

▣

# Langage Evolué 1 (Python)

Hamza DRID

[h.drid@univ-batna2.dz](mailto:h.drid@univ-batna2.dz)

# What is Python?

- Python is an open source, object-oriented, high-level powerful programming language.
- Developed by Guido van Rossum in the early 1990s. Named after Monty Python
- Python runs on many Unix variants, on the Mac, and on Windows 2000 and later.
- Available for download from <http://www.python.org>.

# Python Program

- Python programs are composed of modules
- Modules contain statements
- Statements contain expressions
- Expressions create and process objects

# Features of Python

- **Open source:** Python is publicly available open source software, any one can use source code that doesn't cost anything.
- **Easy-to-learn:** Popular (scripting/extension) language, clear and easy syntax, no type declarations, automatic memory management, high-level data types and operations, design to read (more English like syntax) and write (shorter code compared to C, C++, and Java) fast.
- **High-level Language:**  
High-level language (closer to human) refers to the higher level of concept from machine language (for example assembly languages). Python is an example of a high-level language like C, C++, Perl, and Java with low-level optimization.
- **Portable:**  
High level languages are portable, which means they are able to run across all major hardware and software platforms with few or no change in source code.
- **Object-Oriented:** Python is a full-featured object-oriented programming language, with features such as classes, inheritance, objects, and overloading.
- **Python is Interactive :**  
Python has an interactive console where you get a Python prompt (command line) and interact with the interpreter directly to write and test your programs. This is useful for mathematical programming.
- **Interpreted :** Python programs are interpreted, takes source code as input, and then compiles (to portable byte-code) each statement and executes it immediately. No need to compiling or linking
- **Extendable :** Python is often referred to as a "glue" language, meaning that it is capable to work in mixed-language environment. The Python interpreter is easily extended and can add a new built-in function or modules written in C/C++/Java code.
- **Libraries :** Databases, web services, networking, numerical packages, graphical user interfaces, 3D graphics, others.
- **Supports :** Support from online Python community

# Python Interpreter

- In interactive mode, type Python programs and the interpreter displays the result:
- Type python into your terminal's command line
- After a short message, the >>> symbol will appear
- The above symbol signals the start of a Python interpreter's command line.
- Python interpreter evaluates inputs (For example >>> 4\*(6-2) return 16)

# How stable is Python?

- Very stable. New, stable releases have been coming out roughly every 6 to 18 months since 1991, and this seems likely to continue. Currently there are usually around 18 months between major releases.
- The latest stable releases can always be found on the [Python download page](#). There are two recommended production-ready versions at this point in time, because at the moment there are two branches of stable releases: 2.x and 3.x.

# Python Installation on Windows

- Download .msi file of Python from <https://www.python.org/downloads/windows>
- Once downloaded, double-click the file to install Python 3.9.10 on your Windows.
- **Start Installation:**

## Stable Releases

- [Python 3.9.10 - Jan. 14, 2022](#)

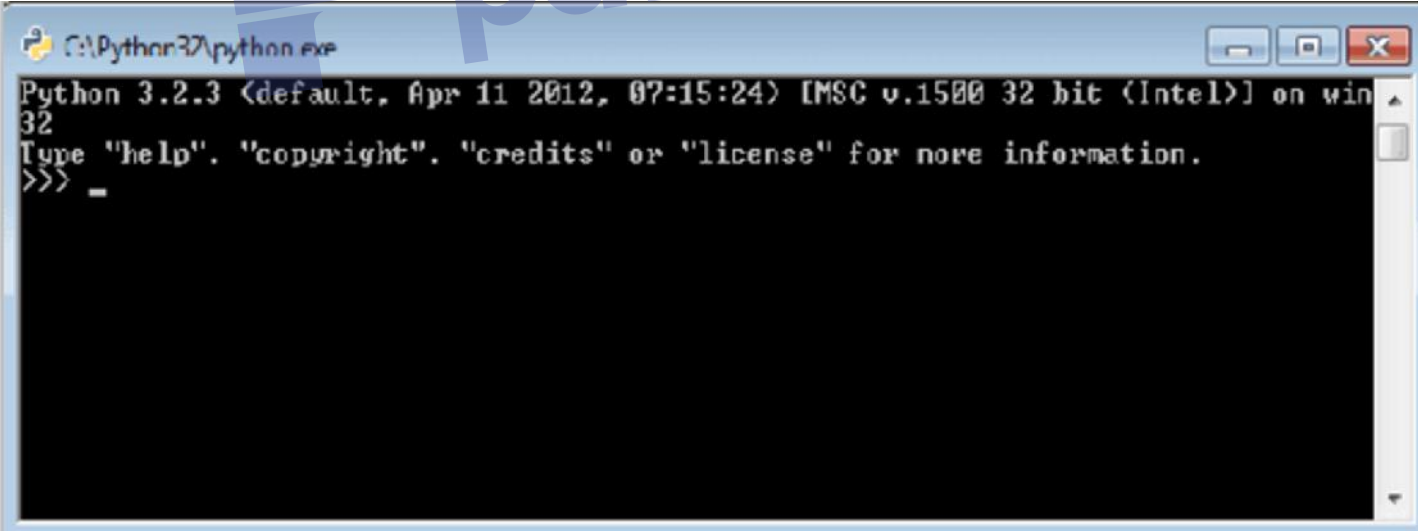
Note that Python 3.9.10 cannot be used on Windows 7 or earlier.

- [Download Windows embeddable package \(32-bit\)](#)
- [Download Windows embeddable package \(64-bit\)](#)
- [Download Windows help file](#)
- [Download Windows installer \(32-bit\)](#)
- [Download Windows installer \(64-bit\)](#)



# Python Installation on Windows

- Click Finish button at this stage and that's it. You may open Python command line from Start Menu and you may run python commands as shown bellow:



```
C:\Python32\python.exe
Python 3.2.3 (default, Apr 11 2012, 07:15:24) [MSC v.1500 32 bit (Intel)] on win
32
Type "help". "copyright". "credits" or "license" for more information.
>>> _
```



# Python IDLE

- IDLE is an integrated development environment (an application like a word processor which helps developers to write programs) for Python.
- IDLE is the Python IDE which comes with Python, built with the tkinter GUI toolkit.
- It has two modes Interactive and Development.

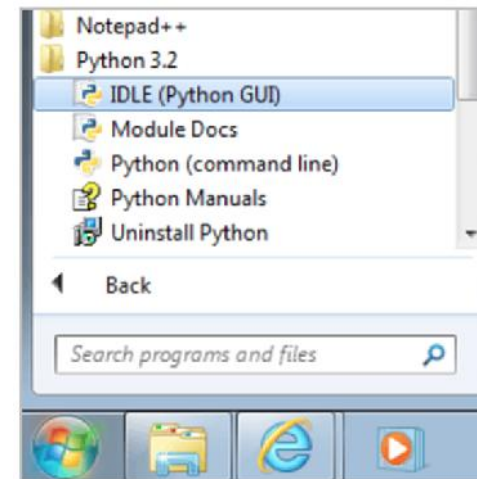
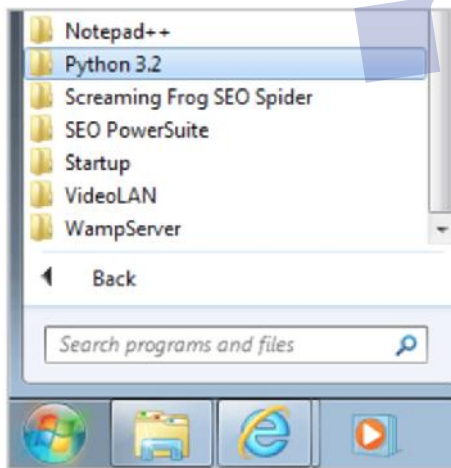
# IDLE features

- Cross Platform : Works on Unix and Windows.
- Multi-window text editor
- Python shell window with syntax highlighting.
- Integrated debugger.
- Coded in Python, using the tkinter GUI toolkit

# Python IDLE: Interactive Mode

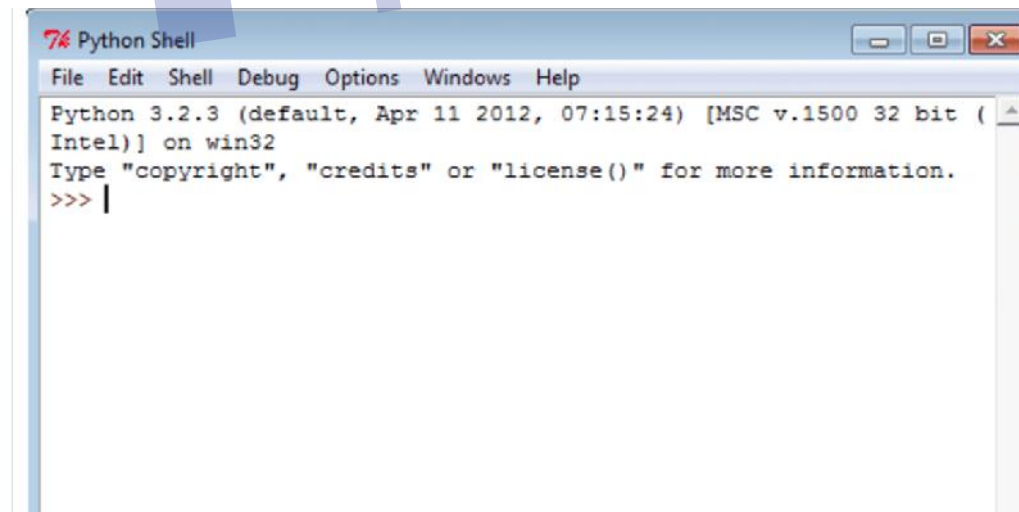
- Let us assume that we've already installed Python.
- Click on start button and find Python 3.2 tab in installed programs

(<C:\Users\Hamza\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Python 3.9>)



# Python IDLE: Interactive Mode

- To start IDLE click on IDLE (Python GUI) icon, you will see a new window opens up and the cursor is waiting beside
- '>>>' sign which is called command prompt.



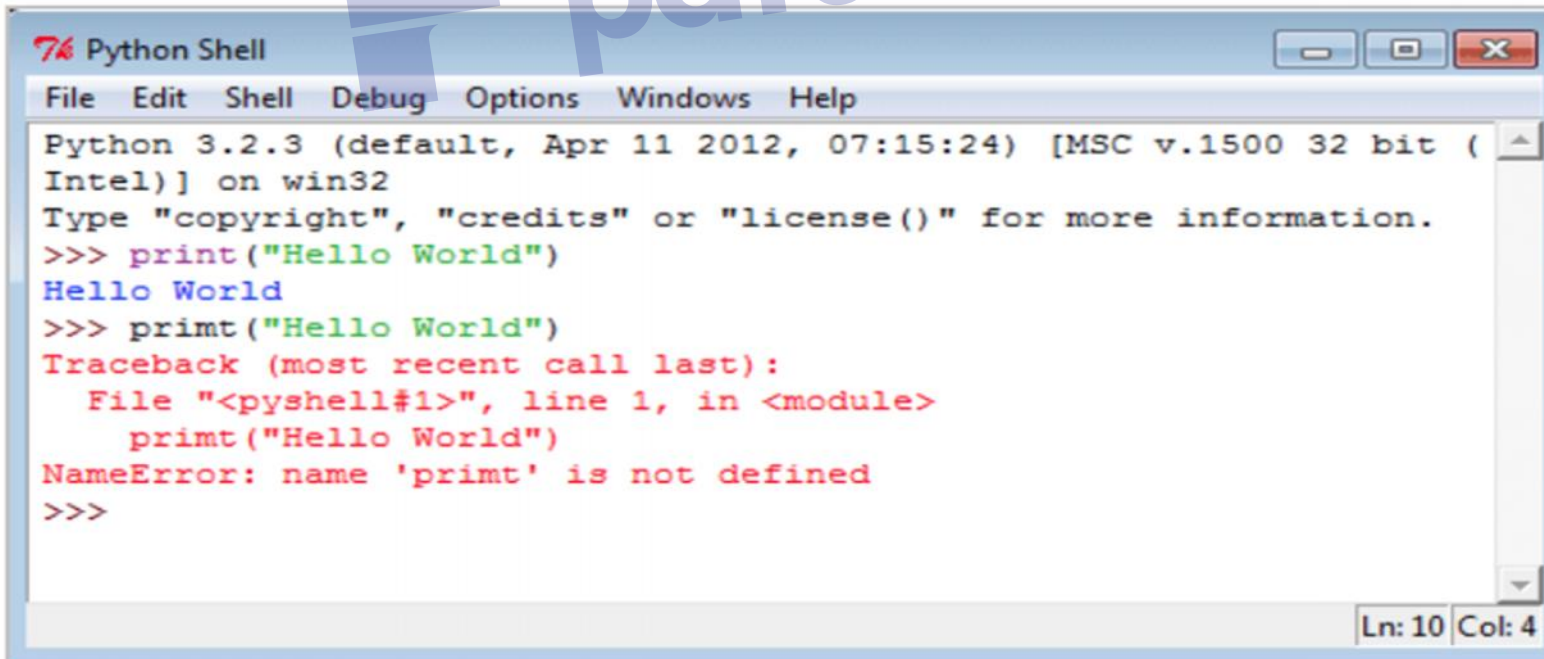
```
Python Shell
File Edit Shell Debug Options Windows Help
Python 3.2.3 (default, Apr 11 2012, 07:15:24) [MSC v.1500 32 bit (
Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> |
```

# Python IDLE: Interactive Mode

- This mode is called interactive mode as you can interact with IDLE directly, you type something (single unit in a programming language) and press enter key Python will execute it
  - but you can not execute your entire program here.
- At the command prompt type copyright and press enter key Python executes the copyright information.

# Python IDLE: Interactive Mode

- Now Python is ready to read new command. Let's execute the following commands one by one.
  - Command -1 : `print("Hello World")`
  - Command -2 : `printt("Hello World")`
- The first command is correct and but the second one has a syntax error, here is the response from Python.



```
Python Shell
File Edit Shell Debug Options Windows Help
Python 3.2.3 (default, Apr 11 2012, 07:15:24) [MSC v.1500 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> print("Hello World")
Hello World
>>> printt("Hello World")
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    printt("Hello World")
NameError: name 'printt' is not defined
>>>
```

Ln: 10 Col: 4

# Python IDLE

- **File Menu**
- New window: Create a new editing window. Shortcut key : Ctrl+N
- Open : Open an existing file. Shortcut key : Ctrl+O
- Recent Files : List of recently open files.
- Open module : Open an existing module. Shortcut key : Alt+M
- Class browser : Show classes and methods in the current file. shortcut key : Alt+C
- Path browser : Show sys.path directories, modules, classes, and methods.
- Save : Save current window to the associated file. Shortcut key : Ctrl+S
- Save As : Save current window to new file, which becomes the associated file. Shortcut key : Ctrl+shift+S
- Save Copy As : Save current window to a different file without changing the associated file. Shortcut key : Alt+shift+S
- Print window. Shortcut key : Ctrl+P
- Close : Close current window (asks to save if unsaved). Shortcut key : Alt+F4
- Exit : Close all windows and quit IDLE (asks to save if unsaved). Shortcut key : Ctrl+Q

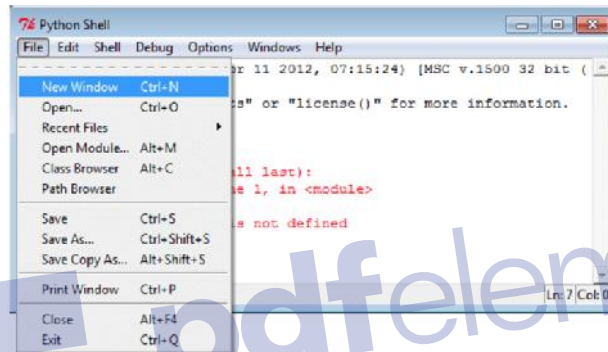
# Python IDLE

- **Edit Menu**
- Undo : Undo last change to current window (max 1000 changes). Shortcut key : Ctrl+Z
- Redo : Redo last undone change to current window. Shortcut key : Ctrl+shift+Z
- Cut : Copy selection into system-wide clipboard; then delete the selection. Shortcut key : Ctrl+X
- Copy : Copy selection into the system-wide clipboard. Shortcut key : Ctrl+C
- Paste : Insert system-wide clipboard into the window. Shortcut key : Ctrl+V
- Select All: All Select the entire contents of the edit buffer. Shortcut key : Ctrl+A
- Find... : Open a search dialog box with many options. Shortcut key : Ctrl+F
- Find again : Repeat last search. Shortcut key : Ctrl+G
- Find selection : Search for the string in the selection. Shortcut key : Ctrl+F3
- Find in Files... : Open a search dialog box for searching files. Shortcut key : Alt+F3
- Replace...: Open a search-and-replace dialog box. Shortcut key : Ctrl+H
- Go to line : Ask for a line number and show that line. Shortcut key : Alt+G
- Expand word : Expand the word you have typed to match another word in the same buffer; repeat to get a different expansion
- Show call tip. Shortcut key : Ctrl+backslash
- Show surrounding parentheses. Shortcut key : Ctrl+0
- Show completions. Shortcut key : Ctrl+space

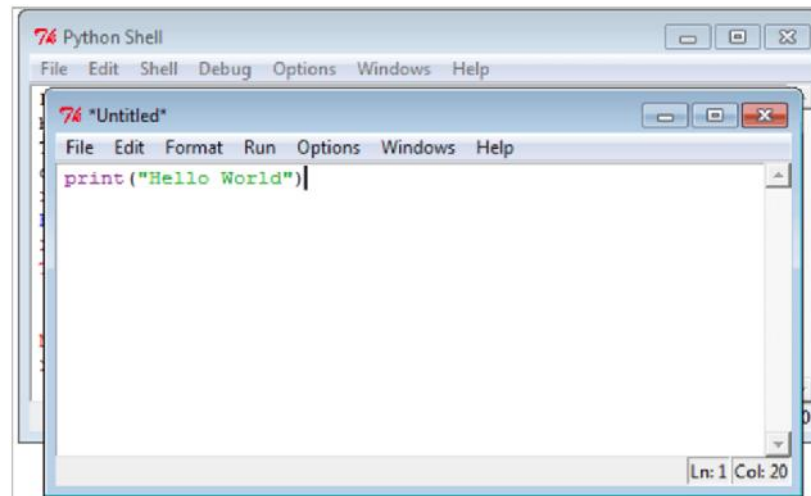


# Python IDLE : Development mode

- At first, start with a new window.

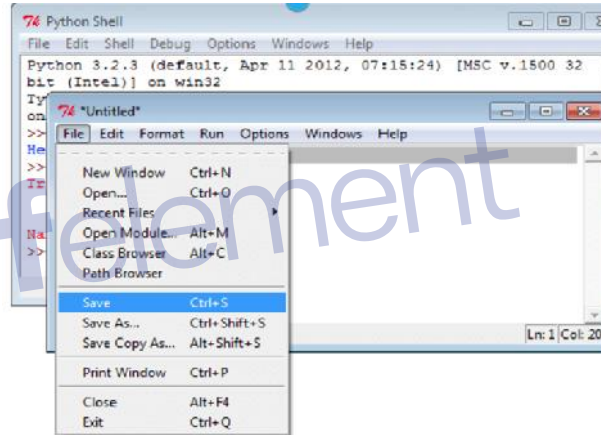


- Clicking on "New window" under file menu a new window will come. Type print "Hello World" in the new window.

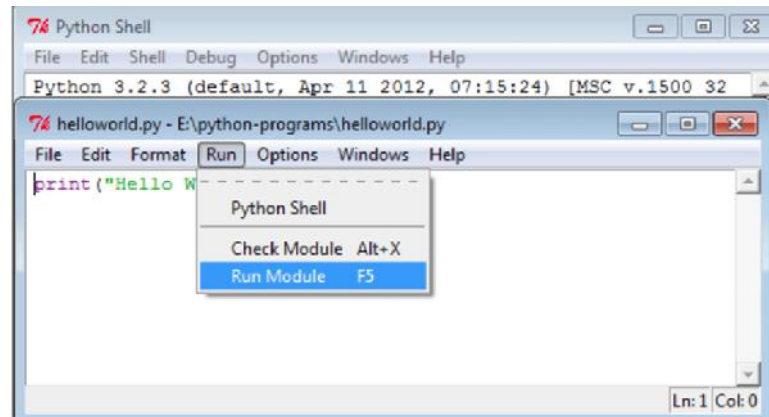


# Python IDLE : Development mode

- Let's save (Save command is located under the File menu) the file now. We save the program as helloworld.py under E:/python-programs folder.



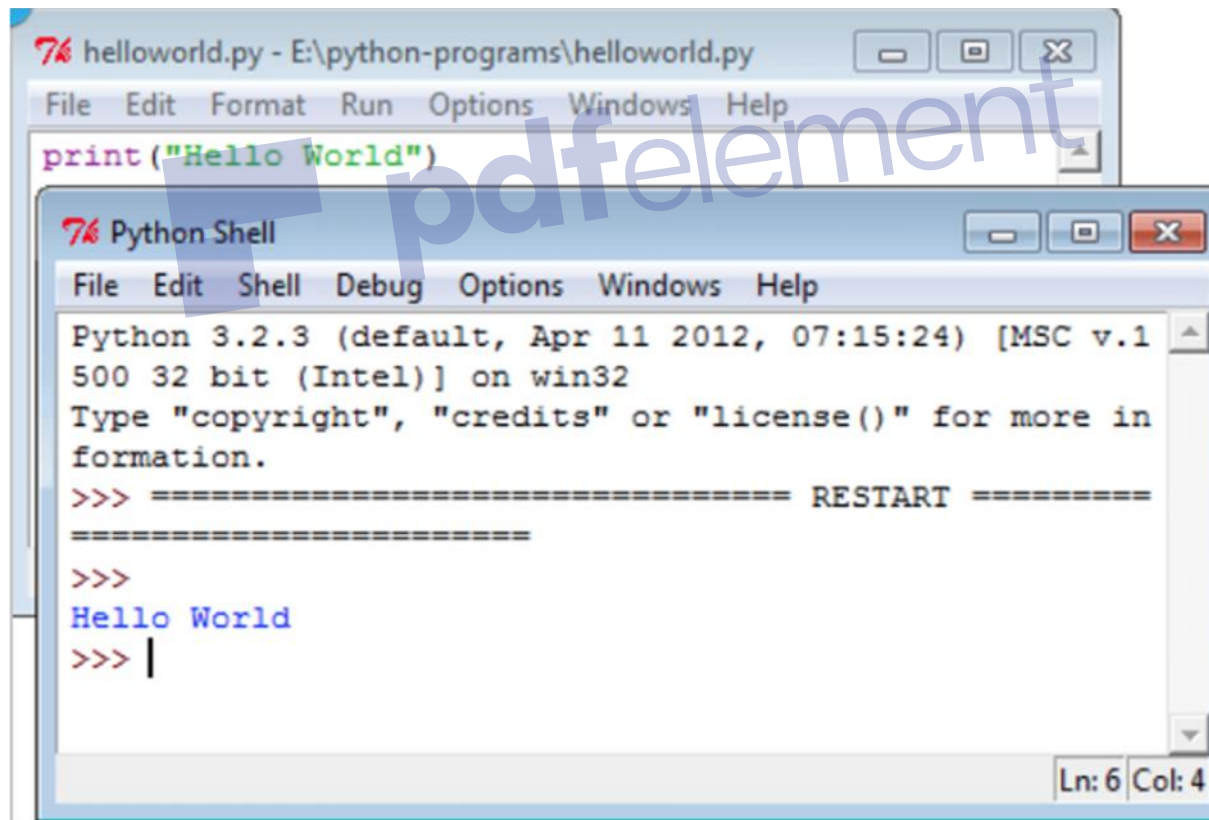
- To run the program select Run menu.



- ,

# Python IDLE : Development mode

- Now click on Run Module or press F5 as a shortcut key to see the result.



The screenshot shows two windows from the Python IDLE application. The top window is titled 'helloworld.py - E:\python-programs\helloworld.py' and contains the code `print("Hello World")`. The bottom window is titled 'Python Shell' and shows the output of the code execution. The shell displays the Python version (3.2.3), system information, and the output 'Hello World'. The shell prompt is currently at the start of a new line.

```
Python 3.2.3 (default, Apr 11 2012, 07:15:24) [MSC v.1500 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART =====
>>>
Hello World
>>> |
```

Ln: 6 Col: 4

# Python Syntax

- A Python program is read by a parser.
- Python was designed to be a highly readable language.
- The syntax of the Python programming language is the set of rules which defines how a Python program will be written.

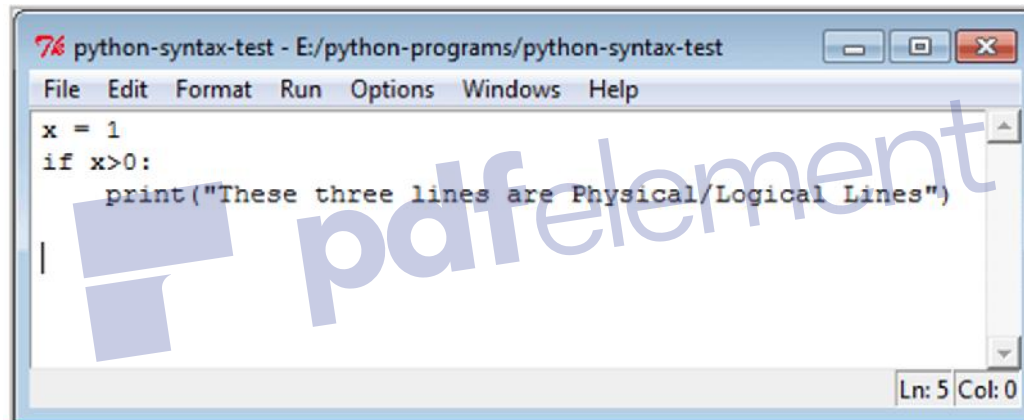
# Python Syntax

- **Python Line Structure:**

- A Python program is divided into a number of logical lines and every logical line is terminated by the token NEWLINE.
- A logical line is created from one or more physical lines.
- A line contains only spaces, tabs, form feeds possibly a comment, is known as a blank line, and Python interpreter ignores it.
- A physical line is a sequence of characters terminated by an end-of-line sequence (in windows it is called CR LF or return followed by a linefeed and in Unix, it is called LF or linefeed).

# Python Syntax

- See the following example.

A screenshot of a Python IDE window titled "python-syntax-test - E:/python-programs/python-syntax-test". The window has a menu bar with "File", "Edit", "Format", "Run", "Options", "Windows", and "Help". The code editor contains the following Python code:

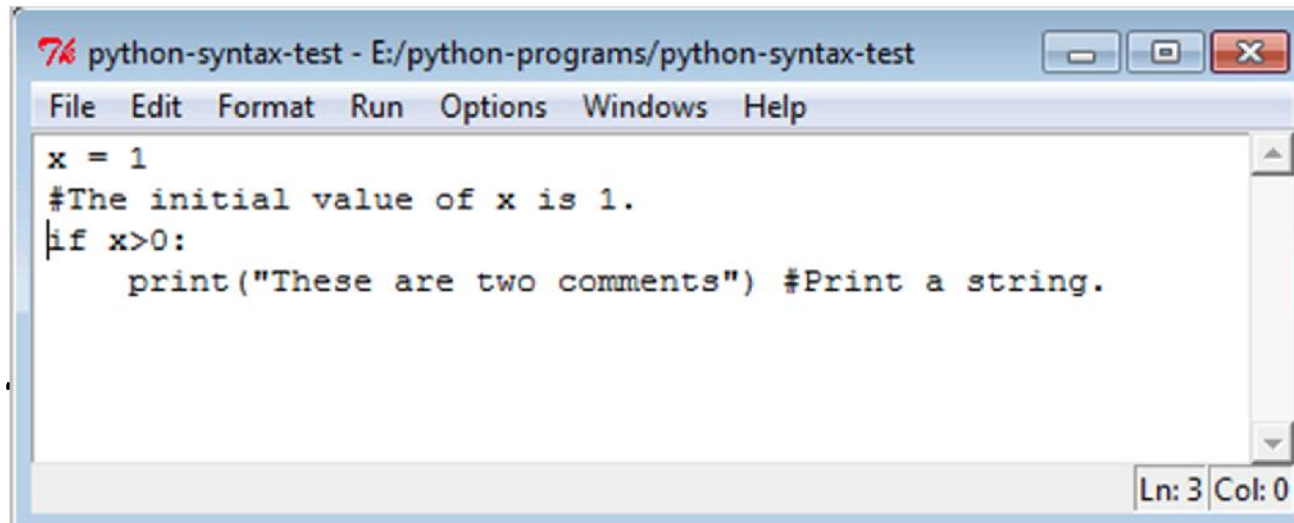
```
x = 1
if x>0:
    print("These three lines are Physical/Logical Lines")
|
```

The code is indented. A large, semi-transparent watermark "pdfelement" is overlaid on the code. The status bar at the bottom right shows "Ln: 5 Col: 0".

```
python-syntax-test - E:/python-programs/python-syntax-test
File Edit Format Run Options Windows Help
x = 1
if x>0:
    print("These three lines are Physical/Logical Lines")
|
Ln: 5 Col: 0
```

# Python Syntax

- **Comments in Python:**
  - A comment begins with a hash character(#) which is not a part of the string literal and ends at the end of the physical line.
  - All characters after the # character up to the end of the line are part of the comment and the Python interpreter ignores them.



```
python-syntax-test - E:/python-programs/python-syntax-test
File Edit Format Run Options Windows Help
x = 1
#The initial value of x is 1.
if x>0:
    print("These are two comments") #Print a string.
Ln: 3 Col: 0
```

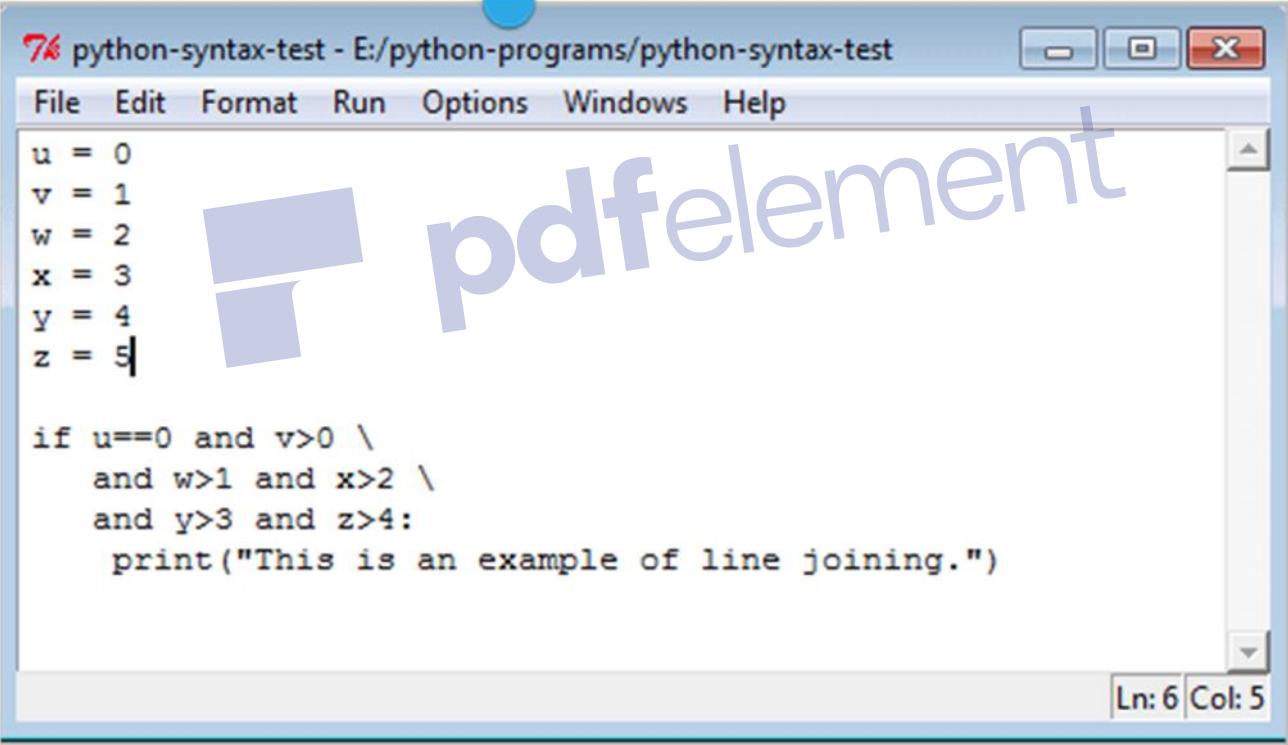
# Python Syntax

- **Joining two lines:**
  - When you want to write a long code in a single line you can break the logical line in two or more physical lines using backslash character(\).
  - Therefore when a physical line ends with a backslash characters(\) and not a part of a string literal or comment then it can join another physical line.
  - See the following example.



# Python Syntax

- Joining two lines:



The screenshot shows a Python IDE window titled "python-syntax-test - E:/python-programs/python-syntax-test". The window contains the following Python code:

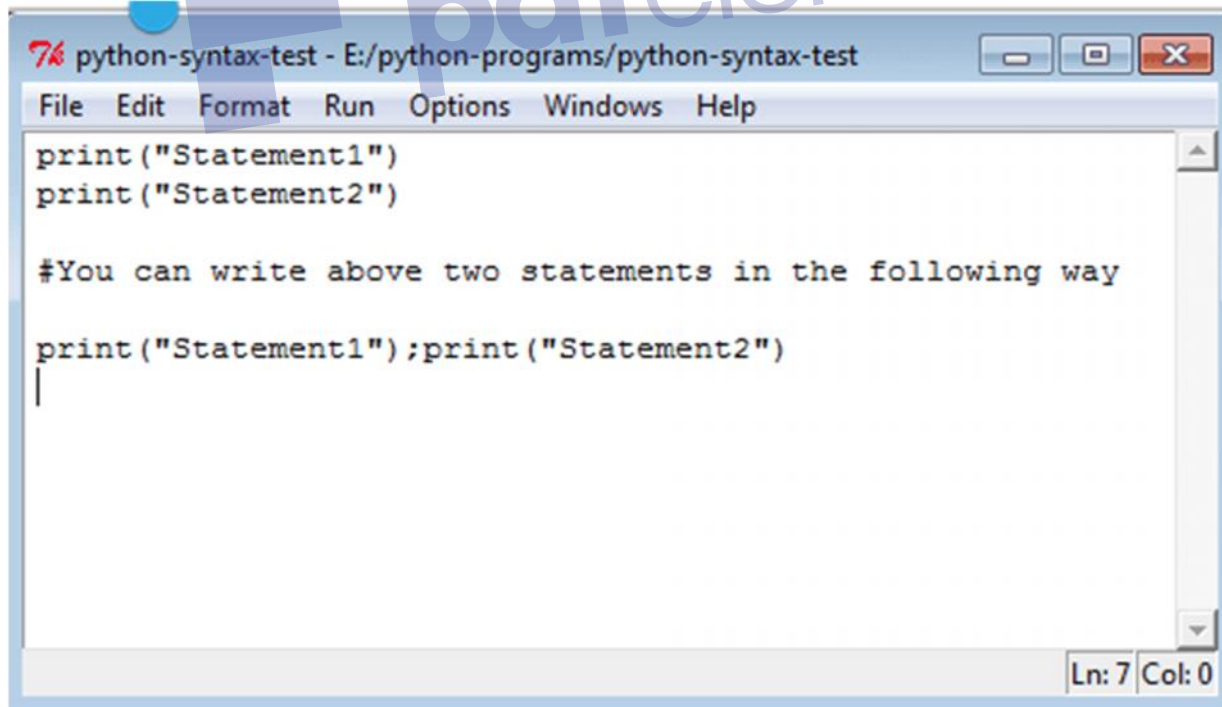
```
u = 0
v = 1
w = 2
x = 3
y = 4
z = 5

if u==0 and v>0 \
    and w>1 and x>2 \
    and y>3 and z>4:
    print("This is an example of line joining.")
```

The status bar at the bottom right of the window indicates "Ln: 6 Col: 5". A large, semi-transparent watermark "pdfelement" is overlaid on the code.

# Python Syntax

- **Multiple Statements on a Single Line:**
  - You can write two separate statements into a single line using a semicolon (;) character between two line.



```
python-syntax-test - E:/python-programs/python-syntax-test
File Edit Format Run Options Windows Help
print("Statement1")
print("Statement2")

#You can write above two statements in the following way
print("Statement1");print("Statement2")
|
```

Ln: 7 Col: 0

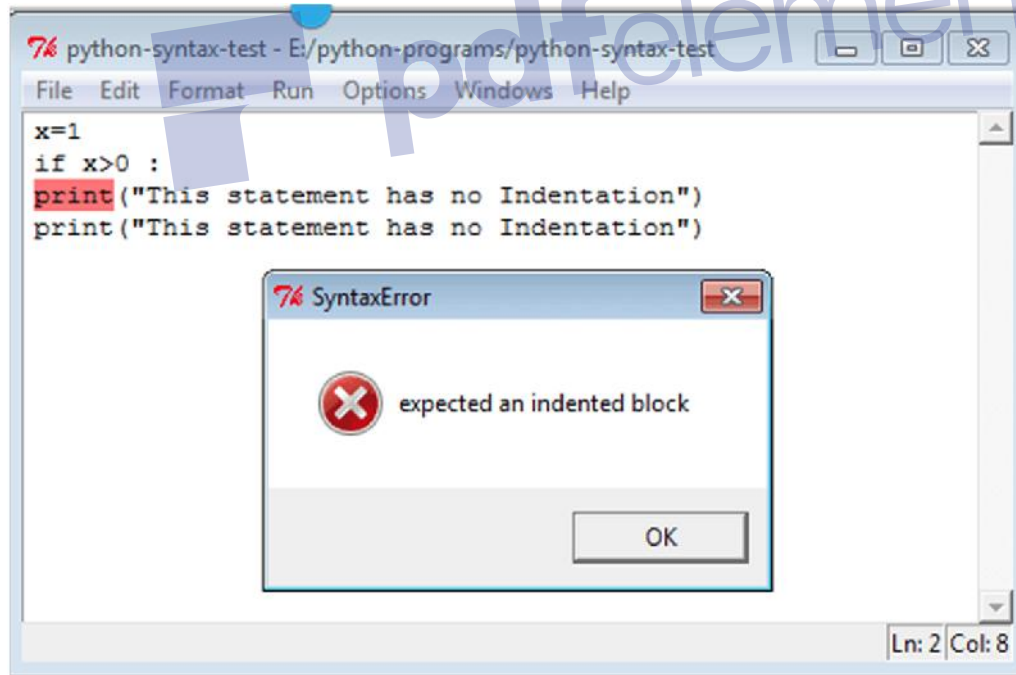
# Python Syntax

- **Indentation:**

- Python uses whitespace (spaces and tabs) to define program blocks whereas other languages like C, C++ use braces ({}).  
– The number of whitespaces (spaces and tabs) in the indentation is not fixed, but all statements within the block must be the indented same amount.

# Python Syntax

- **Indentation:**
  - In the following program, the block statements have no indentation.



The screenshot shows a Python IDE window titled "python-syntax-test - E:/python-programs/python-syntax-test". The code in the editor is:

```
x=1
if x>0 :
print("This statement has no Indentation")
print("This statement has no Indentation")
```

The word "print" on the first line of the if block is highlighted in red. A dialog box titled "SyntaxError" is open in the foreground, displaying a red 'X' icon and the message "expected an indented block". The dialog box has an "OK" button. The status bar at the bottom right of the IDE window shows "Ln: 2 Col: 8".

# Python Syntax

- **Python Coding Style:**

- Use 4 spaces per indentation and no tabs.
- Do not mix tabs and spaces. Tabs create confusion and it is recommended to use only spaces.
- Maximum line length : 79 characters which help users with a small display.
- Use blank lines to separate top-level function and class definitions and single blank line to separate methods definitions inside a class and larger blocks of code inside functions.
- When possible, put inline comments (should be complete sentences).
- Use spaces around expressions and statements.

# Python Syntax

- **Python Reserve words:**
  - The following identifiers are used as reserved words of the language, and cannot be used as ordinary identifiers.

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

# Python print() function

- The print statement can be used in the following ways :
  - `print("Good Morning")`
  - `print("Good", <Variable Containing the String>)`
  - `print("Good" + <Variable Containing the String>)`
  - `print("Good %s" % <variable containing the string>)`
- In Python, single, double and triple quotes are used to denote a string.
  - Most use single quotes when declaring a single character.
  - Double quotes when declaring a line and triple quotes when declaring a paragraph/multiple lines.

# Python print() function

## Double Quotes Use:

### Example:

```
1 print("Python is very simple language")
```

Output:

```
Python is very simple language
```

## Single Quotes Use:

### Example:

```
1 print('Hello')
```

Output:

```
Hello
```

## Triple Quotes Use:

### Example:

```
1 print("""Python is very popular language.  
2 It is also friendly language.""")
```

Output:

```
Python is very Popular Language.  
It is also friendly language.
```



# Python print() function

- **Variable Use:**

- Strings can be assigned to variable say string1 and string2 which can called when using the print statement.

Example:

```
1 str1 = 'wel'
2 print(str1, 'come')
```

Output:

```
wel come
```

Example:

```
1 str1 = 'welcome'
2 str2 = 'Python'
3 print(str1, str2)
```

Output:

```
welcome Python
```

# Python print() function

- **String Concatenation:**
  - String concatenation is the "addition" of two strings. Observe that while concatenating there will be no space between the strings.

Example:

```
1 str1 = 'Python'  
2 str2 = ':'  
3 print('Welcome' + str1 + str2)
```

Output:

```
WelcomePython:
```

# Python print() function

- **Using as String:**
  - %s is used to refer to a variable which contains a string.

Example:

```
1 str1 = 'Python'
2 print("Welcome %s" % str1)
```

Output:

```
Welcome Python
```

# Python print() function

- **Using other data types:**
  - Similarly, when using other data types
  - %d -> Integer
  - %e -> exponential
  - %f -> Float
  - %o -> Octal
  - %x -> Hexadecimal
  - This can be used for conversions inside the print statement itself.

# Python print() function

## Using as Integer:

### Example:

```
1 print("Actual Number = %d" %15)
```

### Output:

```
Actual Number = 15
```

## Using as Exponential:

### Example:

```
1 print("Exponential equivalent of the number = %e" %15)
```

### Output:

```
Exponential equivalent of the number = 1.500000e+01
```

## Using as Float:

### Example:

```
1 print("Float of the number = %f" %15)
```

### Output:

```
Float of the number = 15.000000
```

# Python print() function

- **Using multiple variables:**
  - When referring to multiple variables parenthesis is used.

Example:

```
1 str1 = 'World'
2 str2 = ':'
3 print("Python %s %s" %(str1, str2))
```

Output:

```
Python World :
```

# Python print() function

- **Other Examples of Print Statement:**

## Example-2:

\n is used for Line Break.

```
1 | print("Sunday\nMonday\nTuesday\nWednesday\nThursday\nFriday\nSaturday")
```

Output:

```
Sunday
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday
```

## Example-3:

Any word print multiple times.

```
1 | print('-w3r'*5)
```

Output:

```
-w3r-w3r-w3r-w3r-w3r
```

# Python print() function

## Example-4:

\t is used for tab.

```
1 print("""
2 Language:
3 \t1 Python
4 \t2 Java\n\t3 JavaScript
5 """)
```

Output:

```
Language:
  1 Python
  2 Java
  3 JavaScript
```



# Exercises

- **Exercise 1 : Simple Messages**
  - Assign a message to a variable, and print that message. Then change the value of your variable to a new message, and print the new message.
- **Exercise 2 : Favorite Number**
- Use a variable to represent your favorite number. Then, using that variable, create a message that reveals your favorite number. Print that message.

# Exercices

- `msg = "I love learning to use Python."`
  - `print(msg)`
  - `msg = "It's really satisfying!"`
  - `print(msg)`
- Output:
  - I love learning to use Python.
  - It's really satisfying!
- **\*\*\*\*\***
  - `fav_num = 42`
  - `msg = "My favorite number is {fav_num}."`
  - `print(msg)`
- Output:
  - My favorite number is 42.

# Python Variable

- **Variable and Value**

- A variable is a memory location where a programmer can store a value. Example : roll\_no, amount, name etc.
- Value is either string, numeric etc.
  - Example : "Sara", 120, 25.36
- Variables are created when first assigned.
- Variables must be assigned before being referenced.
- The value stored in a variable can be accessed or updated later.
- No declaration required
  - The type (string, int, float etc.) of the variable is determined by Python
- The interpreter allocates memory on the basis of the data type of a variable.

# Python Variable

- **Python Variable Name Rules**
  - Must begin with a letter (a - z, A - B) or underscore (`_`)
  - Other characters can be letters, numbers or `_`
  - Case Sensitive
  - Can be any (reasonable) length
  - There are some reserved words which you cannot use as a variable name because Python uses them for other things.

# Python Variable

- **Good Variable Name**

- Choose meaningful name instead of short name.  
roll\_no is better than rn.
- Maintain the length of a variable name.  
Roll\_no\_of\_a-student is too long?
- Be consistent; roll\_no or RollNo
- Begin a variable name with an underscore(\_) character for a special case.

# Python Variable

- **Python Assignment Statements**

- The assignment statement creates new variables and gives them values. Basic assignment statement in Python is :

Syntax:

```
<variable> = <expr>
```

- Where the equal sign (=) is used to assign value (right side) to a variable name (left side).

# Python Variable

- See the following statements :

```
1 >>> Item_name = "Computer" #A String
2 >>> Item_qty = 10 #An Integer
3 >>> Item_value = 1000.23 #A floating point
4 >>> print(Item_name)
5 Computer
6 >>> print(Item_qty)
7 10
8 >>> print(Item_value)
9 1000.23
10 >>>
```

- One thing is important, assignment statement read right to left only.

```
1 >>> a = 12
2 >>> 12 = a
3 SyntaxError: can't assign to literal
4 >>>
```

# Python Variable

- **Multiple Assignment**

- The basic assignment statement works for a single variable and a single expression. You can also assign a single value to more than one variables simultaneously.

```
var1=var2=var3...varn= = <expr>
```

```
<
```

- **Example:**

- `x = y = z = 1` Now check the individual value in Python Shell.

```
1 >>> x = y = z = 1
2 >>> print(x)
3 1
4 >>> print(y)
5 1
6 >>> print(z)
7 1
8 >>>
```



# Python Variable

- **Multiple Assignment**

- Here is an another assignment statement where the variables assign many values at the same time.

Syntax:

```
<var>, <var>, ..., <var> = <expr>, <expr>, ..., <expr>
```

- In the above example x, y and z simultaneously get the new values 1, 2 and "abcd".

```
1 >>> x,y,z = 1,2,"abcd"
2 >>> print(x)
3 1
4 >>> print(y)
5 2
6 >>> print(z)
7 abcd
```

# Python Variable

- **Multiple Assignment**

- You can reuse variable names by simply assigning a new value to them :

```
1 >>> x = 100
2 >>> print(x)
3 100
4 >>> x = "Python"
5 >>> print(x)
6 Python
7 >>>
```

- Other ways to define value

```
1 >>> five_millions = 5_000_000
2 >>> five_millions
```

Output:

```
5000000
```

# Python Variable

- **Swap variables**

- Python swap values in a single line and this applies to all objects in python.

Syntax:

```
var1, var2 = var2, var1
```

Example:

|    |                 |
|----|-----------------|
| 1  | >>> x = 10      |
| 2  | >>> y = 20      |
| 3  | >>> print(x)    |
| 4  | 10              |
| 5  | >>> print(y)    |
| 6  | 20              |
| 7  | >>> x, y = y, x |
| 8  | >>> print(x)    |
| 9  | 20              |
| 10 | >>> print(y)    |
| 11 | 10              |
| 12 | >>>             |

# Python Variable

- **Local and global variables in Python**
  - In Python, if a variable is assigned a value anywhere within the function's body, it's assumed to be a local unless explicitly declared as global.

Example:

```
1 var1 = "Python"
2 def func1():
3     var1 = "PHP"
4     print("In side func1() var1 = ",var1)
5
6 def func2():
7     print("In side func2() var1 = ",var1)
8 func1()
9 func2()
```

Output:

```
In side func1() var1 = PHP
In side func2() var1 = Python
```

# Python Variable

- **Local and global variables in Python**
  - You can use a global variable in other functions by declaring it as global keyword :

Example:

```
1 def func1():
2     global var1
3     var1 = "PHP"
4     print("In side func1() var1 = ",var1)
5
6 def func2():
7     print("In side func2() var1 = ",var1)
8 func1()
9 func2()
```

Output:

```
In side func1() var1 = PHP
In side func2() var1 = PHP
```

# Python Variable



# Python Data Type



# Python Data Type

- **Introduction**
- Type represents the kind of value and determines how the value can be used.
  - All data values in Python are encapsulated in relevant object classes.
  - Everything in Python is an object and every object has an identity, a type, and a value.
  - Like another object-oriented language such as Java or C++, there are several data types which are built into Python.
- To determine a variable's type in Python you can use the `type()` function.
- The value of some objects can be changed.
- Objects whose value can be changed are called mutable and objects whose value is unchangeable (once they are created) are called immutable.



# Python Data Type

- **Numbers**

- Numbers are created by numeric literals.
- Numeric objects are immutable, which means when an object is created its value cannot be changed.
- Python has three distinct numeric types: integers, floating point numbers, and complex numbers.
- Integers represent negative and positive integers without fractional parts
- floating point numbers represents negative and positive numbers with fractional parts.
- Booleans are a subtype of plain integers.

# Python Data Type

```
Python Shell
File Edit Shell Debug Options Windows Help
Python 3.2.3 (default, Apr 11 2012, 07:15:24) [MSC v.1500 32 bit
(Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> a=1452
>>> type(a)
<class 'int'>
>>> b=(-4587)
>>> type(b)
<class 'int'>
>>> c=0
>>> type(c)
<class 'int'>
>>> g=1.03
>>> type(g)
<class 'float'>
>>> h=-11.23
>>> type(h)
<class 'float'>
>>> i=.34
>>> type(i)
<class 'float'>
>>> j=2.12e-10
>>> type(j)
<class 'float'>
>>> k=5E220
>>> type(k)
<class 'float'>
>>>
Ln: 27 Col: 4
```

# Python Data Type

## Floats and Integers

```
1 >>> x = 8
2 >>> y = 7
3 >>> x+y
4 >>> x-y
5 >>> x/y
6 >>> x*y
```

Output:

```
15
1
1.1428571428571428
56
```

# Python Data Type

## Exponent

```
1 >>> 4**3
2 >>> 3**4
```

Output:

```
64
81
```

## inter division

```
1 >>> 12/3
2 >>> 64//4
3 >>> 15//3
```

Output:

```
4.0
16
5
```

## Remainder

```
1 >>> 15 % 4
```

Output:

```
3
```

# Python Data Type

## Self-assignment

```
1 >>> count = 0
2 >>> count +=2
3 >>> count
```

Output:

```
2
```

```
1 >>> count -=2
2 >>> count
```

Output:

```
0
```

```
1 >>> count +=2
2 >>> count *=4
3 >>> count
```

Output:

```
8
```

```
1 >>> count **=4
2 >>> count
```

Output:

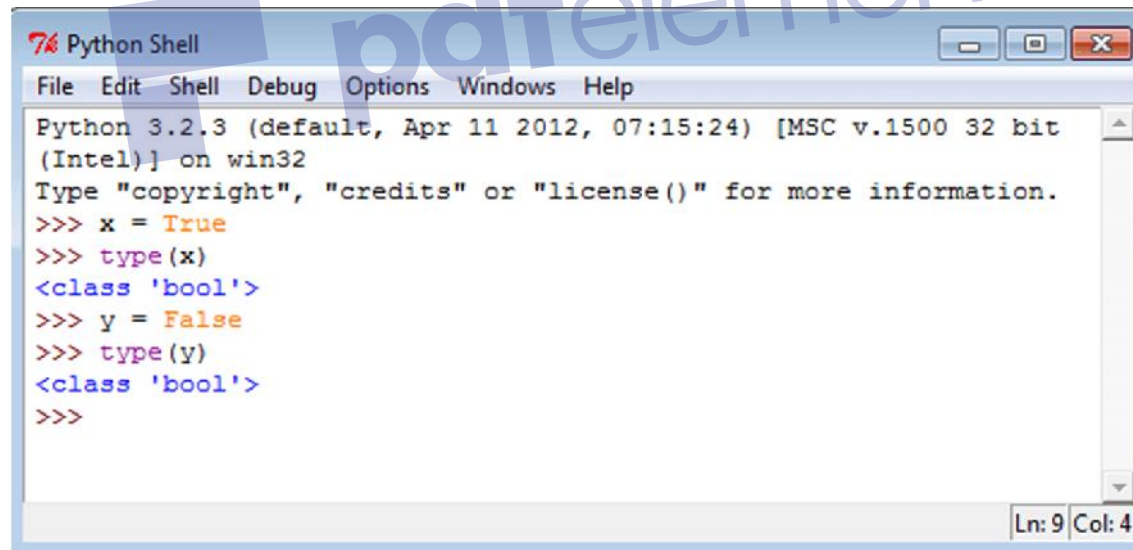
```
4096
```

pdfelement

# Python Data Type

- **Boolean (bool)**

- The simplest build-in type in Python is the bool type, it represents the truth values False and True. See the following statements in Python shell.



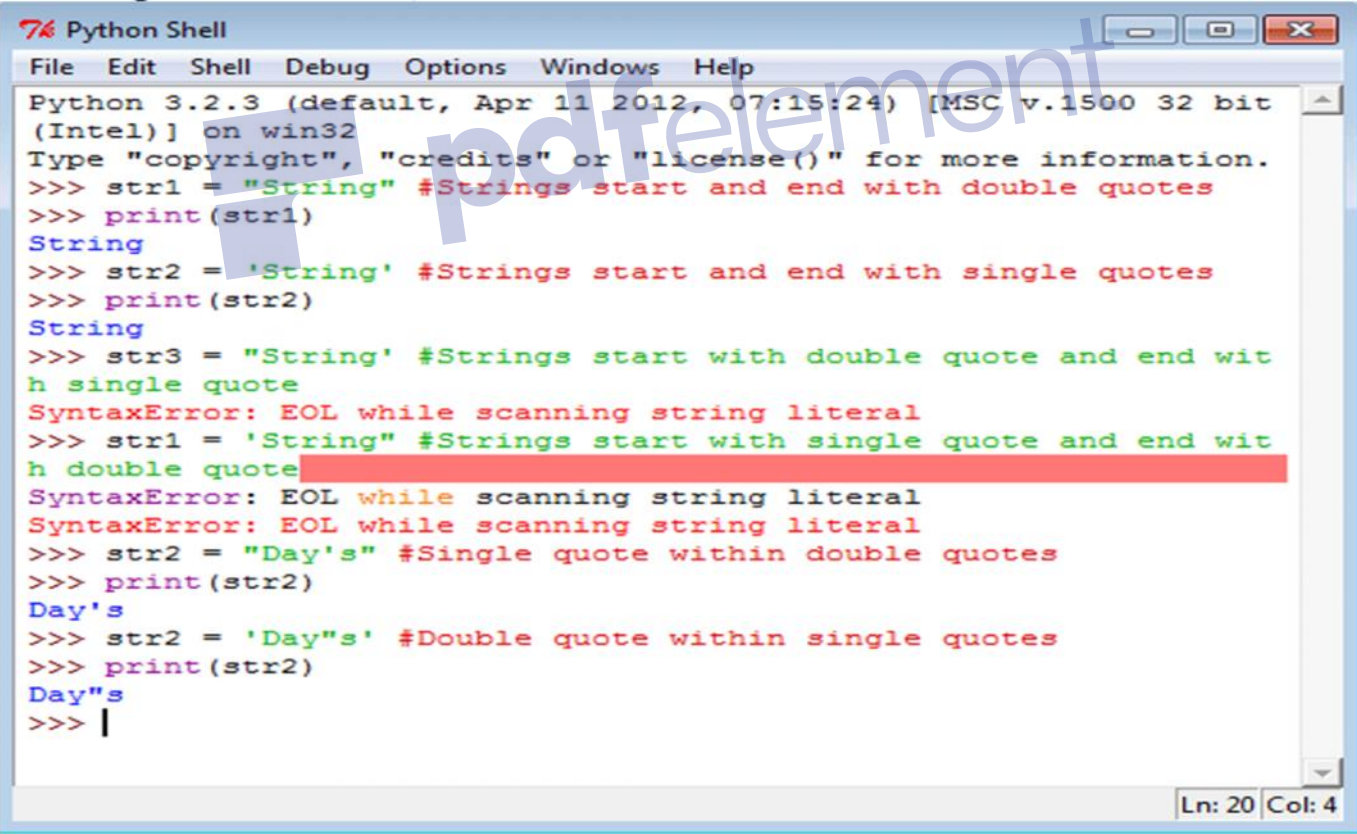
```
Python Shell
File Edit Shell Debug Options Windows Help
Python 3.2.3 (default, Apr 11 2012, 07:15:24) [MSC v.1500 32 bit
(Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> x = True
>>> type(x)
<class 'bool'>
>>> y = False
>>> type(y)
<class 'bool'>
>>>
```

Ln: 9 Col: 4

# Python Data Type

- **Strings**

- In Python, a string type object is a sequence (left-to-right order) of characters.



```
Python Shell
File Edit Shell Debug Options Windows Help
Python 3.2.3 (default, Apr 11 2012, 07:15:24) [MSC v.1500 32 bit
(Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> str1 = "String" #Strings start and end with double quotes
>>> print(str1)
String
>>> str2 = 'String' #Strings start and end with single quotes
>>> print(str2)
String
>>> str3 = "String' #Strings start with double quote and end wit
h single quote
SyntaxError: EOL while scanning string literal
>>> str1 = 'String" #Strings start with single quote and end wit
h double quote
SyntaxError: EOL while scanning string literal
SyntaxError: EOL while scanning string literal
>>> str2 = "Day's" #Single quote within double quotes
>>> print(str2)
Day's
>>> str2 = 'Day"s' #Double quote within single quotes
>>> print(str2)
Day"s
>>> |
```

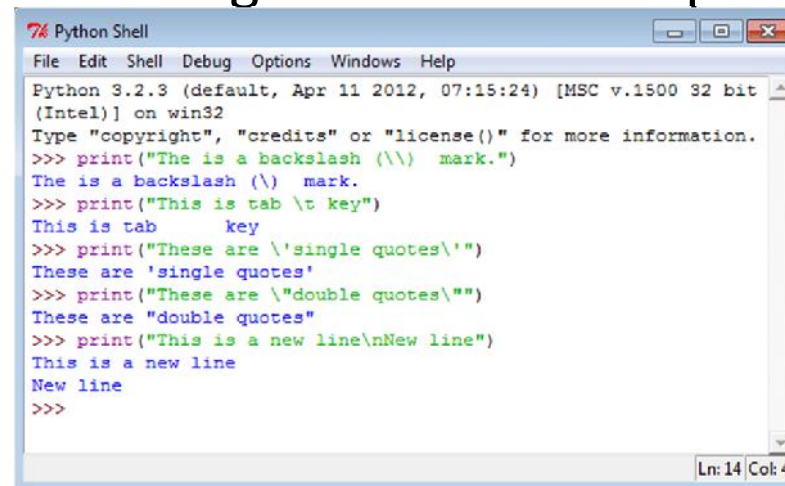
Ln: 20 Col: 4

# Python Data Type

- **Special characters in strings**
  - The backslash (\) character is used to introduce a special character. See the following table.

| Escape sequence | Meaning        |
|-----------------|----------------|
| \n              | Newline        |
| \t              | Horizontal Tab |
| \\              | Backslash      |
| \'              | Single Quote   |
| \"              | Double Quote   |

- See the following statements on special characters.

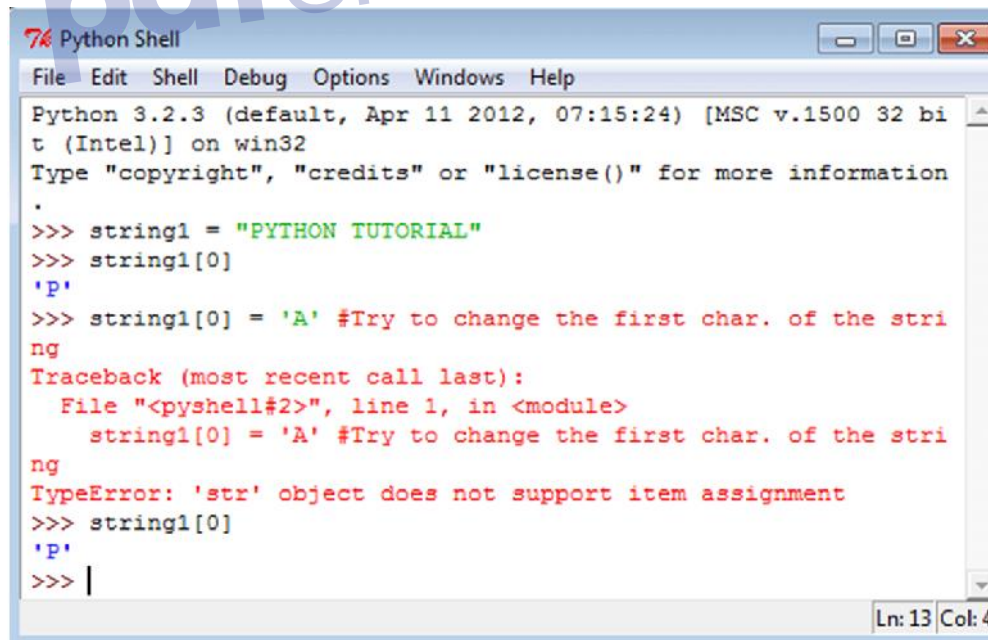


```
Python Shell
Python 3.2.3 (default, Apr 11 2012, 07:15:24) [MSC v.1500 32 bit
(Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> print("The is a backslash (\\) mark.")
The is a backslash (\) mark.
>>> print("This is tab \t key")
This is tab      key
>>> print("These are \'single quotes\'")
These are 'single quotes'
>>> print("These are \"double quotes\"")
These are "double quotes"
>>> print("This is a new line\nNew line")
This is a new line
New line
>>>
```



# Python Data Type

- **Strings are immutable**
  - Strings are immutable character sets. Once a string is generated, you can not change any character within the string. See the following statements.

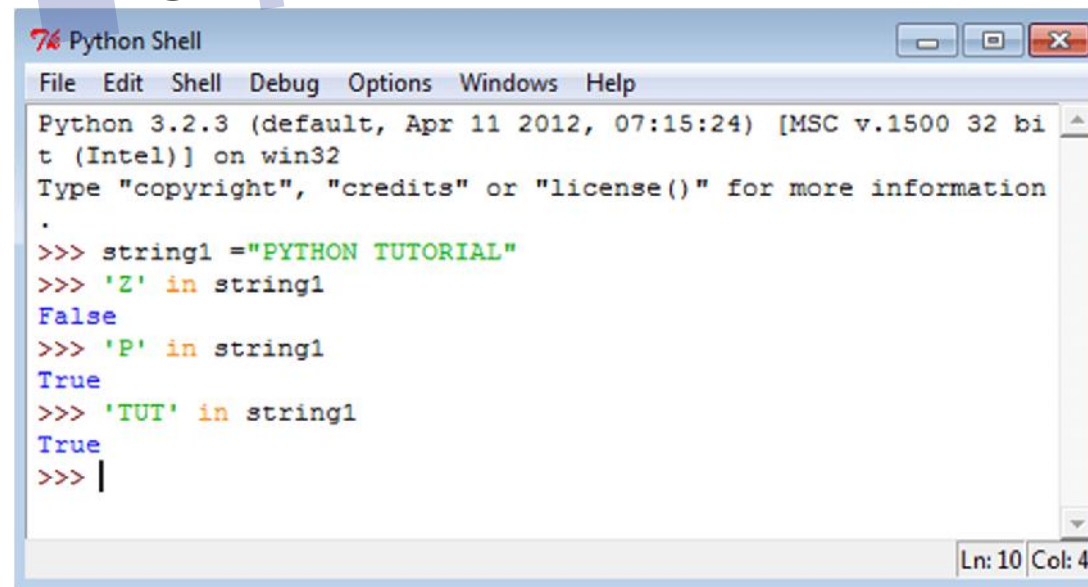


```
Python Shell
File Edit Shell Debug Options Windows Help
Python 3.2.3 (default, Apr 11 2012, 07:15:24) [MSC v.1500 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information
.
>>> string1 = "PYTHON TUTORIAL"
>>> string1[0]
'P'
>>> string1[0] = 'A' #Try to change the first char. of the string
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    string1[0] = 'A' #Try to change the first char. of the string
TypeError: 'str' object does not support item assignment
>>> string1[0]
'P'
>>> |
```

Ln: 13 Col: 4

# Python Data Type

- **'in' operator in Strings**
  - The 'in' operator is used to check whether a character or a substring is present in a string or not. The expression returns a Boolean value. See the following statements.



```
Python Shell
File Edit Shell Debug Options Windows Help
Python 3.2.3 (default, Apr 11 2012, 07:15:24) [MSC v.1500 32 bit
(Intel)] on win32
Type "copyright", "credits" or "license()" for more information
.
>>> string1 = "PYTHON TUTORIAL"
>>> 'Z' in string1
False
>>> 'P' in string1
True
>>> 'TUT' in string1
True
>>> |
```

Ln: 10 Col: 4

# Python Data Type

- Conversion

—

```
1 >>> float("4.5")
2 >>> int("25")
3 >>> int(5.625)
4 >>> float(6)
5 >>> int(True)
6 >>> float(False)
7 >>> str(True)
8 >>> bool(0)
9 >>> bool('Hello world')
10 >>> bool(223.5)
```

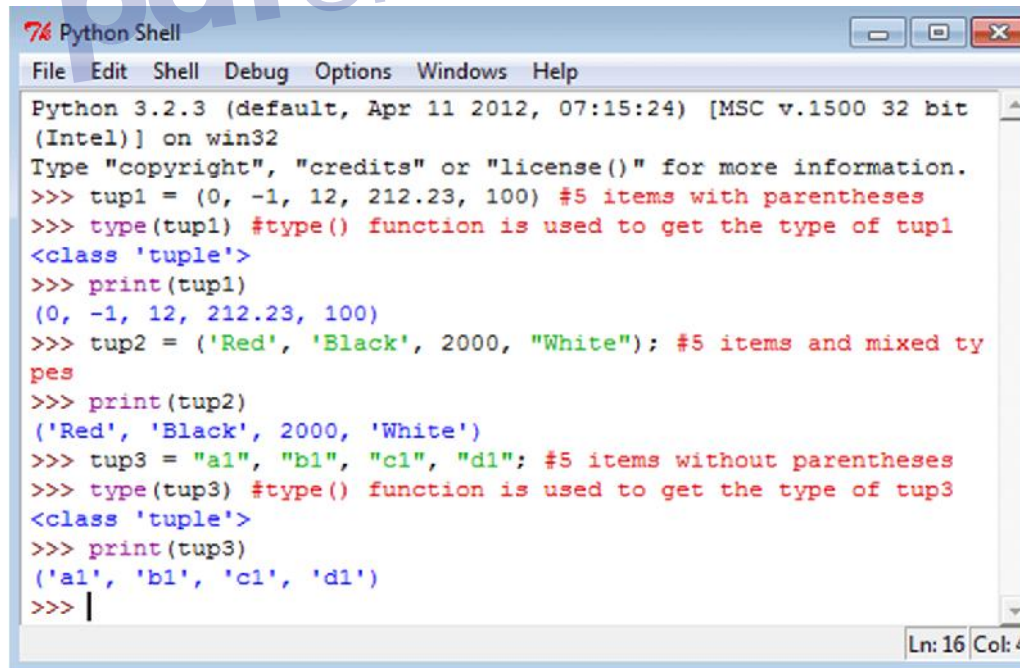
Output:

```
4.5
25
5
6.0
1
0.0
'True'
False
True
True
```

# Python Data Type

- **Tuples**

- A tuple is a container which holds a series of comma-separated values (items or elements) between parentheses.
- Tuples are immutable (i.e. you cannot change its content once created) and can hold mix data types.
- **Creating Tuples**

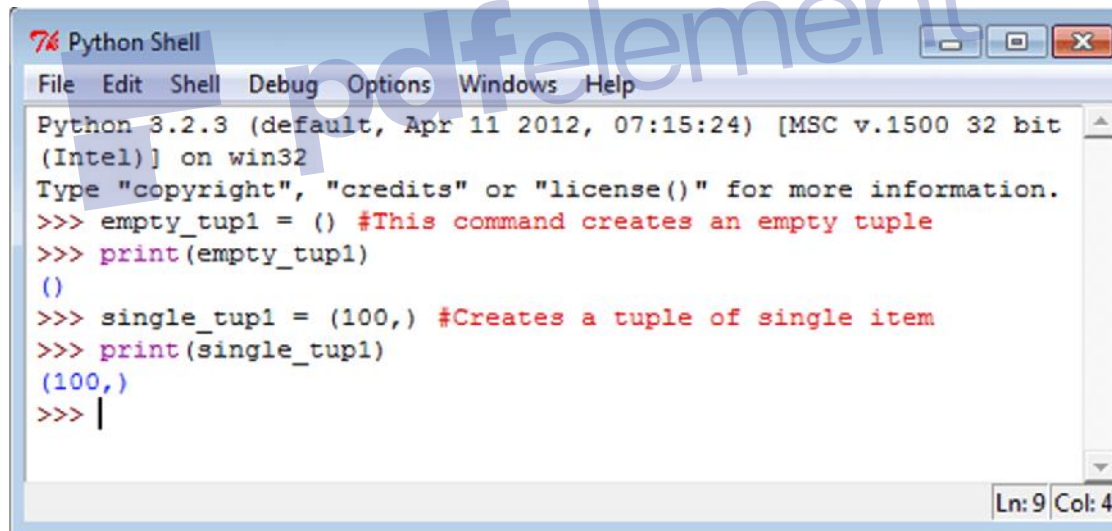


```
Python Shell
File Edit Shell Debug Options Windows Help
Python 3.2.3 (default, Apr 11 2012, 07:15:24) [MSC v.1500 32 bit
(Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> tup1 = (0, -1, 12, 212.23, 100) #5 items with parentheses
>>> type(tup1) #type() function is used to get the type of tup1
<class 'tuple'>
>>> print(tup1)
(0, -1, 12, 212.23, 100)
>>> tup2 = ('Red', 'Black', 2000, "White"); #5 items and mixed ty
pes
>>> print(tup2)
('Red', 'Black', 2000, 'White')
>>> tup3 = "a1", "b1", "c1", "d1"; #5 items without parentheses
>>> type(tup3) #type() function is used to get the type of tup3
<class 'tuple'>
>>> print(tup3)
('a1', 'b1', 'c1', 'd1')
>>> |
```

Ln: 16 Col: 4

# Python Data Type

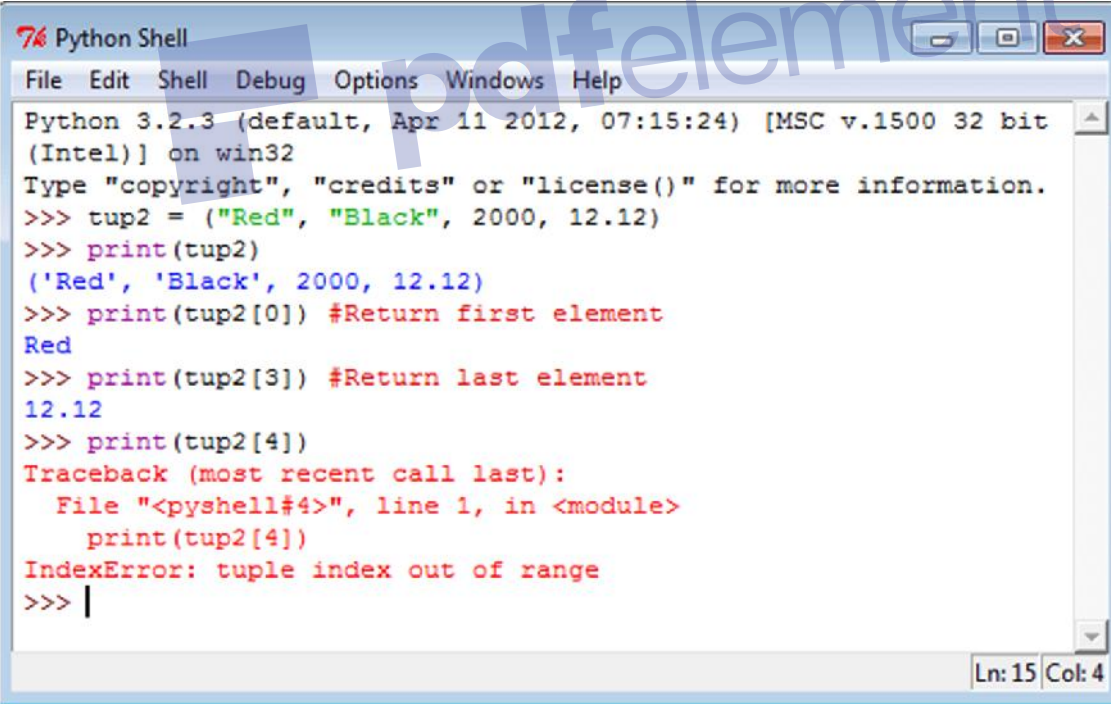
- **Creating Tuples**
  - To create an empty tuple or create a tuple with single element use the following commands.



```
Python Shell
File Edit Shell Debug Options Windows Help
Python 3.2.3 (default, Apr 11 2012, 07:15:24) [MSC v.1500 32 bit
(Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> empty_tup1 = () #This command creates an empty tuple
>>> print(empty_tup1)
()
>>> single_tup1 = (100,) #Creates a tuple of single item
>>> print(single_tup1)
(100,)
>>> |
Ln: 9 Col: 4
```

# Python Data Type

- Elements of a tuple are indexed like other sequences. The tuple indices start at 0. See the following statements.

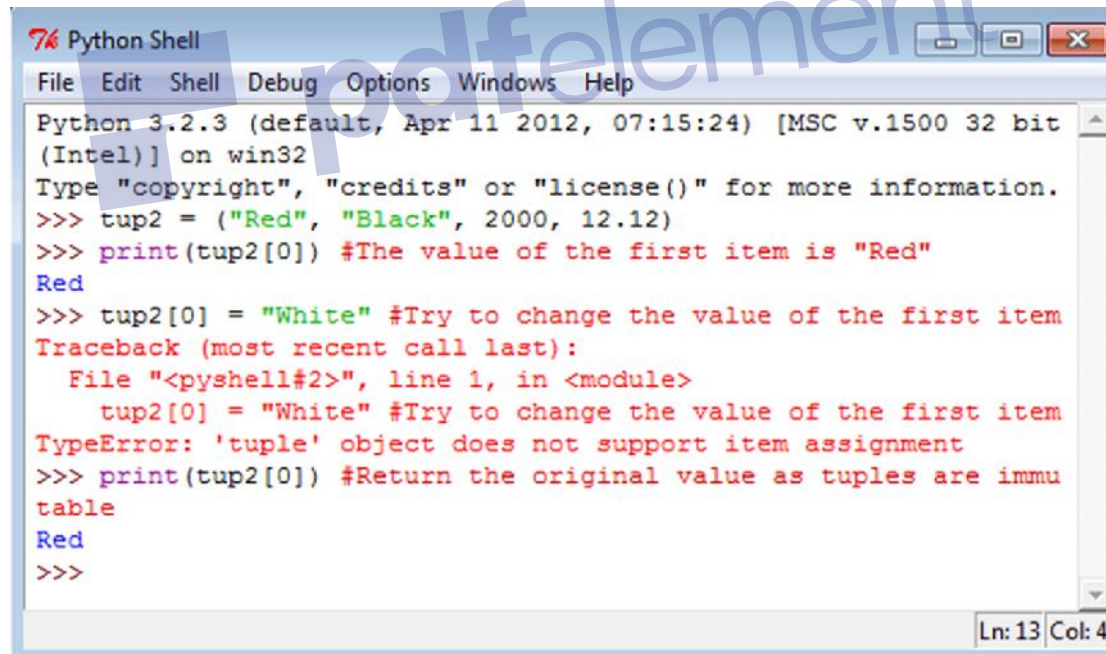


```
Python Shell
File Edit Shell Debug Options Windows Help
Python 3.2.3 (default, Apr 11 2012, 07:15:24) [MSC v.1500 32 bit
(Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> tup2 = ("Red", "Black", 2000, 12.12)
>>> print(tup2)
('Red', 'Black', 2000, 12.12)
>>> print(tup2[0]) #Return first element
Red
>>> print(tup2[3]) #Return last element
12.12
>>> print(tup2[4])
Traceback (most recent call last):
  File "<pyshell#4>", line 1, in <module>
    print(tup2[4])
IndexError: tuple index out of range
>>> |
```

Ln: 15 Col: 4

# Python Data Type

- Tuples are immutable which means it's items values are unchangeable. See the following statements.



```
Python Shell
File Edit Shell Debug Options Windows Help
Python 3.2.3 (default, Apr 11 2012, 07:15:24) [MSC v.1500 32 bit
(Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> tup2 = ("Red", "Black", 2000, 12.12)
>>> print(tup2[0]) #The value of the first item is "Red"
Red
>>> tup2[0] = "White" #Try to change the value of the first item
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    tup2[0] = "White" #Try to change the value of the first item
TypeError: 'tuple' object does not support item assignment
>>> print(tup2[0]) #Return the original value as tuples are immu
table
Red
>>>
```

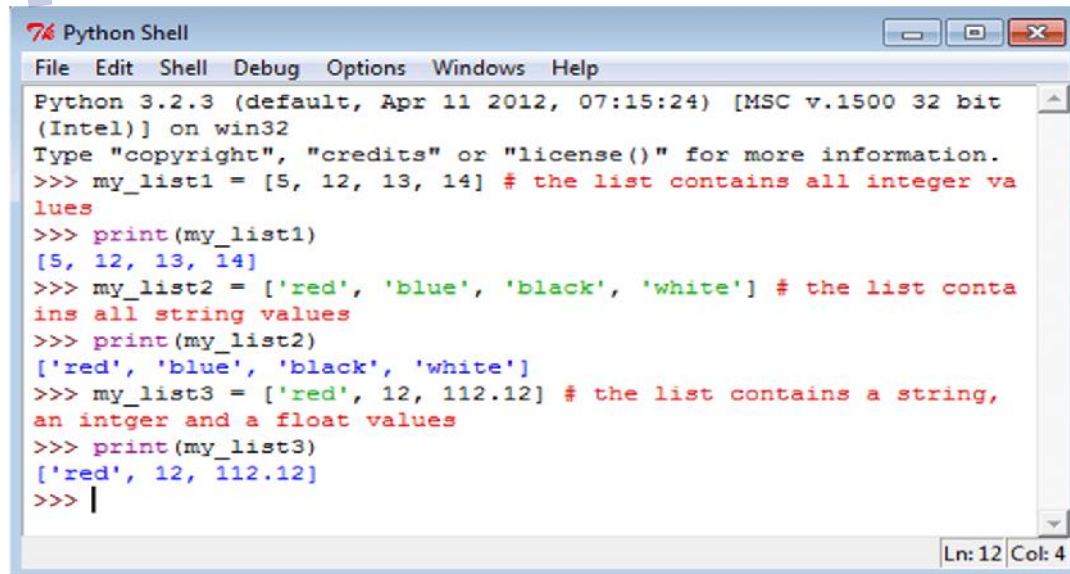
Ln: 13 Col: 4

# Python Data Type

- **Lists**

- A list is a container which holds comma-separated values (items or elements) between square brackets where items or elements need not all have the same type.

- **Creating Lists**



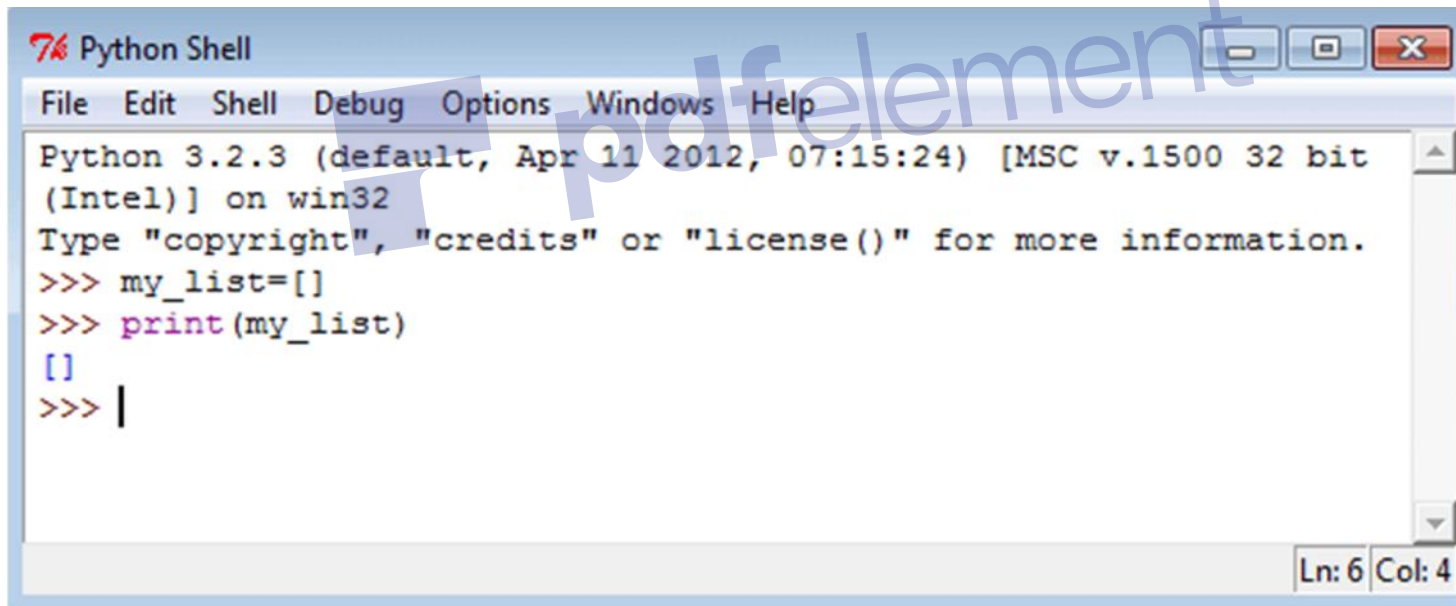
```
Python Shell
File Edit Shell Debug Options Windows Help
Python 3.2.3 (default, Apr 11 2012, 07:15:24) [MSC v.1500 32 bit
(Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> my_list1 = [5, 12, 13, 14] # the list contains all integer va
lues
>>> print(my_list1)
[5, 12, 13, 14]
>>> my_list2 = ['red', 'blue', 'black', 'white'] # the list conta
ins all string values
>>> print(my_list2)
['red', 'blue', 'black', 'white']
>>> my_list3 = ['red', 12, 112.12] # the list contains a string,
an intger and a float values
>>> print(my_list3)
['red', 12, 112.12]
>>> |
```

Ln: 12 Col: 4



# Python Data Type

- A list without any element is called an empty list. See the following statements.



```
Python Shell
File Edit Shell Debug Options Windows Help
Python 3.2.3 (default, Apr 11 2012, 07:15:24) [MSC v.1500 32 bit
(Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> my_list=[]
>>> print(my_list)
[]
>>> |
```

Ln: 6 Col: 4

# Python Operators

# Python Operators

- In computer programming languages operators are special symbols which represent computations, conditional matching etc. The values the operator uses are called operands.

```
c = a + b
```

```
Here a and b are called operands and '+' is an operator
```

```
<
```

# Python Operators

- **Python Arithmetic Operators**

| Operator | Name           | Example | Result   |
|----------|----------------|---------|--|
| +        | Addition       | $x+y$   | Sum of x and y.  |
| -        | Subtraction    | $x-y$   | Difference of x and y.   |
| *        | Multiplication | $x*y$   | Product of x and y.  |
| /        | Division       | $x/y$   | Quotient of x and y.   |
| %        | Modulus        | $x\%y$  | Remainder of x divided by y.   |
| **       | Exponent       | $x**y$  | $x**y$ will give x to the power y  |
| //       | Floor Division | $x/ y$  | The division of operands where the result is the quotient in which the digits after the decimal point are removed. |

# Python Operators

- **Python Comparison Operators**

| Operator | Name                     | Example              | Result  |
|----------|--------------------------|----------------------|---|
| ==       | Equal                    | <code>x==y</code>    | True if x is exactly equal to y.  |
| !=       | Not equal                | <code>x!=y</code>    | True if x is exactly not equal to y.  |
| >        | Greater than             | <code>x&gt;y</code>  | True if x (left-hand argument) is greater than y (right-hand argument).             |
| <        | Less than                | <code>x&lt;y</code>  | True if x (left-hand argument) is less than y (right-hand argument).                |
| >=       | Greater than or equal to | <code>x&gt;=y</code> | True if x (left-hand argument) is greater than or equal to y (right-hand argument). |
| <=       | Less than or equal to    | <code>x&lt;=y</code> | True if x (left-hand argument) is less than or equal to y (right-hand argument).    |

# Python Operators

- **Python Logical Operators**

| Operator | Example   | Result  |
|----------|-----------|---|
| and      | (x and y) | is True if both x and y are true.                                 |
| or       | (x or y)  | is True if either x or y is true.                                 |
| not      | (x not y) | If a condition is true then Logical not operator will make false. |

# Python Operators

- **Python Assignment Operators**

| Operator | Shorthand | Expression | Description   |
|----------|-----------|------------|---|
| +=       | x+=y      | x = x + y  | Adds 2 numbers and assigns the result to left operand.  |
| -=       | x-= y     | x = x -y   | Subtracts 2 numbers and assigns the result to left operand.   |
| *=       | x*= y     | x = x*y    | Multiplies 2 numbers and assigns the result to left operand.  |
| /=       | x/= y     | x = x/y    | Divides 2 numbers and assigns the result to left operand.   |
| %=       | x%= y     | x = x%y    | Computes the modulus of 2 numbers and assigns the result to left operand.                                 |
| **=      | x**=y     | x = x**y   | Performs exponential (power) calculation on operators and assign value to the equivalent to left operand. |
| //=      | x//=y     | x = x//y   | Performs floor division on operators and assign value to the left operand.                                |

# Python Operators

- **Python Bitwise Operators**

| Operator | Shorthand   | Expression   | Description   |
|----------|-------------|--------------|---|
| &        | And         | $x \& y$     | Bits that are set in both x and y are set.          |
|          | Or          | $x   y$      | Bits that are set in either x or y are set.         |
| ^        | Xor         | $x \wedge y$ | Bits that are set in x or y but not both are set.   |
| ~        | Not         | $\sim x$     | Bits that are set in x are not set, and vice versa. |
| <<       | Shift left  | $x \ll y$    | Shift the bits of x, y steps to the left            |
| >>       | Shift right | $x \gg y$    | Shift the bits of x, y steps to the right.          |



# Exercises

- **Exercise 1 (Names)**

Store the names of a few of your friends in a list called names. Print each person's name by accessing each element in the list, one at a time.

- **Exercise 2 (Greetings)**

Start with the list you used in Exercise 3-1, but instead of just printing each person's name, print a message to them. The text of each message should be the same, but each message should be personalized with the person's name.

# Exercises

- **Exercises 3 Seeing the World**

- Think of at least five places in the world you'd like to visit.
- Store the locations in a list. Make sure the list is not in alphabetical order.
- Print your list in its original order. Don't worry about printing the list neatly, just print it as a raw Python list.
- Use `sorted()` to print your list in alphabetical order without modifying the actual list.
- Show that your list is still in its original order by printing it.
- Use `sorted()` to print your list in reverse alphabetical order without changing the order of the original list.
- Show that your list is still in its original order by printing it again.
- Use `reverse()` to change the order of your list. Print the list to show that its order has changed.
- Use `reverse()` to change the order of your list again. Print the list to show it's back to its original order.
- Use `sort()` to change your list so it's stored in alphabetical order. Print the list to show that its order has been changed.
- Use `sort()` to change your list so it's stored in reverse alphabetical order. Print the list to show that its order has changed.

# Solution

- Exo 1

```
names = ['ron', 'tyler', 'dani']
```

```
print(names[0])
```

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

```
print(names[2])
```

pdfelement

# Solution

## Exo 2

```
guests = ['Mohamed', 'Assem', 'Fatima']

name = guests[0].title()
print(name, ", please come to dinner.")

name = guests[1].title()
print(name, ", please come to dinner.")

name = guests[2].title()
print(name, ", please come to dinner.")
```

# Solution

- Exo 3

```
locations = ['himalaya', 'andes', 'tierra del fuego', 'labrador', 'guam']

print("Original order:")
print(locations)

print("\nAlphabetical:")
print(sorted(locations))

print("\nOriginal order:")
print(locations)

print("\nReverse alphabetical:")
print(sorted(locations, reverse=True))

print("\nOriginal order:")
print(locations)

print("\nReversed:")
locations.reverse()
print(locations)
```

# Solution

- Exo 3 (part 2)

```
print("\nOriginal order:")  
locations.reverse()  
print(locations)
```

```
print("\nAlphabetical")  
locations.sort()  
print(locations)
```

```
print("\nReverse alphabetical")  
locations.sort(reverse=True)  
print(locations)
```

pdfelement

# Python if elif else

# Python if elif else

- The if-elif-else statement is used to conditionally execute a statement or a block of statements.
- Conditions can be true or false, execute one thing when the condition is true, something else when the condition is false.



# Python if elif else

- **if statement**

- The Python if statement is same as it is with other programming languages.
- It executes a set of statements conditionally, based on the value of a logical expression.

**Syntax:**

```
if expression :  
    statement_1  
    statement_2  
    .....
```

# Python if elif else

- **if .. else statement**

- In Python if .. else statement, if has two blocks, one following the expression and other following the else clause.

```
if expression :  
    statement_1  
    statement_2  
    ....  
else :  
    statement_3  
    statement_4  
    ....
```

# Python if elif else

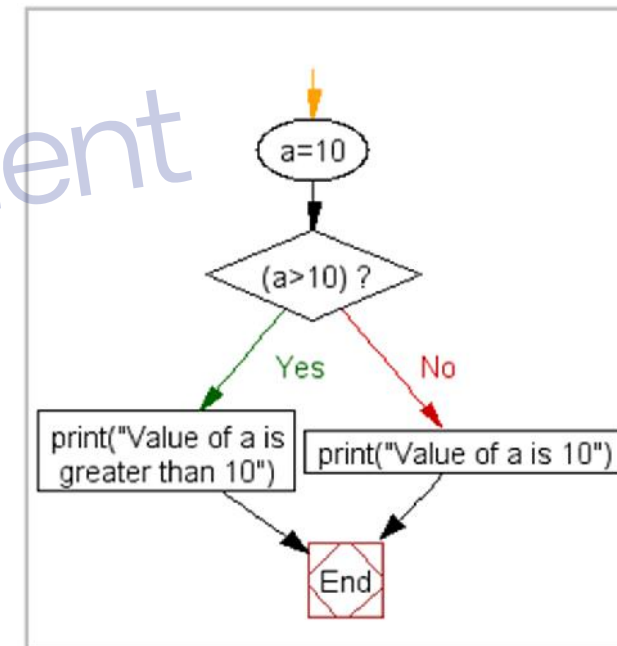
- if .. else statement

```
1 a=10
2 if(a>10):
3     print("Value of a is greater than 10")
4 else :
5     print("Value of a is 10")
```

Output:

```
Value of a is 10
```

Flowchart:



# Python if elif else

- **if .. elif .. else statement**
- In this case ----->  
Python evaluates each expression (i.e. the condition) one by one and if a true condition is found the statement(s) block under that expression will be executed.
- If no true condition is found the statement(s) block under else will be executed.

Syntax:

```
if expression1 :  
    statement_1  
    statement_2  
    ....  
  
elif expression2 :  
    statement_3  
    statement_4  
    ....  
  
elif expression3 :  
    statement_5  
    statement_6  
    .....  
  
else :  
    statement_7  
    statement_8
```

# Python if elif else

- if .. elif .. else statement

```
1 var1 = 1+2j
2 if (type(var1) == int):
3     print("Type of the variable is Integer")
4 elif (type(var1) == float):
5     print("Type of the variable is Float")
6 elif (type(var1) == complex):
7     print("Type of the variable is Complex")
8 elif (type(var1) == bool):
9     print("Type of the variable is Bool")
10 elif (type(var1) == str):
11     print("Type of the variable is String")
12 elif (type(var1) == tuple):
13     print("Type of the variable is Tuple")
14 elif (type(var1) == dict):
15     print("Type of the variable is Dictionaries")
16 elif (type(var1) == list):
17     print("Type of the variable is List")
18 else:
19     print("Type of the variable is Unknown")
```

# Python if elif else

# for loop

- Like most other languages, Python has for loops, but it differs a bit from other like C or Pascal.
- In Python for loop is used to iterate over the items of any sequence including the Python list, string, tuple etc.
- The for loop is also used to access elements from a container (for example list, string, tuple) using built-in function range().

# for loop

- The for loop is also used to access elements from a container (for example list, string, tuple) using built-in function range().

Syntax:

```
for variable_name in sequence :  
    statement_1  
    statement_2  
    .....
```



# for loop

- **Parameter:**

Syntax:

```
for variable_name in sequence :  
    statement_1  
    statement_2  
    ....
```

| Name                              | Description  |
|-----------------------------------|--|
| variable_name                     | It indicates target variable which will set a new value for each iteration of the loop.  |
| sequence                          | A sequence of values that will be assigned to the target variable variable_name. Values are provided using a list or a string or from the built-in function range(). |
| statement_1,<br>statement_2 ..... | Block of program statements.   |

# for loop

- **Example: Python for loop**

```
1 >>> #The list has four elements, indices start at 0 and end at 3
2 >>> color_list = ["Red", "Blue", "Green", "Black"]
3 >>> for c in color_list:
4     print(c)
```

- In the above example `color_list` is a sequence contains a list of various color names.
- When the for loop executed the first item (i.e. Red) is assigned to the variable `c`.
- After this, the print statement will execute and the process will continue until we reach the end of the list.

# for loop

- **Python for loop and range() function**
  - The range() function returns a list of consecutive integers.
  - The function has one, two or three parameters where last two parameters are optional.
  - It is widely used in for loops. Here is the syntax.

```
range(a)  
range(a,b)  
range(a,b,c)
```

<

# for loop

- **range(a)** : Generates a sequence of numbers from 0 to a, excluding a, incrementing by 1.

Syntax:

```
for <variable> in range(<number>):
```

Example:

```
1 >>> for a in range(4):
2     print(a)
3
4     0
5     1
6     2
7     3
8 >>>
```

# for loop

- **range(a,b)**: Generates a sequence of numbers from a to b excluding b, incrementing by 1.

Syntax:

```
for "variable" in range("start_number", "end_number"):
```

Example:

```
1 >>> for a in range(2,7):
2     print(a)
3
4     2
5     3
6     4
7     5
8     6
9 >>>
```

# for loop

- **range(a,b,c):** Generates a sequence of numbers from a to b excluding b, incrementing by c.

Example:

```
1 >>> for a in range(2,19,5):
2     print(a)
3
4     2
5     7
6     12
7     17
8 >>>
```

# for loop

- **Iterating over tuple**

- The following example counts the number of even and odd numbers from a series of numbers.

```
1 numbers = (1, 2, 3, 4, 5, 6, 7, 8, 9) # Declaring the tuple
2 count_odd = 0
3 count_even = 0
4 for x in numbers:
5     if x % 2:
6         count_odd+=1
7     else:
8         count_even+=1
9 print("Number of even numbers :",count_even)
10 print("Number of odd numbers :",count_odd)
```

# for loop

- **Iterating over tuple**

Output:

```
Number of even numbers:4  
Number of odd numbers: 5
```

- In the above example a tuple named numbers is declared which holds the integers 1 to 9.
- The best way to check if a given number is even or odd is to use the modulus operator (%).  
The operator returns the remainder when dividing two numbers.  
Modulus of  $8 \% 2$  returns 0 as 8 is divided by 2, therefore 8 is even and modulus of  $5 \% 2$  returns 1 therefore 5 is odd.
- The for loop iterates through the tuple and we test modulus of  $x \% 2$  is true or not, for every item in the tuple and the process will continue until we reach the end of the tuple.  
When it is true count\_even increase by one otherwise count\_odd is increased by one.  
Finally, we print the number of even and odd numbers through print statements.



# for loop

- **Example: Iterating over list**
  - In the following example for loop iterates through the list "datalist" and prints each item and its corresponding Python type.

```
1 datalist = [1452, 11.23, 1+2j, True, 'w3resource', (0, -1), [5, 12],  
2 {"class":'V', "section":'A'}]  
3 for item in datalist:  
4     print ("Type of ",item, " is ", type(item))
```

# for loop

- **Example: Iterating over list**

Output:

```
Type of 1452 is <class 'int'>
Type of 11.23 is <class 'float'>
Type of (1+2j) is <class 'complex'>
Type of True is <class 'bool'>
Type of w3resource is <class 'str'>
Type of (0, -1) is <class 'tuple'>
Type of [5, 12] is <class 'list'>
Type of {'section': 'A', 'class': 'V'} is <class 'dict'>
```

<

# for loop

- **Example: Iterating over dictionary**
  - In the following example for loop iterates through the dictionary "color" through its keys and prints each key.

```
1 >>> color = {"c1": "Red", "c2": "Green", "c3": "Orange"}
2 >>> for key in color:
3     print(key)
4
5 c2
6 c1
7 c3
8 >>>
```

# for loop

- **Example: Iterating over dictionary**
  - Following for loop iterates through its values :

```
1 >>> color = {"c1": "Red", "c2": "Green", "c3": "Orange"}
2 >>> for value in color.values():
3     print(value)
4
5 Green
6 Red
7 Orange
8 >>>
```

# for loop

- You can attach an optional else clause with for statement, in this case, syntax will be -

```
for variable_name in sequence :  
    statement_1  
    statement_2  
    ....  
else :  
    statement_3  
    statement_4  
    ....
```

- The else clause is only executed after completing the for loop. If a break statement executes in first program block and terminates the loop then the else clause does not execute.

# Python while loop

# While loop

- Loops are used to repeatedly execute a block of program statements. The basic loop structure in Python is while loop. Here is the syntax.

Syntax:

```
while (expression) :  
    statement_1  
    statement_2
```

- The while loop runs as long as the expression (condition) evaluates to True and execute the program block.
- The condition is checked every time at the beginning of the loop and the first time when the expression evaluates to False, the loop stops without executing any remaining statement(s).

# While loop

- The following example prints the digits 0 to 4 as we set the condition  $x < 5$ .

```
1 x = 0;  
2 while (x < 5):  
3     print(x)  
4     x += 1
```

- One thing we should remember that a while loop tests its condition before the body of the loop is executed.
- If the initial test returns false, the body is not executed at all.

```
1 x = 10;  
2 while (x < 5):  
3     print(x)  
4     x += 1
```

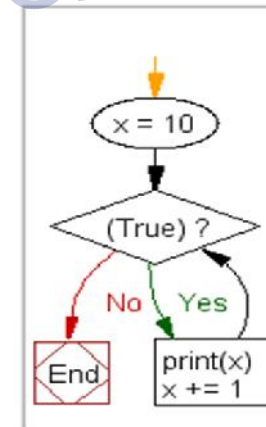


# While loop

- The following while loop is an infinite loop, using True as the condition:

```
1 x = 10;  
2 while (True):  
3     print(x)  
4     x += 1
```

Flowchart:



# While loop

- **while and else statement**

- There is a structural similarity between while and else statement.
- Both have a block of statement(s) which is only executed when the condition is true.
- The difference is that block belongs to if statement executes once whereas block belongs to while statement executes repeatedly.
- You can attach an optional else clause with while statement, in this case, syntax will be

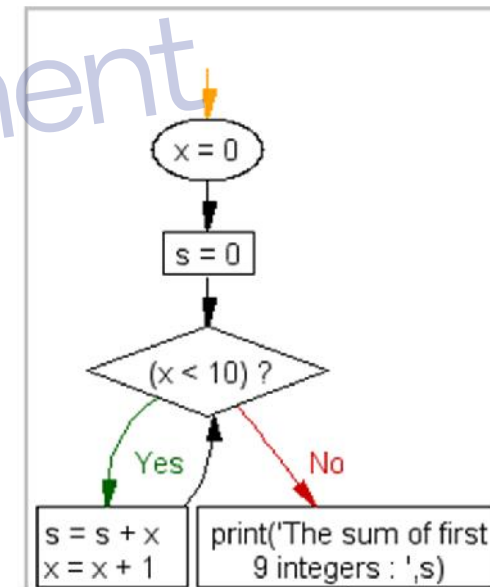
```
while (expression) :  
    statement_1  
    statement_2  
    .....  
else :  
    statement_3  
    statement_4  
    .....
```

- The while loop repeatedly tests the expression (condition) and, if it is true, executes the first block of program statements.
- The else clause is only executed when the condition is false

# While loop

```
1 x = 0;  
2 s = 0  
3 while (x < 10):  
4     s = s + x  
5     x = x + 1  
6 else :  
7     print('The sum of first 9 integers : ',s)
```

Flowchart:



# While loop

- Example: while loop with if-else and break statement

```
1 x = 1;
2 s = 0
3 while (x < 10):
4     s = s + x
5     x = x + 1
6     if (x == 5):
7         break
8 else :
9     print('The sum of first 9 integers : ',s)
10 print('The sum of ',x,' numbers is :',s)
```

- In the above example the loop is terminated when x becomes 5. Here we use break statement to terminate the while loop without completing it, therefore program control goes to outside the while - else structure and execute the next print statement.

# Exercices

- **Exercice 1**

- Une autre boucle while : calculez la somme d'une suite de nombres positifs ou nuls. Comptez combien il y avait de données et combien étaient supérieures à 100.
- Entrer un nombre inférieur ou égal à 0 indique la fin de la suite.

# Exercices

- **Exercice 2**

- L'utilisateur donne un entier positif  $n$  et le programme affiche PAIR s'il est divisible par 2, IMPAIR sinon.

- **Exercice 3**

- L'utilisateur donne un entier positif et le programme annonce combien de fois de suite cet entier est divisible par 2.

# Exercice

- **Exercice 4**

- L'utilisateur donne un entier supérieur à 1 et le programme affiche, s'il y en a, tous ses diviseurs propres sans répétition ainsi que leur nombre. S'il n'y en a pas, il indique qu'il est premier. Par exemple :

```
Entrez un entier strictement positif : 12
Diviseurs propres sans répétition de 12 : 2 3 4 6 (soit 4 diviseurs propres)
Entrez un entier strictement positif : 13
Diviseurs propres sans répétition de 13 : aucun ! Il est premier
```

# Exercice

- Solution 2

```
"""Somme d'entiers et nombre d'entiers supérieurs à 100."""  
  
# Programme principal =====  
somme, nombreTotal, nombreGrands = 0, 0, 0  
  
x = int(input("x (0 pour terminer) ?"))  
while x > 0:  
    somme += x  
    nombreTotal += 1  
    if x > 100:  
        nombreGrands += 1  
    x = int(input("x (0 pour terminer) ?"))  
  
print("\nSomme :", somme)  
print(nombreTotal, "valeur(s) en tout, dont", nombreGrands, "supérieure(s) à 100")
```



# Exercise

- Solution 3

```
"""Parité."""  
  
# Programme principal =====  
n = int(input("Entrez un entier strictement positif :"))  
while n < 1:  
    n = int(input("Entrez un entier STRICTEMENT POSITIF, s.v.p. :"))  
  
if n%2 == 0:  
    print(n, "est pair.")  
else :  
    print(n, "est impair.")
```

# Exercise

- Solution 4

```
"""Nombre de fois qu'un entier est divisible par 2."""  
  
# Programme principal =====  
n = int(input("Entrez un entier strictement positif :"))  
while n < 1:  
    n = int(input("Entrez un entier STRICTEMENT POSITIF, s.v.p. :"))  
save = n  
  
cpt = 0  
while n%2 == 0:  
    n /= 2  
    cpt += 1  
  
print(save, "est", cpt, "fois divisible par 2.")
```

# Exercice

- Solution 4

```
"""Diviseurs propres d'un entier."""  
  
# Programme principal =====  
n = int(input("Entrez un entier strictement positif :"))  
while n < 1:  
    n = int(input("Entrez un entier STRICTEMENT POSITIF, s.v.p. :"))  
  
i = 2 # plus petit diviseur possible de n  
cpt = 0 # initialise le compteur de divisions  
p = n/2 # calculé une fois dans la boucle  
  
print("Diviseurs propres sans répétition de", n, " :", end=' ')  
while i <= p :  
    if n%i == 0:  
        cpt += 1  
        print(i, end=' ')  
    i += 1  
  
if not cpt :  
    print("aucun ! Il est premier.")  
else :  
    print("(soit", cpt, "diviseurs propres)")
```

# Exercice

- **Exercice 5**

- Écrire un programme qui approxime par défaut la valeur de la constante mathématique  $e$ , pour  $n$  assez grand, en utilisant la formule :

$$e \approx \sum_{i=0}^n \frac{1}{i!}$$

- Pour cela, définissez la fonction factorielle et, dans votre programme principal, saisissez l'ordre  $n$  et affichez l'approximation correspondante de  $e$ .

```
"""Approximation de 'e'."""  
  
# Définition de fonction ~~~~~  
def fact(n) :  
    r = 1  
    for i in range(1, n+1) :  
        r *= i  
    return r  
  
# Programme principal =====  
n = int(input("n ?"))  
exp = 0.0  
for i in range(n) :  
    exp = exp + 1.0/fact(i)  
  
print("Approximation de 'e' : { :.3f}".format(exp))
```

# Exercice

- Exercice 6
  - Ecrire un programme python qui demande un nombre de départ, et qui calcule la somme des entiers jusqu'à ce nombre. Par exemple, si l'on entre 5, le programme doit calculer :  $1 + 2 + 3 + 4 + 5 = 15$  NB : on souhaite afficher uniquement le résultat, pas la décomposition du calcul.

# Exercice

- **Solution 5**
- Variables N, i, Som en Entier  
Debut  
Ecrire "Entrez un nombre :"  
Lire N  
Som  $\leftarrow$  0  
Pour i  $\leftarrow$  1 à N  
Som  $\leftarrow$  Som + i  
i Suivant  
Ecrire "La somme est : ", Som  
Fin

# Exercice

- **Exercice 6**

- Ecrire un algorithme qui demande successivement 7 nombres à l'utilisateur, et qui lui dise ensuite quel était le plus grand parmi ces 7 nombres :
  - Entrez le nombre numéro 1 : 12
  - Entrez le nombre numéro 2 : 14 etc.
  - Entrez le nombre numéro 7 : 6
  - Le plus grand de ces nombres est : 14
- Modifiez ensuite l'algorithme pour que le programme affiche de surcroît en quelle position avait été saisie ce nombre :
  - C'était le nombre numéro 2



- **Solution**
- Variables N, i, PG en Entier
- Debut
- $PG \leftarrow 0$
- Pour  $i \leftarrow 1$  à 20
- Ecrire "Entrez un nombre : "
- Lire N
- Si  $i = 1$  ou  $N > PG$  Alors
- $PG \leftarrow N$
- FinSi
- i Suivant
- Ecrire "Le nombre le plus grand était : ", PG
- Fin
- En ligne 3, on peut mettre n'importe quoi dans PG, il suffit que cette variable soit affectée pour que le premier passage en ligne 7 ne provoque pas d'erreur.
- Pour la version améliorée, cela donne :
- Variables N, i, PG, IPG en Entier
- Debut
- $PG \leftarrow 0$
- Pour  $i \leftarrow 1$  à 20
- Ecrire "Entrez un nombre : "
- Lire N
- Si  $i = 1$  ou  $N > PG$  Alors
- $PG \leftarrow N$
- $IPG \leftarrow i$
- FinSi
- i Suivant
- Ecrire "Le nombre le plus grand était : ", PG Ecrire "Il a été saisi en position numéro ", IPG
- Fin

# User defined function



# Python: user defined functions

- In all programming and scripting language, a function is a block of program statements which can be used repetitively in a program.
  - It saves the time of a developer.
- In Python concept of function is same as in other languages.
- There are some built-in functions which are part of Python.

# User defined function

- In Python, a user-defined function's declaration begins with the keyword `def` and followed by the function name.
- The function may take arguments(s) as input within the opening and closing parentheses, just after the function name followed by a colon.
- After defining the function name and arguments(s) a block of program statement(s) start at the next line and these statement(s) must be indented.

# User defined function

- Here is the syntax of a user defined function.

**Syntax:**

```
def function_name(argument1, argument2, ...) :  
    statement_1  
    statement_2  
    ....
```

<

# User defined function

- **Call a function**

- Calling a function in Python is similar to other programming languages, using the function name, parenthesis (opening and closing) and parameter(s).

- See the syntax, followed by an example.

Syntax:

```
function_name(arg1, arg2)
```

<

# User defined function

Example:

```
1 def avg_number(x, y):  
2     print("Average of ",x," and ",y, " is ",(x+y)/2)  
3 avg_number(3, 4)
```

Output:

```
Average of 3 and 4 is 3.5
```

- Lines 1-2 : Details (definition) of the function.
- Line 3 : Call the function.
- Line 1 : Pass parameters :  $x = 3$ ,  $y = 4$
- Line 2 : Print the value of two parameters as well as their average value.

# User defined function

- **Function without arguments:**

Example:

```
1 def printt():  
2     print("This is Python 3.2 Tutorial")  
3     print("This is Python 3.2 Tutorial")  
4     print("This is Python 3.2 Tutorial")  
5 printt()
```

Output:

```
This is Python 3.2 Tutorial  
This is Python 3.2 Tutorial  
This is Python 3.2 Tutorial
```

- Lines 1-4 : Details (definition) of the function.
- Line 5 : Call the function.
- Line 1 : No parameter passes.
- Line 2-4 : Execute three print statements.



# User defined function

- **The Return statement in function**

- The following function returns the square of the sum of two numbers.

```
1 def nsquare(x, y):  
2     return (x*x + 2*x*y + y*y)  
3 print("The square of the sum of 2 and 3 is : ", nsquare(2, 3))  
4
```

- Lines 1-2 : Details (definition) of the function.
- Line 3 : Call the function within a print statement.
- Line 1 : Pass the parameters  $x = 2$ ,  $y = 3$
- Line 2 : Calculate and return the value of  $(x + y)^2$

# User defined function

- **Default Argument Values**

- In function's parameters list we can specify a default value(s) for one or more arguments.
- A default value can be written in the format "argument1 = value",
- therefore we will have the option to declare or not declare a value for those arguments.

# User defined function

- **Default Argument Values**

- The following function returns the square of the sum of two numbers, where default value of the second argument is 2.

```
1 def nsquare(x, y = 2):  
2     return (x*x + 2*x*y + y*y)  
3 print("The square of the sum of 2 and 2 is : ", nsquare(2))  
4 print("The square of the sum of 2 and 3 is : ", nsquare(2,4))
```

Output:

```
The square of the sum of 2 and 2 is: 16  
The square of the sum of 2 and 4 is : 36
```

<

# User defined function

- **Explanation:**

```
1 def nsquare(x, y = 2):  
2     return (x*x + 2*x*y + y*y)  
3 print("The square of the sum of 2 and 2 is : ", nsquare(2))  
4 print("The square of the sum of 2 and 3 is : ", nsquare(2,4))
```

- Lines 1-2 : Details (definition) of the function.
- For first print statement [ Line no 3]
  - Line 3 : Call the function without a second argument, within a print statement.
  - Line 1 : Pass the parameters  $x = 2$ , default value.
  - Line 2 : Calculate and return the value of  $(x + y)^2$
- For second print statement [ Line no 4]
  - Line 3 : Call the function with all arguments, within a print statement.
  - Line 1 : Pass the parameters  $x = 2$ ,  $y = 4$ .
  - Line 2 : Calculate and return the value of  $(x + y)^2$

# User defined function

- **Keyword Arguments:**

Example:

```
1 def marks(english, math = 85, science = 80):
2   print('Marks in : English is - ', english, ', Math - ', math, ', Science - ', science)
3   marks(71, 77)
4   marks(65, science = 74)
5   marks(science = 70, math = 90, english = 75)
```

Output:

```
Marks in : English is - 71 , Math - 77 , Science - 80
Marks in : English is - 65 , Math - 85 , Science - 74
Marks in : English is - 75 , Math - 90 , Science - 70
```

<

# User defined function

- Arbitrary Argument Lists:

Example:

```
1 def sum(*numbers):  
2     s = 0  
3     for n in numbers:  
4         s += n  
5     return s  
6  
7 print(sum(1,2,3,4))
```

Output:

```
10
```