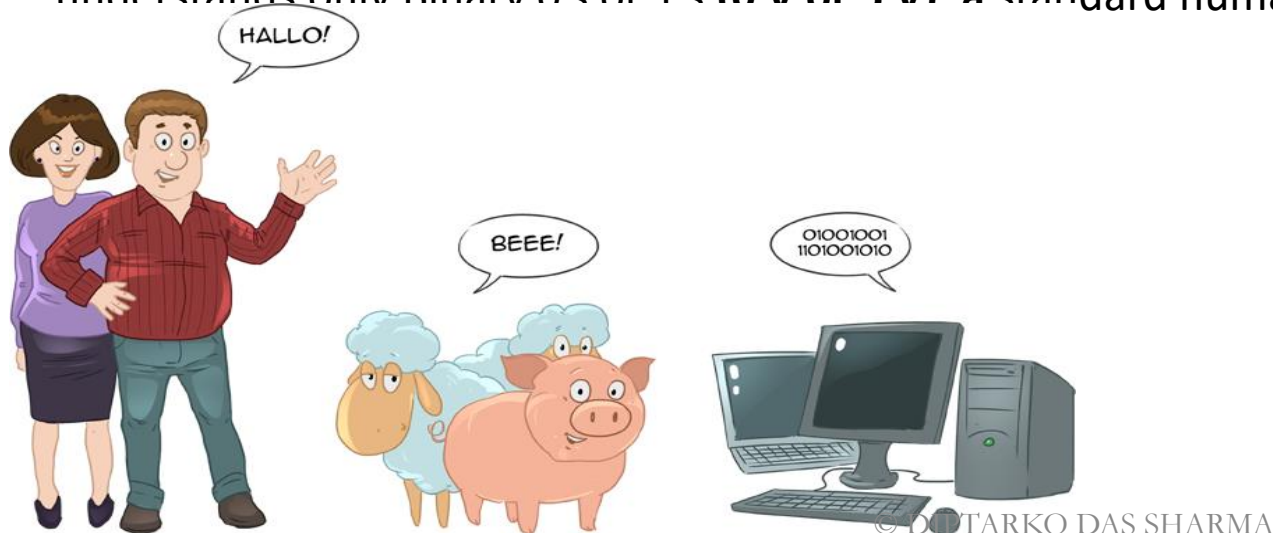


# Python interpreters

- Your life now becomes a breeze. You can go for shopping, or touring the entire German countryside without any tension 😊



- A program interpreter works in much the same way, interpreting to the Computer, who understands only Binary 0's or 1's (0 V or 5 V) a standard human language.





# Python interpreters Vs Compilers(Contd)

Interpreters	Compilers
Translates program one statement at a time.	Scans the entire program and translates it as a whole into machine code.
It takes less amount of time to analyze the source code but the overall execution time is slower.	It takes large amount of time to analyze the source code but the overall execution time is comparatively faster.
No intermediate object code is generated, hence are memory efficient.	Generates intermediate object code which further requires linking, hence requires more memory.
Continues translating the program until the first error is met, in which case it stops. Hence debugging is easy.	It generates the error message only after scanning the whole program. Hence debugging is comparatively hard.
Programming language like Python, Ruby use interpreters.	Programming language like C, C++ use compilers.

# How does Python interpreter Work?

When you instruct Python to run your script, there are a few steps that Python carries out before your code actually starts crunching away. Specifically, it's first compiled to something called byte code, and then routed to something called a virtual machine.

- Internally, and almost completely hidden from you, Python first compiles your source code (the statements in your file) into a format known as byte code. **Compilation** is simply a **translation step**, and byte code is a lower-level, and platform-independent, representation of your source code. Roughly, each of your source statements is translated into a group of byte code instructions. This byte code translation is performed to speed execution—byte code can be run much quicker than the original source code statements.
- Once your program has been compiled to byte code (or the byte code has been loaded from .pyc files), it is shipped off for execution to something generally known as the Python Virtual Machine (PVM, for the more acronym-inclined among you). The PVM sounds more impressive than it is; really, it's just a big loop that iterates through your byte code instructions, one by one, to carry out their operations. The PVM is the runtime engine of Python; it's always present as part of the Python system, and is the component that truly runs your scripts. Technically, it's just the last step of what is called the Python interpreter.



# As we take small steps

## *Lets printout*

The print function in Python is a function that outputs to your console window whatever you say you want to print out. At first sight, it might appear that the print function is rather useless for programming, but it is actually one of the most widely used functions in all of python. The reason for this is that it makes for a great debugging tool.

"Debugging" is the term given to the act of finding, removing, and fixing errors and mistakes within code.

If something isn't acting right, you can use the print function to print out what is happening in the program. Many times, you expect a certain variable to be one thing, but you cannot see what the program sees. If you print out the variable, you might see that what you thought was, was not.

```
>>> print("hello there")
```

```
hello there
```

```
>>> print("hello" "there")
```

```
hellothere
```

```
>>> print("hello" ", " "there")
```

```
hello,there
```

```
>>> print('hello "there" ')
```

SyntaxError: EOL while scanning string literal © DIPTARKO DAS SHARMA

# As we take small steps

## Lets printout

OOPS, what went wrong 😞 ?

- Python interpreted it as a string.
- EOL while scanning string literal "EOL" stands for "end of line".
- An EOL error means that Python hit the end of a line while going through a string.

This can be because you forgot ending quotes , or because you tried to make a string extend past one line.  
Strings enclosed in single or double quotes.

>> Simple : Steps in the “\” or backslash. This is an escape character, and it will “escape” the characteristic of the following character, and just keep its visual aspect.

# As we take small steps

- `>>> print(" Hello \"There \" ")`
- `Hello "There "`

How do I print the following?

➤ “Can’t do this”



**You'll have “SUCCESS” here too!!**

# As we take small steps

## Python as a calculator:

```
>>> 2+2
```

```
4
```

```
>>> (50-5*6)/4 (Multiplication and division)
```

```
5.0
```

```
>>> 8/5 # Fractions aren't lost when dividing integers
```

```
1.6
```

```
>>> 7//3 # Integer division returns the floor value
```

```
2
```

```
>>> 7// -3 ## Floor Value
```

```
-3
```

```
>>> 7%3 # Modulo remainder
```



# As we take small steps

## Python as a calculator(Contd):

```
>>> tax = 12.5 / 100
```

```
>>> price = 100.50
```

```
>>> price * tax
```

```
12.5625
```







# Language syntax

*C makes it easy to shoot yourself in the foot; C++ makes it harder, but when you do, it blows away your whole leg. -- Bjarne Stroustrup, developer of the C++ programming language*

# Identifiers

Now for a few boring grammar rules

- In the real world, we all have names, we are all identified by names. A car can be Maruti Suzuki Ritz, or a Lamborghini Huracan. Similarly a bike can be Bullet Enfield or Bajaj Pulsar.



- Similarly, every datatype in Python has to be provided a name before we can work with it. A Python identifier is a name used to identify a variable, function, class, module, or any other object.

## **Naming Rules:**

- Variable lengths can be of anything.
- An identifier starts with a letter A to Z or a to z or an underscore (\_) followed by zero or more letters, underscores and digits (0 to 9).
- No other special characters are allowed.
- Identifier names are case sensitive.
- Python doesn't allow spaces within an identifier.

Python does not allow punctuation characters such as @, \$, and % within identifiers. Python is a case sensitive programming language. Thus, **Manpower** and **manpower** are two different identifiers in Python.

# The python lexicon

Here are naming conventions for Python identifiers:

- Class names start with an **UPPERCASE** letter. All other identifiers start with a **lowercase** letter(Alphabets).
- Starting an identifier with a single leading underscore indicates that the identifier is private.
- Starting an identifier with two leading underscores indicates a strongly private identifier.
- If the identifier also ends with two trailing underscores, the **identifier is a language - defined special name**.

## Examples:

These Work:

X, x, \_WheresThePartyTonight, Bond007

And these Don't:

007Bond, \_WheresThePartyTonight?

Don't believe me?? Try out some whacky identifiers

```
>>> 007Bond = 20
```

**SyntaxError: invalid token**

# Reserved words

➤ Words which cannot be used to name any data identifier are reserved words.

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

“We will know about them later. But, why don’t you try using them none the less and check what happens? 😊”

For e.g try this :-

```
>>global = 1
```

# Comments

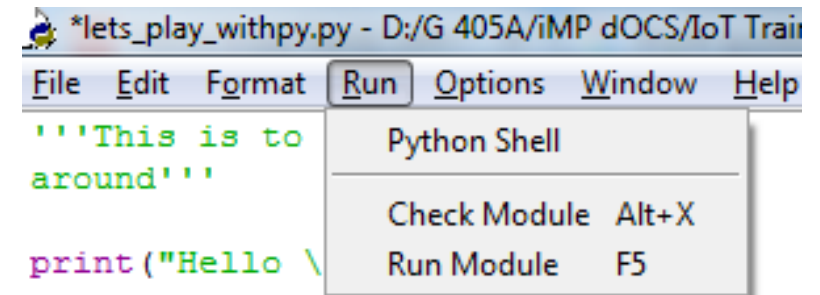
- Pass comments, loud and clear, and probably those rare occasions when you can do so without fear !!
- Helps make the code readable and understandable.
- Easy to reuse code base.
- Considered one of the ***Best Practice*** while writing code.
- Comments in Python start with the hash character, #, and extend to the end of the physical line.

## Example of comments:

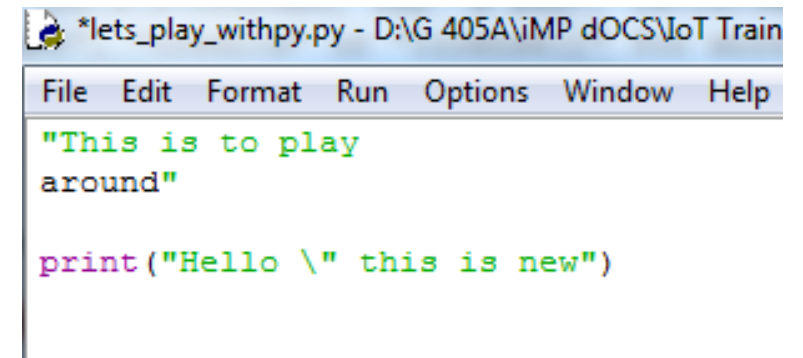
- # this is the first comment
  - SPAM = 1 # and this is the second comment
  - # ... and now a third!
  - STRING = "# This is not a comment."
- 
- ''' (triple quotes) serves as multi-line comment. It can be used to generate d

# Comments (Cntd..)

```
Python 3.6.2 Shell
File Edit Shell Debug Options Window Help
Python 3.6.2 (v3.6.2:5fd33b5, Jul 8 2017, 04:14:34) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> "Hello world '"
SyntaxError: EOL while scanning string literal
>>> 'hello''
'hello'
>>> python dataanalysis
SyntaxError: invalid syntax
>>> python dataanalysis.py
SyntaxError: invalid syntax
>>> dataanalysis.py
Traceback (most recent call last):
  File "<pyshell#4>", line 1, in <module>
    dataanalysis.py
NameError: name 'dataanalysis' is not defined
>>>
== RESTART: D:\G 405A\iMP dOCS\IoT Training\Python programs\dataanalysis.py ==
Traceback (most recent call last):
  File "D:\G 405A\iMP dOCS\IoT Training\Python programs\dataanalysis.py", line 1, in <module>
    import pandas as pd
ModuleNotFoundError: No module named 'pandas'
>>>
RESTART: D:/G 405A/iMP dOCS/IoT Training/Python programs/lets_play_withpy.py
Hello " this is new
>>>
RESTART: D:/G 405A/iMP dOCS/IoT Training/Python programs/lets_play_withpy.py
Hello " this is new
>>>
```

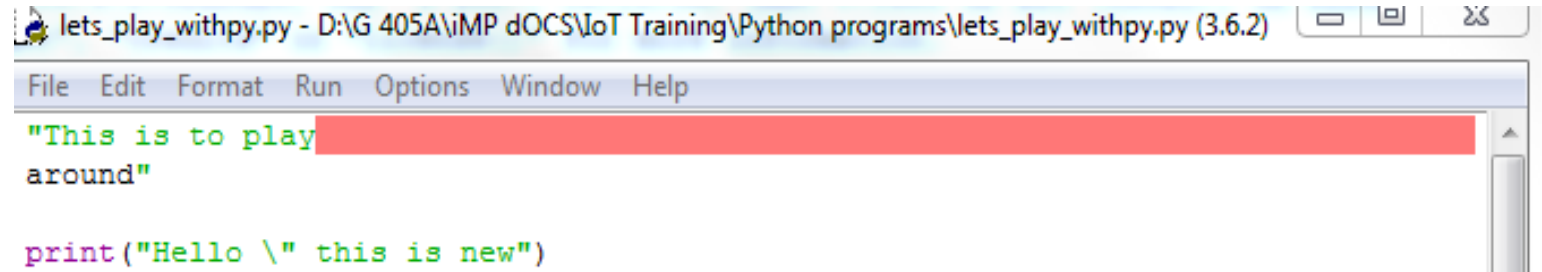


The screenshot shows a Python Shell window titled '\*lets\_play\_withpy.py - D:/G 405A/iMP dOCS/IoT Train'. The menu bar includes File, Edit, Format, Run, Options, Window, and Help. The code in the editor is: `'''This is to around'''` and `print("Hello \`. A context menu is open over the `Run` button, showing options: Python Shell, Check Module (Alt+X), and Run Module (F5).



The screenshot shows the same Python Shell window after execution. The code in the editor is: `"This is to play around"` and `print("Hello \" this is new")`. The output in the shell is: `Hello " this is new`.

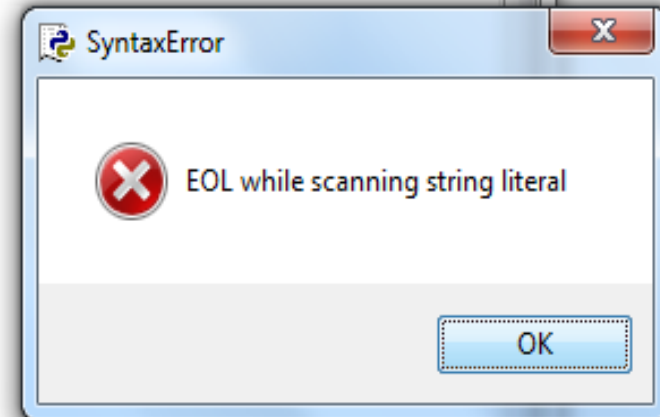
# Comments (Cntd..)



The screenshot shows a Python IDE window titled 'lets\_play\_withpy.py - D:\G 405A\IMP dOCS\IoT Training\Python programs\lets\_play\_withpy.py (3.6.2)'. The menu bar includes File, Edit, Format, Run, Options, Window, and Help. The code editor contains the following Python code:

```
"This is to play  
around"  
  
print("Hello \" this is new")
```

The first line of the string is highlighted in red, indicating a syntax error.

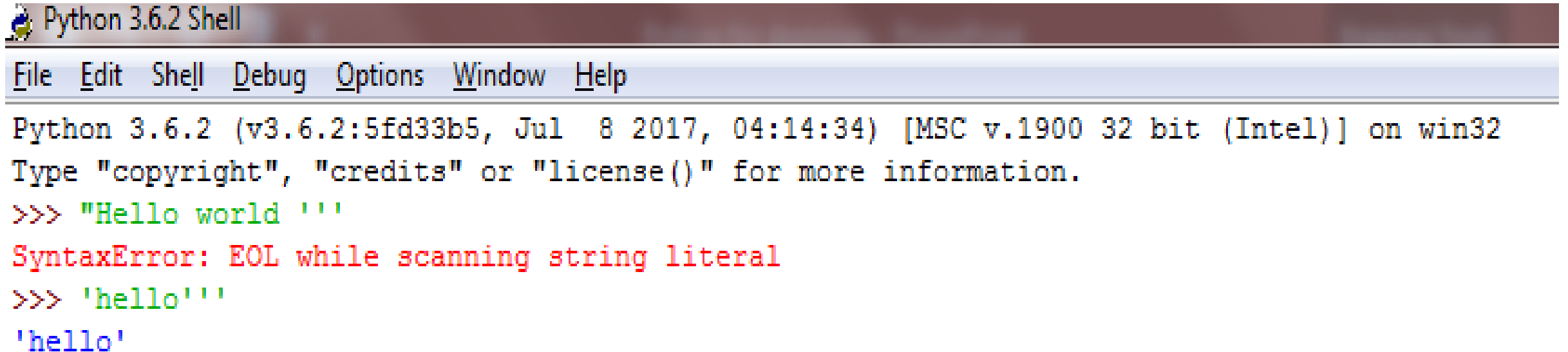




# Common PYTHON Syntax Colours

Common Python syntax colors:

- Keywords      Orange
- Strings        Green
- Comments      Red
- Definitions    Blue
- Misc. Words    Black



```
Python 3.6.2 Shell
File Edit Shell Debug Options Window Help
Python 3.6.2 (v3.6.2:5fd33b5, Jul 8 2017, 04:14:34) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> "Hello world '"
SyntaxError: EOL while scanning string literal
>>> 'hello''
'hello'
```

# Code Blocks & Line Indentation

- Blocks of code are denoted by line indentation. No braces or any keywords to indicate blocks of code for class and function definitions or flow control.
- The number of spaces in the indentation is variable, but all statements within the block must be indented by the same amount.

## Correct

```
>>> if True:  
    print(x)  
else:  
    print("False")
```

## Wrong

“Try out a line of code without indentation and observe the output”.

# Multiline Statements

- Code blocks spreading over to multiple lines.
- Deal with them by putting a “\” at the end of each line.
- Statements contained within brackets( [],{},( )) don't need multi-line breakers.

```
_>>> My_Name = "Diptarko \
```

```
Das \
```

```
Sarma,,
```

```
>>> print(My_Name)
```

```
Diptarko
```

```
Das
```

```
Sarma
```

# Food for thought

- Name some of the features of Python.
- What is the purpose of PATH environment variable?
- Is python a case sensitive language?

# PYTHON LEXICON

**Stimulants**

# Python Lexicon

## Stimulants

### 1. Is Python case sensitive when dealing with identifiers?

- a) yes
- b) no
- c) machine dependent
- d) none of the mentioned

### 2. What is the maximum possible length of an identifier?

- a) 31 characters
- b) 63 characters
- c) 79 characters
- d) none of the mentioned

### 3. Which of the following is invalid?

- a) `_a = 1`
- b) `__a = 1`
- c) `__str__ = 1`
- d) none of the mentioned

# Python Lexicon

## Stimulants

**4. Which of the following is an invalid variable?**

- a) my\_string\_1
- b) 1st\_string
- c) foo
- d) \_

**5. Why are local variable names beginning with an underscore discouraged?**

- a) they are used to indicate a private variables of a class
- b) they confuse the interpreter
- c) they are used to indicate global variables
- d) they slow down execution

**6. All keywords in Python are in**

- a) lower case
- b) UPPER CASE
- c) Capitalized
- d) None of the mentioned



## **Stimulants**

### **7. Which of the following is true for variable names in Python?**

- a) unlimited length
- b) all private members must have leading and trailing underscores
- c) underscore and ampersand are the only two special characters allowed
- d) none of the mentioned

### **8. Which of the following is an invalid statement?**

- a) `abc = 1,000,000`
- b) `a b c = 1000 2000 3000`
- c) `a,b,c = 1000, 2000, 3000`
- d) `a_b_c = 1,000,000`

### **9. Which of the following cannot be a variable?**

- a) `__init__`
- b) `in`
- c) `it`
- d) `on`



# PYTHON DATA TYPES

# Python Data Types – An Introduction

Python supports the following data types:

- Numbers
- String
- Boolean
- List
- Tuples
- Set
- Dictionary

# Python Numbers

- Any variable which stores **Numeric** values are numeric datatypes.

Example of numerics:

10 100.000 51924361L 9.322e-36j

- Whenever you assign a value to the Numeric datatype, it creates a Numeric object.

Example,

```
>>>var1=10
```

```
type(var1)
```

```
<class 'int'>
```

Numeric datatypes are **immutable**. This means changing the value of a numeric variable actually results in the creation of a new object.

## Lets try and learn

```
>>> var1 = 10
```

```
>>> id(var1)
```

```
505702672
```

```
>>> var1=20
```

```
>>> id(var1)
```

```
505702832
```

# Python Numbers (Cntd..)

Deleting a Python Number Type:

- To delete the reference to a number object using the del statement.
- Whenever we use del, the reference to the numeric object is deleted. Example below.

```
>>> var1 = 10.0
```

```
>>> type(var1)
```

```
<class 'float'>
```

```
>>> print(var1)
```

```
10.0
```

```
>>> del var1
```

```
>>> type(var1)
```

Traceback (most recent call last):

```
File "<pyshell#5>", line 1, in <module>
```

```
    type(var1)
```

```
NameError: name 'var1' is not defined
```

# Python Numbers: Different NUMERIC TYPES

Python supports four different numerical types:

- **int (signed integers)**: They are often called just integers or ints, are positive or negative whole numbers with no decimal point.
  - Octal Ex: 0o12
  - Hexadecimal Ex: 0xF
- **float (floating point real values)** : Also called floats, they represent real numbers and are written with a decimal point dividing the integer and fractional parts. Floats may also be in scientific notation, with E or e indicating the power of 10 ( $2.5e2 = 2.5 \times 10^2 = 250$ ).
- **complex (complex numbers)** : are of the form  $a + bJ$ , where  $a$  and  $b$  are floats and  $J$  (or  $j$ ) represents the square root of  $-1$  (which is an imaginary number). The real part of the number is  $a$ , and the imaginary part is  $b$ . Complex numbers are not used much in Python programming.

# Python Numbers (Cntd..)

int	float	complex
10	0.0	3.14j
100	15.20	45.j
-786	-21.9	9.322e-36j
080	32.3+e18	.876j
-0490	-90.	-.6545+0J
-0x260	-32.54e100	3e+26J
0x69	70.2-E12	4.53e-7j

- A complex number consists of an ordered pair of real floating point numbers denoted by  $a + bj$ , where  $a$  is the real part and  $b$  is the imaginary part of the complex number.

# Python Numbers

## ***Complex Number Mathematics***

```
>>> 3.14j
3.14j
>>> (0+1j) * (0+1j)
(-1+0j)
>>> 1j*(0+1j)
(-1+0j)
>>> 3+1j*3
(3+3j)
>>> (3+1j)*3
(9+3j)
>>> a=1.5+0.5j
>>> a.real
1.5
```



# Python Numbers – Mathematical Functions

Sl. No.	Mathematical Functions
1	<a href="#"><u>abs(x)</u></a> The absolute value of x: the (positive) distance between x and zero.[Function]
2	<a href="#"><u>ceil(x)</u></a> The ceiling of x: the largest integer greater than x.[math method]
3	<code>cmp(x, y)</code> -1 if $x < y$ , 0 if $x == y$ , or 1 if $x > y$ . Deprecated in Python 3. Instead use <code>return (x&gt;y)-(x&lt;y)</code> .
4	<a href="#"><u>exp(x)</u></a> The exponential of x: $e^x$ [math method]
5	<a href="#"><u>fabs(x)</u></a> The absolute value of x.[math method]
6	<a href="#"><u>floor(x)</u></a> The floor of x: the smallest integer not greater than x.[math method]
7	<a href="#"><u>log(x)</u></a> The natural logarithm of x, for $x > 0$ . [math method]
8	<a href="#"><u>log10(x)</u></a> The base-10 logarithm of x for $x > 0$ . [math method]
9	<a href="#"><u>max(x1, x2,...)</u></a> The largest of its arguments: the value closest to positive infinity
10	<a href="#"><u>min(x1, x2,...)</u></a> The smallest of its arguments: the value closest to negative infinity.
11	<a href="#"><u>modf(x)</u></a> The fractional and integer parts of x in a two-item tuple. Both parts have the same sign as x. The integer part is returned as a float.

# Python Numbers – Mathematical Functions (Cntd..)

Sl. No.	Mathematical Functions
12	<a href="#"><code>pow(x, y)</code></a> The value of $x^{**}y$ .
13	<a href="#"><code>round(x [,n])</code></a> $x$ rounded to $n$ digits from the decimal point. Python rounds away from zero as a tie-breaker: <code>round(0.5)</code> is 1.0 and <code>round(-0.5)</code> is -1.0.
14	<a href="#"><code>sqrt(x)</code></a> The square root of $x$ for $x > 0$ .

Python converts numbers internally in an expression containing mixed types to a common type for evaluation.

But sometimes, you need to coerce a number explicitly from one type to another to satisfy the requirements of an operator or function parameter.

- Type **`int(x)`** to convert  $x$  to a plain integer.
- Type **`long(x)`** to convert  $x$  to a long integer.
- Type **`float(x)`** to convert  $x$  to a floating-point number.
- Type **`complex(x)`** to convert  $x$  to a complex number with real part  $x$  and imaginary part zero.
- Type **`complex(x, y)`** to convert  $x$  and  $y$  to a complex number with real part  $x$  and imaginary part  $y$ .  $x$  and  $y$  are numeric expressions

# Python Numbers: Mathematical Operators

Arithmetic Operators	Comparison Operators	Logical Operators	Assignment Operators	Bitwise Operators	Membership Operators and Identity Operators
+	>	and	=	&	in
-	<	or	+=		not in
*	>=	not	-=	^	is
/	<=		/=	~	is not
%	==		*=	>>	
**	!=			<<	
//					

# Python Numbers: Operator Precedence

Operator	Description
**	(raise to the power) ~ Unlike other operators has a right to left associativity.
~ + -	Complement Unary plus Unary minus (method names for the last two are +@ and -@)
* / % //	Multiply Divide Modulo Floor division
+ -	Addition Subtraction
>> <<	Right bitwise shift Left bitwise shift
&	Bitwise 'AND'
^ 	Bitwise exclusive 'OR' Regular 'OR'

# Python Numbers: Operator Precedence (Cntd..)

Operator	Description
<=	Comparison operator – Less than or equal to
<	Comparison operator – Less than
>	Comparison operator – Greater than
>=	Comparison operator – Greater than or equal to
==	Equality operators
!=	
=	Assignment operators
%=	
/=	
//=	
-=	
+=	
*=	
**=	
is	Identity operators
is not	
in	Membership operators
not in	
not or and	Logical operators © DIPTARKO DAS SHARMA

# PYTHON NUMERICS

## Stimulants

## Stimulants

1. What is the output of `print( 0.1 + 0.2 == 0.3)`?

- a) True
- b) False
- c) Machine dependent
- d) Error

2. Which of the following is not a complex number?

- a)  $k = 2 + 3j$
- b) `k = complex(2, 3)`
- c)  $k = 2 + 3I$
- d)  $k = 2 + 3J$

### Stimulants

3. The value of the expressions  $4/(3*(2-1))$  and  $4/3*(2-1)$  is the same. State whether true or false.

- a) True
- b) False

4. What is the value of x if:

$x = \text{int}(43.55 + 2/2)$

- a) 43
- b) 44
- c) 22
- d) 23



## Stimulants

5. Which of the following is incorrect?

- a) `float('inf')`
- b) `float('nan')`
- c) `float('56'+ '78')`
- d) `float('12+34')`

6. What does  $(1>3)-(1<3)$  evaluate to?

- a) -1
- b) 1
- c) 0
- d) Unknown

### Stimulants

7. What are the values of the following expressions:

$2^{(3^2)}$

$(2^3)^2$

$2^3 \cdot 2^2$

a) 64, 512, 64

b) 64, 64, 64

c) 512, 512, 512

d) 512, 64, 512

8. What is the value of the following expression:

$\text{float}(22//3+3/3)$

a) 8

b) 8.0

c) 8.3

d) 8.33

# Python Strings

- A group of characters strung together into a word,sentence, line or even paragraph is a string

“Hi” “Hi There” “Yo Man!! Oppa Gangnam Style”

- Strings are amongst the most popular types in Python. We can create them simply by enclosing characters in quotes. Python treats single quotes the same as double quotes. Creating strings is as simple as assigning a value to a variable.

## For example:

```
var1 = 'Hello World!'
var2 = "Python Programming"
>>>word = 'word'
>>>sentence = "This is a sentence."
>>>paragraph = """This is a \
paragraph \
across \
multiple lines."""
>>> type("Hello World")
<class 'str'>
```

# Python Strings (Cntd..)

## Try and Learn:

Type out the code, observe the output and lets try to understand the logic

```
var1 = 'Hello World!'
var2 = "Python Programming"
print ("var1[0]: ", var1[0])
print ("var2[1:5]: ", var2[1:5])
```

Lets try a few more, lets bring it on!!

- `print("Hello Students"*2)`
- `var1 = "Python Programming"`
- `print(var1[-5:-1]+'Abdracadabra')`

# Python Strings – Indexing & Slicing

## STRING INDEXING & SLICING

- Remember arrays? A string is an array of characters.
- Just like an array, you do have indexes for strings.
- Python provides special tools to create substrings out of strings, it is called slicing.
- Let's do and learn Slicing, we have already seen a few scenarios on the last slide.

```
>>>str = 'Hello World!'
```

```
>>>str # Prints complete string
```

```
>>>str[0] # Prints first character of the string
```

```
>>>str[2:5] # Prints characters starting from 3rd(Included) to 5th(excluded)
```

```
>>>str[2:] # Prints string starting from 3rd character
```

```
>>> str[::-1] # Prints the string in reverse order, step size of -1.
```

```
>>>str[-1] # Prints the last item from the end
```

```
>>>str * 2 # Prints string two times
```

```
>>>str + "TEST" # Prints concatenated string
```

# STRING OPERATIONS

OPERATOR	DESCRIPTION	EXAMPLE
+	Concatenation - Adds values on either side of the operator	a + b will give HelloPython
*	Repetition - Creates new strings, concatenating multiple copies of the same string	a*2 will give -HelloHello
[]	Slice - Gives the character from the given index	a[1] will give e
[ : ]	Range Slice - Gives the characters from the given range	a[1:4] will give ell
in	Membership - Returns true if a character exists in the given string	'H' in a will give 1
not in	Membership - Returns true if a character does not exist in the given string	'M' not in a will give 1

# STRING METHODS

STRINGS are classes.They come inbuilt with lots of useful methods.Lots of string editing stuff becomes easy to carry out because of these inbuilt methods.So here are a few important string methods .

```
sTr = 'Python'
```

Len : Calculate the length of a string

```
>> len(sTr)
```

```
6
```

# STRING METHODS

**Replace :** The `replace()` method replaces a specified phrase with another specified phrase.

**Syntax :** `string.replace(oldvalue, newvalue, count)`

Parameter	Description
<i>Oldvalue</i>	Required. The string to search for
<i>newvalue</i>	Required. The string to replace the old value with
<i>Count</i>	Optional. A number specifying how many occurrences of the old value you want to replace. Default is all occurrences

**Usage :**

```
>>> txt = "one one was a race horse, two two was one too."
```

```
>>> txt.replace('one','three')
```

```
'three three was a race horse, two two was three too.'
```



# STRING METHODS

**Find** : Find the first occurrence of an element in the string. Returns the index position of the first occurrence, if the element is found, else returns -1 if not found. The `find()` method is almost the same as the `index()` method, the only difference is that the `index()` method raises an exception if the value is not found.

**Syntax** : `string.find(value, start, end)`

Parameter	Description
<i>value</i>	Required. The value to search for
<i>start</i>	Optional. Where to start the search. Default is 0
<i>end</i>	Optional. Where to end the search. Default is to the end of the string.

**Usage** : Where is the first occurrence of the letter 'e'.

```
>>> txt = "Hello, welcome to my world."
```

```
>>> x = txt.find("e")
```

```
>>> x
```

# STRING METHODS

**Isalnum()**: Check if all the characters in the text is alphanumeric. The `isalnum()` method returns True if all the characters are alphanumeric, meaning alphabet letter (a-z) and numbers (0-9).

Example of characters that are not alphanumeric: (space)!#%&? etc.

**Syntax** : ***string.isalnum()***

Parameters : No parameters.

Usage :

```
>>> txt = "Company 12"
```

```
>>> x = txt.isalnum()
```

```
>>> x
```

False

```
>>> txt = "Company12"
```

```
>>> x = txt.isalnum()
```

```
>>> x
```

True

# STRING METHODS

**Split** : Split a string into a list where each word is a list item. You can specify the separator, default separator is any whitespace.

Note : When ***maxsplit*** is specified, the list will contain the specified number of elements plus one.

**Syntax** : `string.split(separator,max)`

Parameter	Description
<i>separator</i>	Optional. Specifies the separator to use when splitting the string. Default value is a whitespace
<i>maxsplit</i>	Optional. Specifies how many splits to do. Default value is -1, which is "all occurrences"

## **Usage** :

```
>>> txt = "hello, my name is Peter, I am 26 years old"
>>> x = txt.split(", ")
>>> x
['hello', 'my name is Peter', 'I am 26 years old']
>>> x = txt.split(", ",maxsplit = 1)
>>> x
['hello', 'my name is Peter, I am 26 years old']
```

# STRING METHODS

Join all items in a tuple into a string, using a hash character as separator. The `join()` method takes all items in an iterable and joins them into one string.

The `join()` method takes all items in an iterable and joins them into one string.

Syntax : ***string.join(iterable)***

Parameter	Description
<i>iterable</i>	Required. Any iterable object where all the returned values are strings

Note: When using a dictionary as an iterable, the returned values are the keys, not the values.

## Usage :

```
>>> myTuple = ("John", "Peter", "Vicky")
```

```
>>> "#".join(myTuple)
'John#Peter#Vicky'
```

# Python Strings – String Operations

## A few String Operations & String Methods

```
>>>s.upper() # Change to upper case
```

```
PYTHON
```

```
>>>s='aaa,bbb,ccc,dd'
```

```
>>>s.split(",") # Split the string into parts using ',' as delimiter
```

```
['aaa','bbb','ccc','dd']
```

```
>>>s.isalpha() # Content tests: isalpha, isdigit, etc.
```

```
False
```

```
>>>s = 'aaa,bbb,ccc,dd \n'
```

```
>>>s.rstrip() # Remove whitespace characters on the right
```

```
aaa,bbb,cccc,dd
```

```
>>> s.startswith("a") # Check if the string starts with 'a'
```

```
True
```

```
>>> s.endswith("c") # Check if the string ends with 'c'
```

```
False
```

# Python Strings – Raw Strings & String Formatting operators

## Raw Strings

A “raw” string will not convert \n sequences to newlines. A raw string precedes with ‘r’

```
>>> print('C:\some\name')
```

```
C:\some
```

```
ame
```

```
>>> print(r'C:\some\name')
```

```
C:\some\name
```

## String Formatting Operators

One of Python's coolest features is the string format operator %. This operator is unique to strings and makes up for the pack of having functions from C's printf() family. Following is a simple example –

```
>>print ("My name is %s and weight is %d kg!" % ('Zara', 21))
```

Can you check what happens when you print the details?

***Strings are immutable .Strings are read only***

# Python Strings – Triple Quotes

## Triple Quotes

- Python's triple quotes comes to the rescue by allowing strings to span multiple lines, including verbatim NEWLINEs, TABs, and any other special characters.
- The syntax for triple quotes consists of three consecutive single or doublequotes.

```
para_str = """this is a long string that is made up of  
several lines and non-printable characters such as  
TAB ( \t ) and they will show up that way when displayed.  
NEWLINEs within the string, whether explicitly given like  
this within the brackets [ \n ], or just a NEWLINE within  
the variable assignment will also show up.
```

```
"""
```

```
print(para_str)
```

When the above code is executed, it produces the following result. Note how every single special character has been converted to its printed form, right down to the last NEWLINE at the end of the string between the "up." and closing triple quotes. Also note that NEWLINEs occur either with an explicit carriage return at the end of a line or its escape code (\n) –

# Python Strings – Escape Characters

## Escape Characters

Backslash notation	Hexadecimal character	Description
\a	0x07	Bell or alert
\b	0x08	Backspace
\cx		Control-x
\C-x		Control-x
\e	0x1b	Escape
\f	0x0c	Formfeed
\M-\C-x		Meta-Control-x
\n	0x0a	Newline
\nnn		Octal notation, where n is in the range 0-7
\r	0x0d	Carriage return



# Python Strings – Escape Characters

## Escape Characters

Backslash notation	Hexadecimal character	Description
\s	0x20	Space
\t	0x09	Tab
\v	0x0b	Vertical tab
\x		Character x
\xnn		Hexadecimal notation, where n is in the range 0-9, a-f, or A-F

# PYTHON STRINGS

## STIMULANTS

## **Stimulants**

### **1. What is the output when following statement is executed ?**

```
>>>"a"+"bc"
```

- a) a
- b) bc
- c) bca
- d) abc

### **2. What is the output when following statement is executed ?**

```
>>>"abcd"[2:]
```

- a) a
- b) ab
- c) cd
- d) dc

### **3. What is the output when following code is executed ?**

```
>>> str1 = 'hello'; >>> str1[-1:]
```

- a) olleh
- b) hello
- c) h
- d) o

## **Stimulants**

### **4. What arithmetic operators cannot be used with strings ?**

- a) +
- b) \*
- c) –
- d) All of the mentioned

### **5. What is the output when following code is executed ?**

```
>>>print (r"\nhello")
```

The output is

- a) a new line and hello
- b) \nhello
- c) the letter r and then hello
- d) Error

### **6. What is the output when following statement is executed ?**

```
>>>print('new' 'line')
```

- a) Error
- b) Output equivalent to print 'new\nline'
- c) newline
- d) new line

## **Stimulants**

### **7. What is the output when following statement is executed ?**

```
>>>print('new' '\t' 'line')
```

- a) Error
- b) Output equivalent to print 'new\nline'
- c) newline
- d) new     line

### **8. print(0xA + 0xB + 0xC) :**

- a) 0xA0xB0xC
- b) Error
- c) 0x22
- d) 33

### **9. What is the output of the following code ?**

```
>>>example = "snow world"
```

```
>>>example[3] = 's'
```

```
>>>print (example)
```

- a) snow
- b) snow world
- c) Error
- d) snos world

## Stimulants

10. What is the output of the following code ?

```
>>>example = "helle"
```

```
>>>example.find("e")
```

a) Error

b) -1

c) 1

d) 0

11. To concatenate two strings what statements are applicable ?

a) `s3 = s1 . s2`

b) `s3 = s1+s2`

c) `s3 = s1.__add__(s2)`

d) `s3 = s1 * s2`

# Python Lists

- Think of a Fruit basket, containing different types of fruits, something like the one shown below :-



- It's an assortment of fruits, isn't it?
- Similarly, **sequence** is a sort of data structure, which keeps different datatypes within it.
- Each element of a sequence is assigned a number - its position or index. The first index is zero, the second index is one, and so forth.
- Python has six built-in types of sequences, but the most common ones are, **lists and tuples**.
- There are certain things you can do with all the sequence types. These operations include indexing, slicing, adding, multiplying, and checking for membership. In addition, Python has built-in functions for finding the length of a sequence and for finding its largest and smallest elements.
- The list is the most versatile datatype available in Python, which can be written as a list of comma-separated values (items) between square brackets. Important thing about a list is that the items in a list need not be of the same type.

# Python Lists(Contd)

- Creating a list is as simple as putting different comma-separated values between square brackets. For example –

```
list1 = ['physics', 'chemistry', 45, 40.2];
```

```
list2 = [1, 2, 3, 4, 5 ];
```

```
list3 = ["a", "b", "c", "d"];
```

- Similar to string indices, list indices start at 0, and lists can be sliced, concatenated and so on.



# Python Lists –Accessing values in Lists

To access values in lists, use the square brackets for slicing along with the index or indices to obtain value available at that index. For example –

## **Lets try it out**

```
>>> list1 = ['physics', 'chemistry', 45, 40.2]
>>> list2 = [1, 2, 3, 4, 5, 6, 7 ]
>>> print ("list1[0]: ", list1[0])
>>> print ("list2[1:5]: ", list2[1:5])
>>> list = [ 'abcd', 786 , 2.23, 'john', 70.2 ]
>>> tinylist = [123, 'joy']
>>> list # Prints complete list
>>> list[0] # Prints first element of the list
>>> list[1:3] # Prints elements starting from 2nd till 3rd
>>> list[2:] # Prints elements starting from 3rd element
>>> list[:3] # Prints elements starting from beginning till 3rd
>>> list[1:-1] # Prints all elements except the first and last
>>> tinylist * 2 # Prints list two times
>>> list + tinylist # Prints concatenated lists
>>> len(list) # Prints length of the list
```

# Python Lists- Updating and Deleting from Lists

- Unlike the other datatypes we played with so far, lists are mutable. Which means, we can actually change the value of lists .

```
>>>list = [ 'abcd', 786 , 2.23, 'john', 70.2 ]
```

```
>>> list[0] = 111 #Modify the value of the 0th element
```

```
>>> list[1:3]=['Sam',2000,300.89] #Modifies the value of 1st to 3rd element of the list.
```

```
>>> list[1:3]=[] #Deletes from 1st to the 3rd element, but doesn't take the 3rd element.
```

```
>>> list[:] =[] #Clears complete list
```

# Python Lists - Nesting Lists

It is possible to nest lists (create lists containing other lists), for example:

```
>>> q = [2, 3]
```

```
>>> p = [1, q, 4]
```

```
>>> len(p)
```

```
3
```

```
>>> p[1]
```

```
[2, 3]
```

```
>>> p[1][0]
```

```
2
```

# Python Lists – Adding to lists

➤ Method 1 : Using list concatenation

```
>>> a=a+[10,20]
```

This will create a second list in memory which can (temporarily) consume a lot of memory when you're dealing with large lists

➤ Method 2 : Using append

Append takes a single argument(any datatype) and adds to the end of list

```
>>>list1=[10,20,30]
```

```
>>>list1.append('new')
```

```
>>> list1
```

```
[10,20,30,'new']
```

```
>>> list1.append([1,2,3])
```

```
>>>list1
```

```
[10,20,30,'new',[1,2,3]]
```

# Python Lists – Adding to lists

## ➤ Method 3 : Using extend

extend takes a single argument(list),and adds each of the items to the list

```
>>>a=[10,20,30]
```

```
>>>a.extend([1,2,3])
```

```
>>>a
```

```
[10,20,30,1,2,3]
```

## ➤ Method 4 :Using insert

**Insert can be used to insert an item in the desired place**

```
>>>a=[10,20,30]
```

```
>>>a.insert(0,'new')
```

```
>>> a
```

```
['new',10,20,30]
```

```
>>>a.insert(100,'python')
```

```
>>> a
```

```
['new',10,20,30,'python']
```

# Python Lists – Deletion in lists

- We have already discussed deletion of lists in one of the sections before
- Deletion was done by selecting the exact index which we wanted to delete.
- For e.g `a[1,2,3]` is a list, and to delete the second element of the list we simply used the statement,  
`a[1]=[]`
- There is also a built in method called `remove()`, which can be used to remove elements from a list.
- Here's how we can use this Superpower 😊
- **`list.remove(obj)` : Removes an object from the list**

```
>>> a = [-1, 1, 66.25, 333, 333, 1234.5]
```

```
>>> a.remove(200)
```

Traceback (most recent call last):

```
File "<pyshell#4>", line 1, in <module>
```

```
a.remove(200)
```

ValueError: list.remove(x): x not in list

```
>>> a.remove(333) # removes the first matching value
```

```
>>> print(a) [-1, 1, 66.25, 333, 1234.5]
```

# Python Lists – A few more list methods(Contd)

➤ **Count** : Returns count of how many times an object exists in the list.

➤ `a = [66.25, 333, 333, 1, 1234.5]`

`print(a.count(333), a.count(66.25), a.count('x'))`

➤ **Index** : Returns the index of the occurrence of the first given value in the list

```
>>> a=[1,2,2,3]
```

```
>>> a.index(2)
```

```
1
```

```
>>> a.index(500) #Returns ValueError if not in list.
```

Traceback (most recent call last):

```
File "<pyshell#19>", line 1, in <module>
```

```
    a.index(500)
```

```
ValueError: 500 is not in list
```

➤ **Reverse** : **Reverses objects of list in place**

```
>>> a.reverse()
```

```
>>> a
```

```
[3, 2, 2, 1]
```

# Python Lists – Comprehensions

```
>>> squares=[]  
>>> for i in (range(10)):  
    squares.append(i**2)  
>>> print(i)  
9  
>>> squares  
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

**Now, all these lines can be smartly and swiftly done away with, to produce the same effect. That's the elegance and simplicity of Python!!**

```
>>> squares =[i**2 for i in (range(10))]  
>>> squares  
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

```
New_list = [iterator for iterator in iterable]
```





# Python Lists – Summary(Methods & Functions)

➤ A summary of the major list **functions (Try and Learn mode):-**

Method	Description
list.append(obj)	Appends object <b>obj</b> to list
list.count(obj)	Returns count of how many times obj occurs in list
list.extend(seq)	Appends the contents of seq to list
list.index(obj)	Returns the lowest index in list that obj appears
list.pop(listindex)	Returns the removed element whose index is given
list.remove(obj)	Removes object <b>obj</b> from list
list.reverse()	Reverses objects of list in place
list.sort()	Sorts objects inside a list

Function	Description
len(list)	Gives the total length of the list.
max(list)	Returns item from the list with max value.
min(list)	Returns item from the list with min value.
list(seq)	Converts a tuple into list.

# PYTHON STRINGS

## STIMULANTS

# Python List

## Stimulants

1. **Suppose listExample is ['h','e','l','l','o'], what is len(listExample)?**
  - a) 5
  - b) 4
  - c) None
  - d) Error
2. **Suppose list1 is [2445,133,12454,123], what is max(list1) ?**
  - a) 2445
  - b) 133
  - c) 12454
  - d) 123
3. **Suppose list1 is [3, 5, 25, 1, 3], what is min(list1) ?**
  - a) 3
  - b) 5
  - c) 25
  - d) 1
4. **Suppose list1 is [1, 5, 9], what is sum(list1) ?**
  - a) 1
  - b) 9
  - c) 15
  - d) Error

# Python List

## **Stimulants**

**5. Suppose list1 is [2, 33, 222, 14, 25], What is list1[-1] ?**

- a) Error
- b) None
- c) 25
- d) 2

**6. Suppose list1 is [2, 33, 222, 14, 25], What is list1[:-1] ?**

- a) [2, 33, 222, 14].
- b) Error
- c) 25
- d) [25, 14, 222, 33, 2].

**7. What is the output when following code is executed ?**

```
>>>names = ['Amir', 'Bear', 'Charlton', 'Daman']
```

```
>>>print(names[-1][-1])
```

- a) A
- b) Daman
- c) Error
- d) n

# Python List

## **Stimulants**

**5. Suppose list1 is [2, 33, 222, 14, 25], What is list1[-1] ?**

- a) Error
- b) None
- c) 25
- d) 2

**6. Suppose list1 is [2, 33, 222, 14, 25], What is list1[:-1] ?**

- a) [2, 33, 222, 14].
- b) Error
- c) 25
- d) [25, 14, 222, 33, 2].

**7. What is the output when following code is executed ?**

```
>>>names = ['Amir', 'Bear', 'Charlton', 'Daman']
```

```
>>>print(names[-1][-1])
```

- a) A
- b) Daman
- c) Error
- d) n

# Python List

## Stimulants

8. What is the output when following code is executed ?

```
names1 = ['Amir', 'Bear', 'Charlton', 'Daman']
names2 = names1
names3 = names1[:]
names2[0] = 'Alice'
names3[1] = 'Bob'
sum = 0
for ls in (names1, names2, names3):
    print(ls)
    if ls[0] == 'Alice':
        sum += 1
    if ls[1] == 'Bob':
        sum += 10
print( sum)
```

- a) 11
- b) 12
- c) 21
- d) 22

# Python List

## **Stimulants**

**9. What is the output when following code is executed ?**

```
>>>list1 = [11, 2, 23]
```

```
>>>list2 = [11, 2, 2]
```

```
>>>list1 < list2 is
```

- a) True
- b) False
- c) Error
- d) None

**10. To add a new element to a list we use which command ?**

- a) list1.add(5)
- b) list1.append(5)
- c) list1.addLast(5)
- d) list1.addEnd(5)

**11. To insert 5 to the third position in list1, we use which command ?**

- a) list1.insert(3, 5)
- b) list1.insert(2, 5)
- c) list1.add(3, 5)
- d) list1.append(3, 5)

# Python List

## **Stimulants**

**12. What is the output when following code is executed ?**

```
>>>list1 = [11, 2, 23]
>>>list2 = [11, 2, 2]
myList = [1, 5, 5, 5, 5, 1]
max = myList[0]
indexOfMax = 0
for i in range(1, len(myList)):
    if myList[i] > max:
        max = myList[i]
        indexOfMax = i
>>>print(indexOfMax)
```

- a) 1
- b) 2
- c) 3
- d) 4



# Python List

## Stimulants

**13. To which of the following the “in” operator can be used to check if an item is in it?**

- a) Lists
- b) Dictionary
- c) Set
- d) All of the mentioned

**14. What will be the output?**

```
veggies = ['carrot', 'broccoli', 'potato', 'asparagus']  
veggies.insert(veggies.index('broccoli'), 'celery')  
print(veggies)
```

- a) ['carrot', 'celery', 'potato', 'asparagus'].
- b) ['carrot', 'celery', 'broccoli', 'potato', 'asparagus']
- c) ['carrot', 'broccoli', 'celery', 'potato', 'asparagus'].
- d) ['celery', 'carrot', 'broccoli', 'potato', 'asparagus'].

# Python Tuples

- A tuple is a sequence of immutable Python objects. Tuples are sequences, just like lists. The main difference between the tuples and the lists is that the tuples cannot be changed unlike lists. Tuples use parentheses, whereas lists use square brackets.
- Creating a tuple is as simple as putting different comma-separated values. Optionally, you can put these comma-separated values between parentheses also. For example –

```
>>> tup1 = (1,"new",3.99)
>>> type(tup1)
<class 'tuple'>
```

- Tuples are immutable lists.
- Empty tuple is written as two parentheses with empty values.

```
e:g :-tup2 = ();
```

- To write a tuple containing a single value you have to include a comma, even though there is only one value –

```
e:g :- tup3 =("new",)
```

# Python Tuples – Accessing values and Updating Values

## ➤ Accessing Tuples

As in Strings and lists, you access values in tuples using the square brackets and the indexes. Like strings and lists, you can slice tuples as well.

## Try and learn :

```
>>>tuple = ( 'abcd', 786 , 2.23, 'joy', 70.2 )
>>>tinytuple = (123, 'joe')
>>>tuple # Prints complete list
>>>tuple[0] # Prints first element of the list
>>>tuple[1:3] # Prints elements starting from 2nd till 3rd
>>>tuple[2:] # Prints elements starting from 3rd element
>>>tinytuple * 2 # Prints list two times
>>>tuple + tinytuple # Prints concatenated lists
```

## ➤ Updating Tuples

Tuples are immutable, which means you cannot update or change the values of tuple elements. You are able to take portions of the existing tuples to create new tuples as the following example demonstrates –

# Python Tuples – Accessing values and Updating Values(contd)

## Try and Learn

```
tup1 = (12, 34.56)
```

```
tup2 = ('abc', 'xyz')
```

```
# Following action is not valid for tuples
```

```
# tup1[0] = 100;
```

```
# So let's create a new tuple as follows
```

```
tup3 = tup1 + tup2
```

```
print (tup3)
```

# Python Tuples – Deleting Tuples

- Unlike lists, tuples values cant be deleted.
- Deleting a tuple value immediately creates an error

```
>>> Tup1=(1,2,3)
```

```
>>> Tup1
```

```
(1, 2, 3)
```

```
>>> Tup1[0]=[]
```

```
Traceback (most recent call last):
```

```
File "<pyshell#87>", line 1, in <module>
```

```
Tup1[0]=[]
```

```
TypeError: 'tuple' object does not support item assignment
```

# Python Tuples – Summary(Methods & Functions)

➤ A summary of the major Tuple **methods(Try and Learn mode)** :-

Method	Description
tuple.index(obj)	Returns the lowest index in the tuple that obj appears
tuple.count(obj)	Count of how many times the object occurs in Tuple

➤ A summary of the major list **functions (Try and Learn mode):-**

Function	Description
len(tuple)	Gives the total length of the tuple.
max(tuple)	Returns item from the tuple with max value.
min(tuple)	Returns item from the tuple with min value.
list(tuple)	Converts a tuple into list.

# Python Dictionary



What does this remind you of?

# Python Dictionary

- A **dictionary** is an associative array (also known as hashes). Any key of the **dictionary** is associated (or mapped) to a value. The values of a **dictionary** can be any **Python** data type. So **dictionaries** are unordered key-value-pairs.
- A dictionary key can be almost any Python type, but are usually numbers or strings.
- Values can be any arbitrary Python object.
- Dictionaries are enclosed by curly braces ( { } ) and values can be assigned and accessed using square braces ( [ ] ).
- Each key is separated from its value by a colon (:), the items are separated by commas, and the whole thing is enclosed in curly braces. An empty dictionary without any items is written with just two curly braces, like this: {}.
- **Dictionaries** are Mutable.



# Python Dictionary – Accessing Dictionary Values

- To access dictionary elements, you can use the familiar square brackets along with the key to obtain its value. Following is a simple example –

## Try and Learn

```
dict = {'Jack':'9768999','Jill':'345678'}
```

```
print ("Jack's Number ", dict['Jack'])
```

```
print ("Jill's Number ", dict['Jill'])
```

```
print("Gouri's Number", dict['Gouri'])
```

## Deletion of Dictionaries

- You can either remove individual dictionary elements or clear the entire contents of a dictionary. You can also delete entire dictionary in a single operation.
- To explicitly remove an entire dictionary, just use the **del** statement.

```
del dict['Jack'] #Delete a single element
```

```
dict # Now check the dictionary value
```

```
del dict #Delete the entire dictionary
```

```
dict.clear() #Clear the entire array.
```

# Python Dictionary – Accessing Dictionary Values

And here's how we can get the entire key's and value's from the dictionary.

- `dict.keys()` # print all keys in the dictionary.
- `dict.items()` # Prints the entire key/value pairs from the dictionary.
- `dict.values()` # Returns list of dictionary dict's values

## Merging two dicts

```
>>> d = {'spam': 1, 'eggs': 2, 'cheese': 3} >>> e = {'cheese': 'cheddar', 'aardvark': 'Ethel'}  
>>> d | e  
{'spam': 1, 'eggs': 2, 'cheese': 'cheddar', 'aardvark': 'Ethel'} >>> e | d {'cheese': 3, 'aardvark':  
'Ethel', 'spam': 1, 'eggs': 2}
```

# PYTHON DICTIONARY

**Stimulants**

## **Stimulants**

### **1. Which of the following statements create a dictionary?**

- a) `d = {}`
- b) `d = {"john":40, "peter":45}`
- c) `d = {40:"john", 45:"peter"}`
- d) All of the mentioned

### **2. Read the code shown below carefully and pick out the keys?**

`d = {"john":40, "peter":45}`

- a) "john", 40, 45, and "peter"
- b) "john" and "peter"
- c) 40 and 45
- d) `d = (40:"john", 45:"peter")`

### **3. What will be the output?**

`d = {"john":40, "peter":45}`

`"john" in d`

- a) True
- b) False
- c) None
- d) Error

## Stimulants

### 4. What will be the output?

```
d1 = {"john":40, "peter":45}
```

```
d2 = {"john":466, "peter":45}
```

```
d1 == d2
```

- a) True
- b) False
- c) None
- d) Error

### 5. What will be the output?

```
d = {"john":40, "peter":45}
```

```
print(list(d.keys()))
```

- a) ["john", "peter"].
- b) [{"john":40, "peter":45}].
- c) ("john", "peter")
- d) ("john":40, "peter":45)

## **Stimulants**

**6. Suppose `d = {"john":40, "peter":45}`, what happens when we try to retrieve a value using the expression `d["susan"]`?**

- a) Since "susan" is not a value in the set, Python raises a `KeyError` exception
- b) It is executed fine and no exception is raised, and it returns `None`
- c) Since "susan" is not a key in the set, Python raises a `KeyError` exception
- d) Since "susan" is not a key in the set, Python raises a syntax error

**7. What is the output of the following snippet of code?**

```
>>> a={1:"A",2:"B",3:"C"}
```

```
>>> del a
```

- a) method `del` doesn't exist for the dictionary
- b) `del` deletes the values in the dictionary
- c) `del` deletes the entire dictionary
- d) `del` deletes the keys in the dictionary

**8. What is the output of the following snippet of code?**

```
test = {1:'A', 2:'B', 3:'C'}
```

```
test = {}
```

```
print(len(test))
```

- a) 0
- b) `None`
- c) 3
- d) An exception is thrown

### **Stimulants**

#### **9. What is the output of the following snippet of code?**

```
numbers = {}  
letters = {}  
comb = {}  
numbers[1] = 56  
numbers[3] = 7  
letters[4] = 'B'  
comb['Numbers'] = numbers  
comb['Letters'] = letters  
print(comb)
```

- a) Error, dictionary in a dictionary can't exist
- b) 'Numbers': {1: 56, 3: 7}
- c) {'Numbers': {1: 56}, 'Letters': {4: 'B'}}
- d) {'Numbers': {1: 56, 3: 7}, 'Letters': {4: 'B'}}

### **Stimulants**

**10. What is the output of the following snippet of code?**

```
a={}
a['a']=1
a['b']=[2,3,4]
print(a)
```

- a) Exception is thrown
- b) {'b': [2], 'a': 1}
- c) {'b': [2], 'a': [3]}
- d) {'a': 1, 'b': [2, 3, 4]}
- e) {'a': 1, 'b': 2, 3: 4}
- f) None of the above



Thank you