

Brief Python

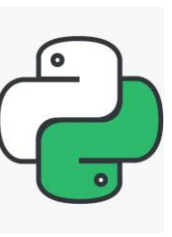
First Python Course for Beginners

Chapter 1: Basic Data Types and Operations

Dr. Eric Chou

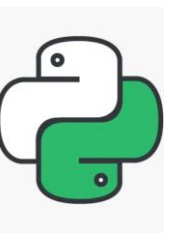
IEEE Senior Member





Objectives

- Python Installation
- Python Idle



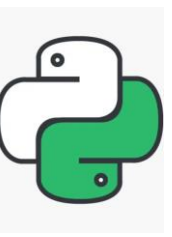
Python Language

- Python is a general-purpose high-level programming language. It is an open-source language, released under a GPL-compatible license.
- These chapters will help you learn Python 3 step by step.
- These chapters are designed for beginners who want to learn Python programming language.



Installation

LECTURE 1

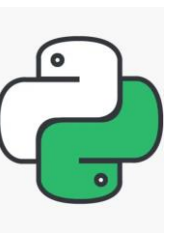


Python

<https://www.python.org/>



- Python is the number 1 language for data science and AI
- Python is the most rapidly growing open-source programming language.
- Python is available for most operating systems, including Windows, UNIX, Linux, and Mac OS.



Python Installation

1. Python Interpreter
2. Python idle

Python

PSF

Docs

PyPI

Jobs

Community



Search

GO

Socialize

About

Downloads

Documentation

Community

Success Stories

News

Events

Download the latest version for Windows

Download Python 3.7.0

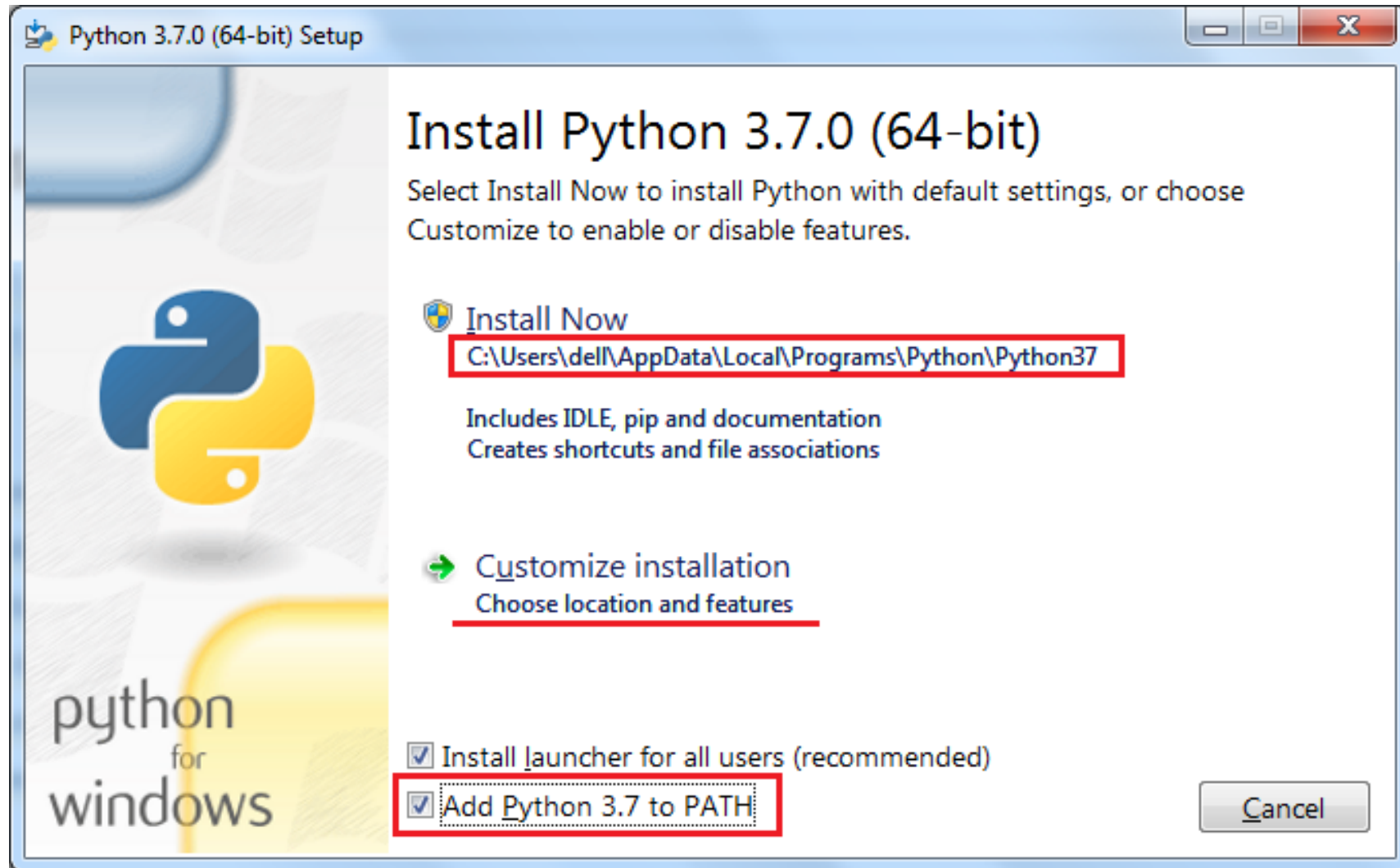
Looking for Python with a different OS? Python for [Windows](#), [Linux/UNIX](#), [Mac OS X](#), [Other](#)

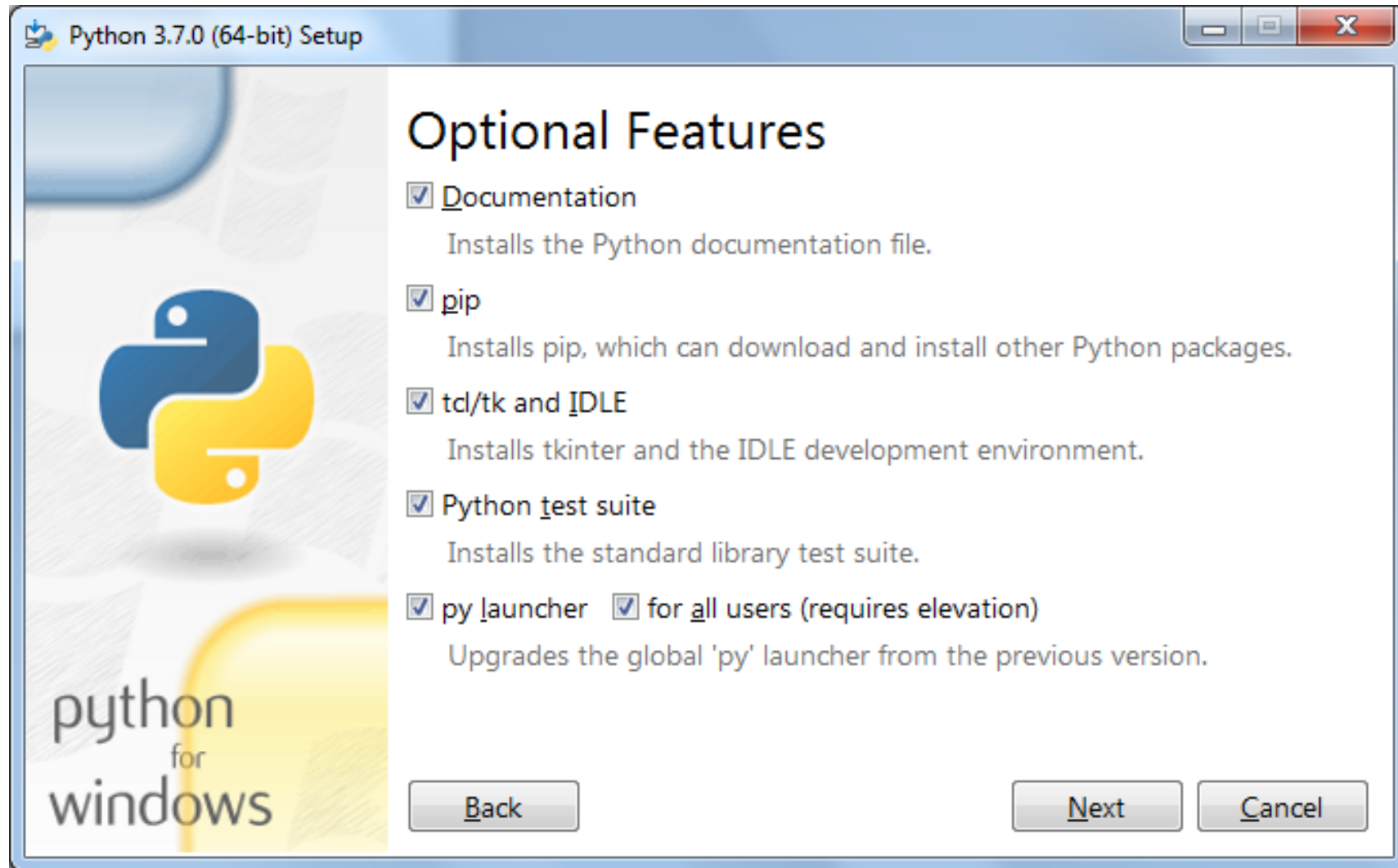
Want to help test development versions of Python? [Pre-releases](#)

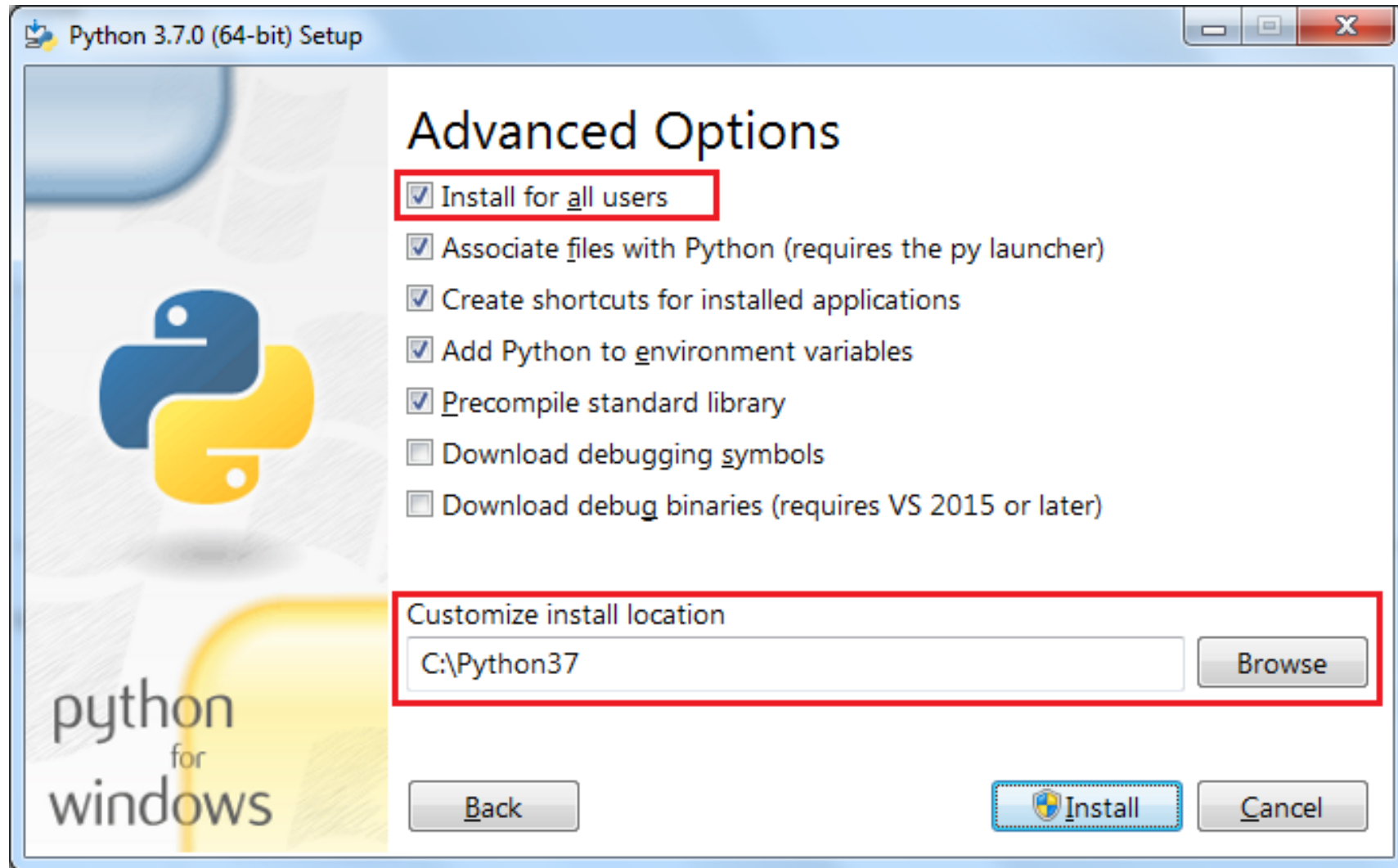
Looking for Python 2.7? See below for specific releases

Python Releases for Windows

- [Latest Python 3 Release - Python 3.7.0](#)
- [Latest Python 2 Release - Python 2.7.15](#)
- [Python 3.7.0 - 2018-06-27](#)
 - [Download Windows x86 web-based installer](#)
 - [Download Windows x86 executable installer](#)
 - [Download Windows x86 embeddable zip file](#)
 - [Download Windows x86-64 web-based installer](#)
 - [Download Windows x86-64 executable installer](#)
 - [Download Windows x86-64 embeddable zip file](#)
 - [Download Windows help file](#)
- [Python 3.6.6 - 2018-06-27](#)









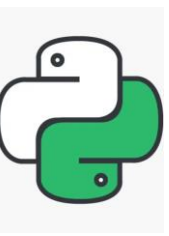
Python Interpreter and Idle

LECTURE 1



Python Interpreter

ACTIVITY



Python - Shell (Interpreter)

- Python is an interpreter language. It means it executes the code line by line. Python provides a Python Shell (also known as Python Interactive Shell) which is used to execute a single Python command and get the result.
- Python Shell waits for the input command from the user. As soon as the user enters the command, it executes it and displays the result.
- To open the Python Shell on Windows, open the command prompt, write **python** and press **enter**.

C:\Windows\system32\cmd.exe - python



Microsoft Windows [Version 6.1.7601]

Copyright (c) 2009 Microsoft Corporation. All rights reserved.

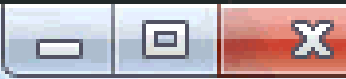
C:\Users\dell>python

Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:59:51) [MSC v.1914 64 bit (AMD64)] on win32

Type "help", "copyright", "credits" or "license" for more information.

>>> _

C:\Windows\system32\cmd.exe - python



Microsoft Windows [Version 6.1.7601]

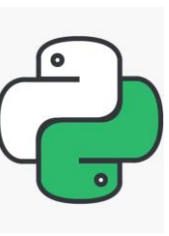
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\dell>python

Python 3.7.0 (tags/v3.7.0:1bf9cc5093, Jun 27 2018, 04:59:51) [MSC v.1914 64 bit (AMD64)] on win32

Type "help", "copyright", "credits" or "license" for more information.

>>>



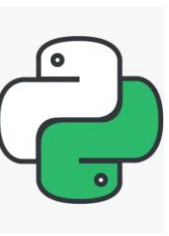
Execute Python Script

As you have seen above, Python Shell executes a single statement. To execute multiple statements, create a Python file with extension .py, and write Python scripts (multiple statements).

For example, enter the following statement in a text editor such as Notepad++.

Example: myPythonScript.py

```
print ("This is Python Script.")  
print ("Welcome to Python Tutorial by eCode24.com")
```



Execute Python Script

Save it as myPythonScript.py, navigate command prompt to the folder where you have saved this file and execute the python myPythonScript.py command, as shown below.

A screenshot of a Windows command prompt window. The title bar shows the path 'C:\Windows\system32\cmd.exe'. The command prompt shows the directory 'D:\python' and the command 'python myPythonScript.py' being executed. The output of the script is displayed: 'This is Python Script.' followed by 'Welcome to Python Tutorial by TutorialsTeacher.com'. The prompt then returns to 'D:\python>_'. A red rectangular box highlights the command and its output.

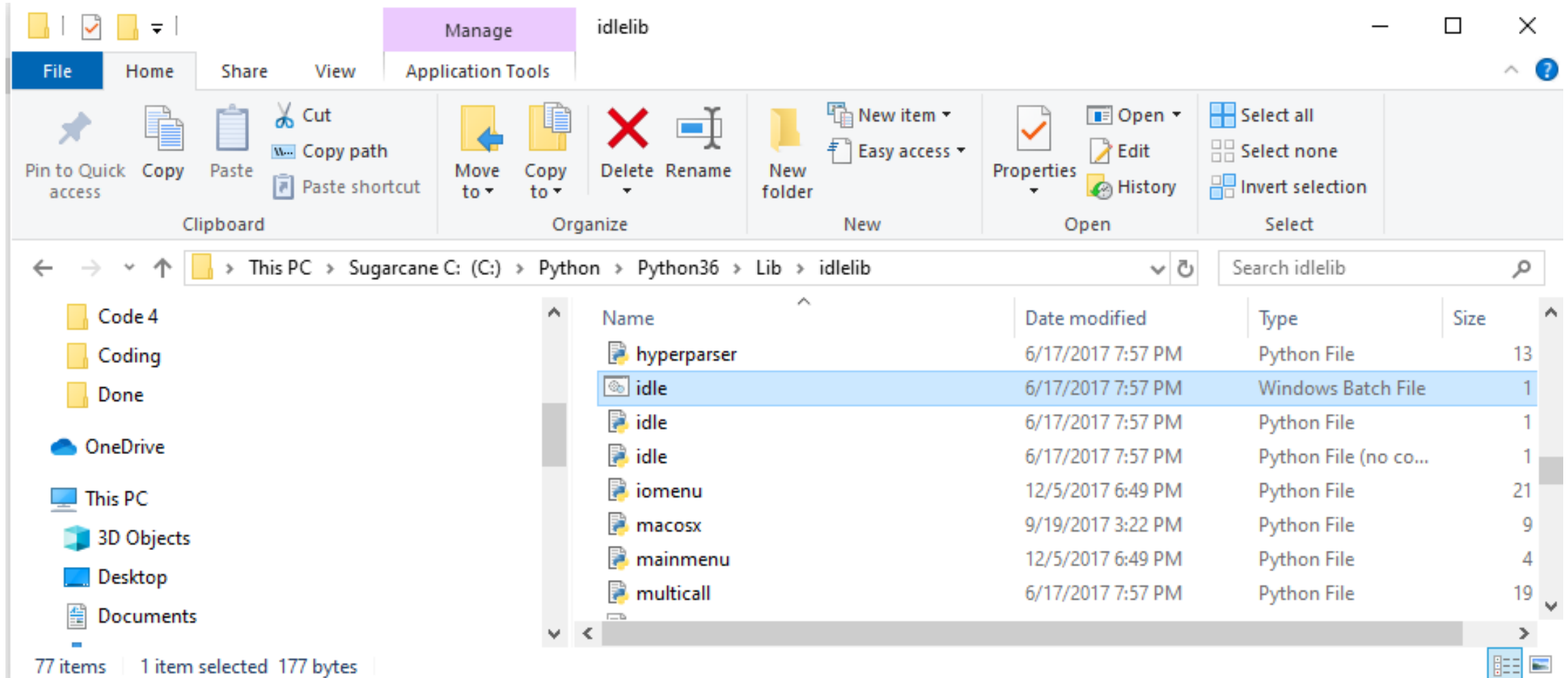
```
C:\Windows\system32\cmd.exe
D:\python>python myPythonScript.py
This is Python Script.
Welcome to Python Tutorial by TutorialsTeacher.com
D:\python>_
```

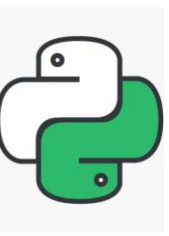
Thus, you can execute Python expressions and commands using Python Shell. We will be using Python Shell to demo Python features through out these tutorials.



Idle

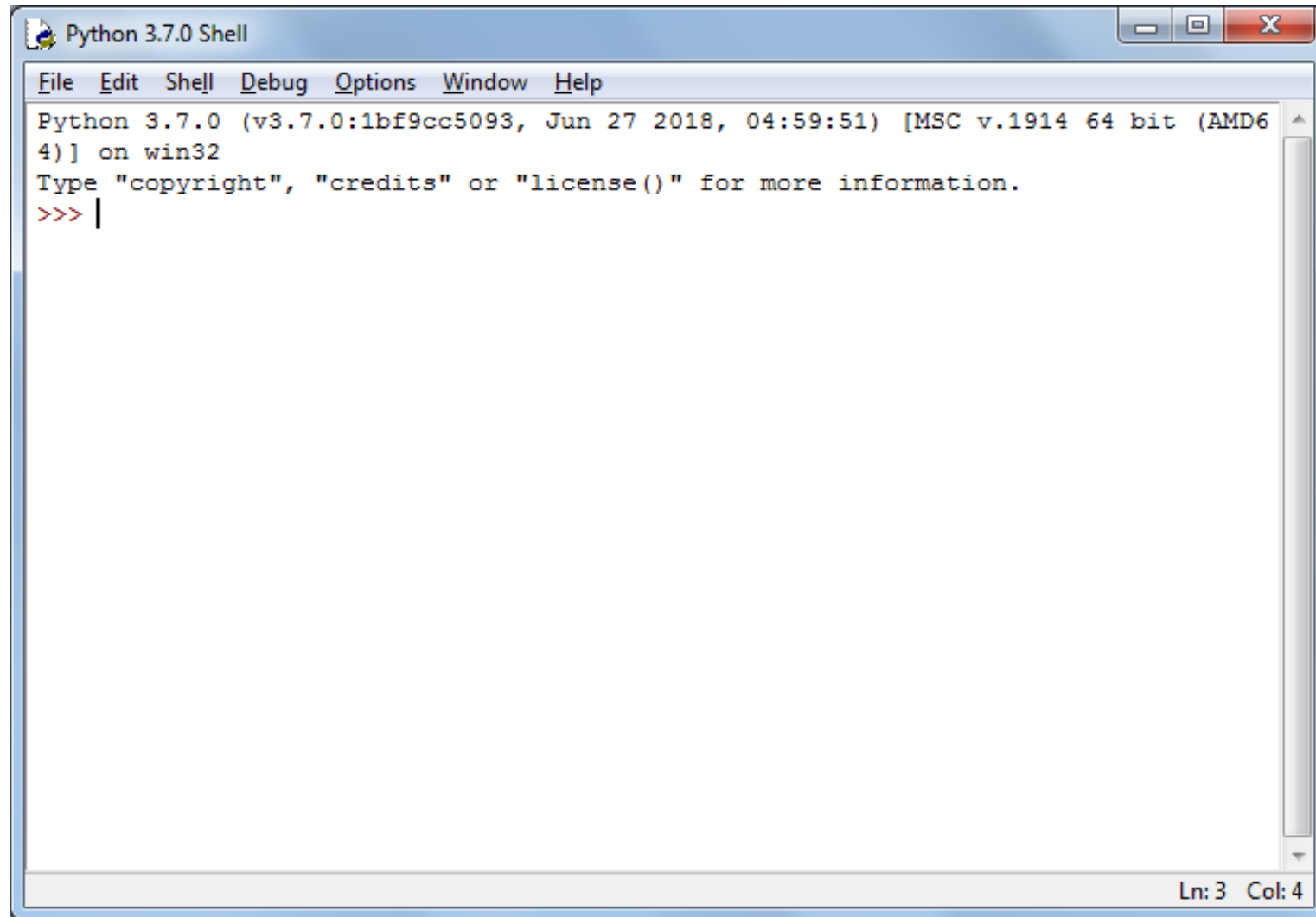
ACTIVITY





Python - IDLE

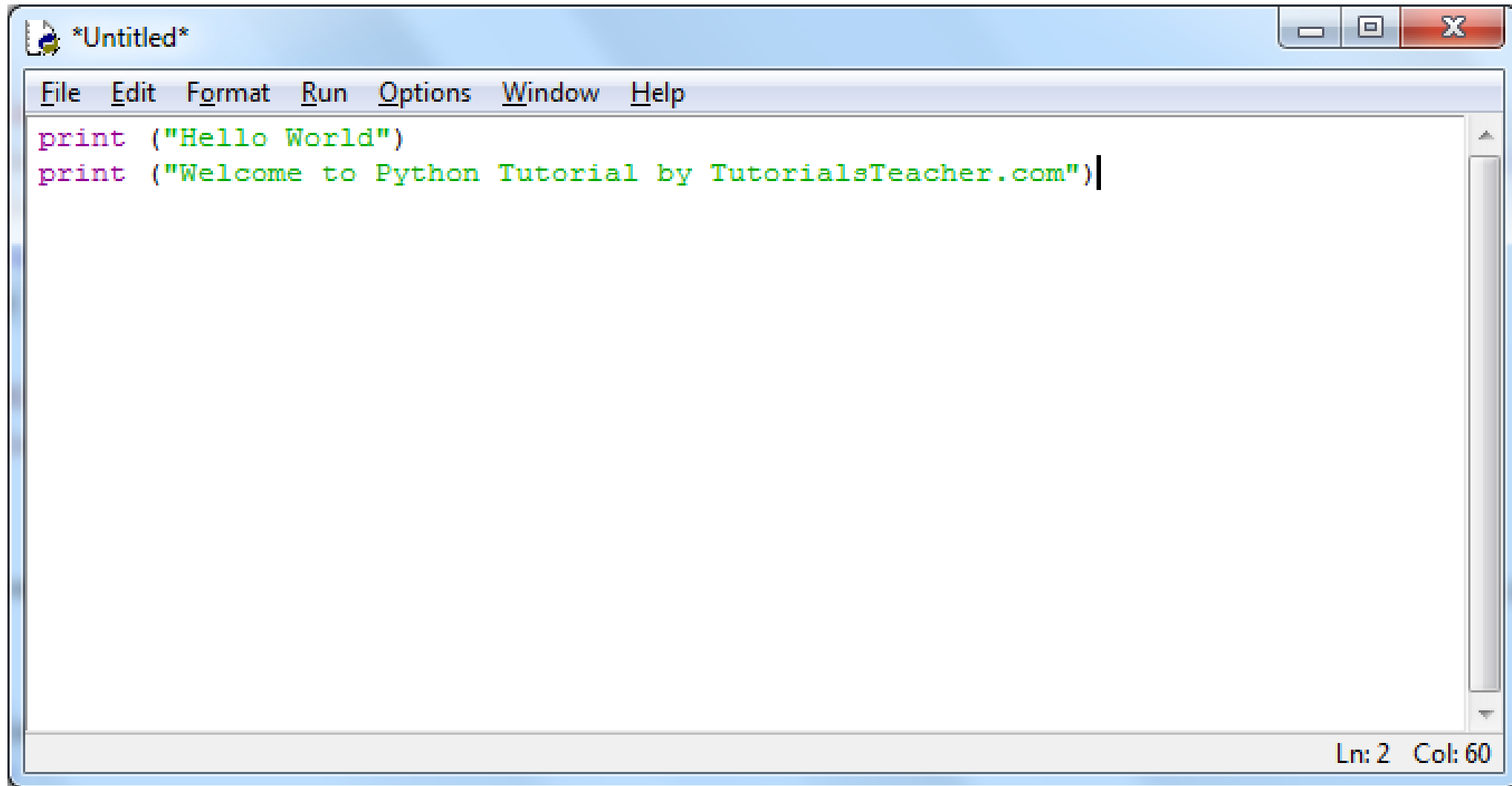
- IDLE (Integrated Development and Learning Environment) is an integrated development environment (IDE) for Python. The Python installer for Windows contains the IDLE module by default.
- IDLE can be used to execute a single statement just like Python Shell and also to create, modify and execute Python scripts. IDLE provides a fully-featured text editor to create Python scripts that includes features like syntax highlighting, autocompletion and smart indent. It also has a debugger with stepping and breakpoints features.

A screenshot of a Python 3.7.0 Shell window. The window has a title bar with the text "Python 3.7.0 Shell" and standard Windows window controls (minimize, maximize, close). Below the title bar is a menu bar with the following items: File, Edit, Shell, Debug, Options, Window, and Help. The main text area contains the following text:

```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:59:51) [MSC v.1914 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> |
```

A vertical scrollbar is visible on the right side of the text area. At the bottom right of the window, the status bar shows "Ln: 3 Col: 4".

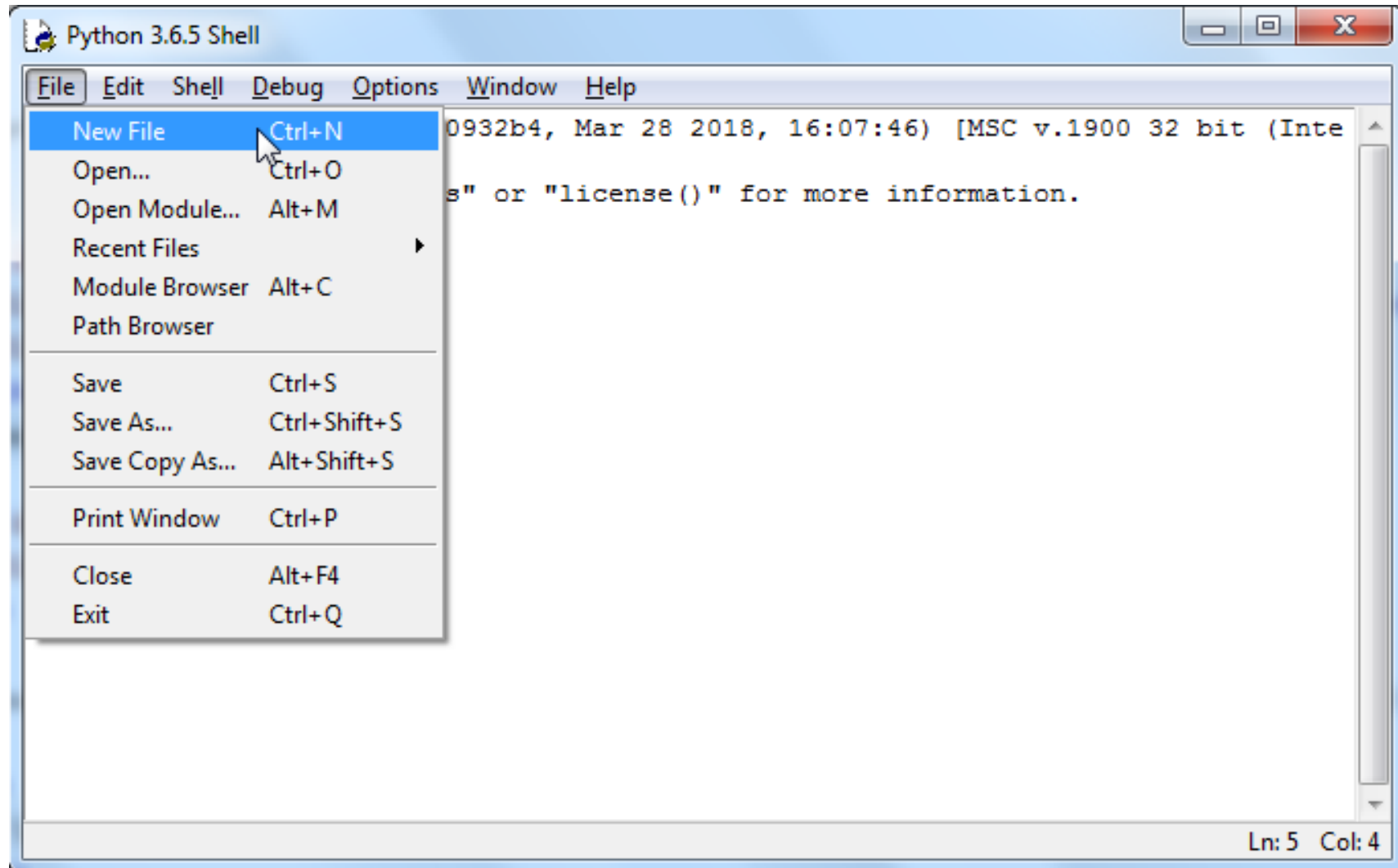
```
Python 3.7.0 Shell
File Edit Shell Debug Options Window Help
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:59:51) [MSC v.1914 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> |
Ln: 3 Col: 4
```

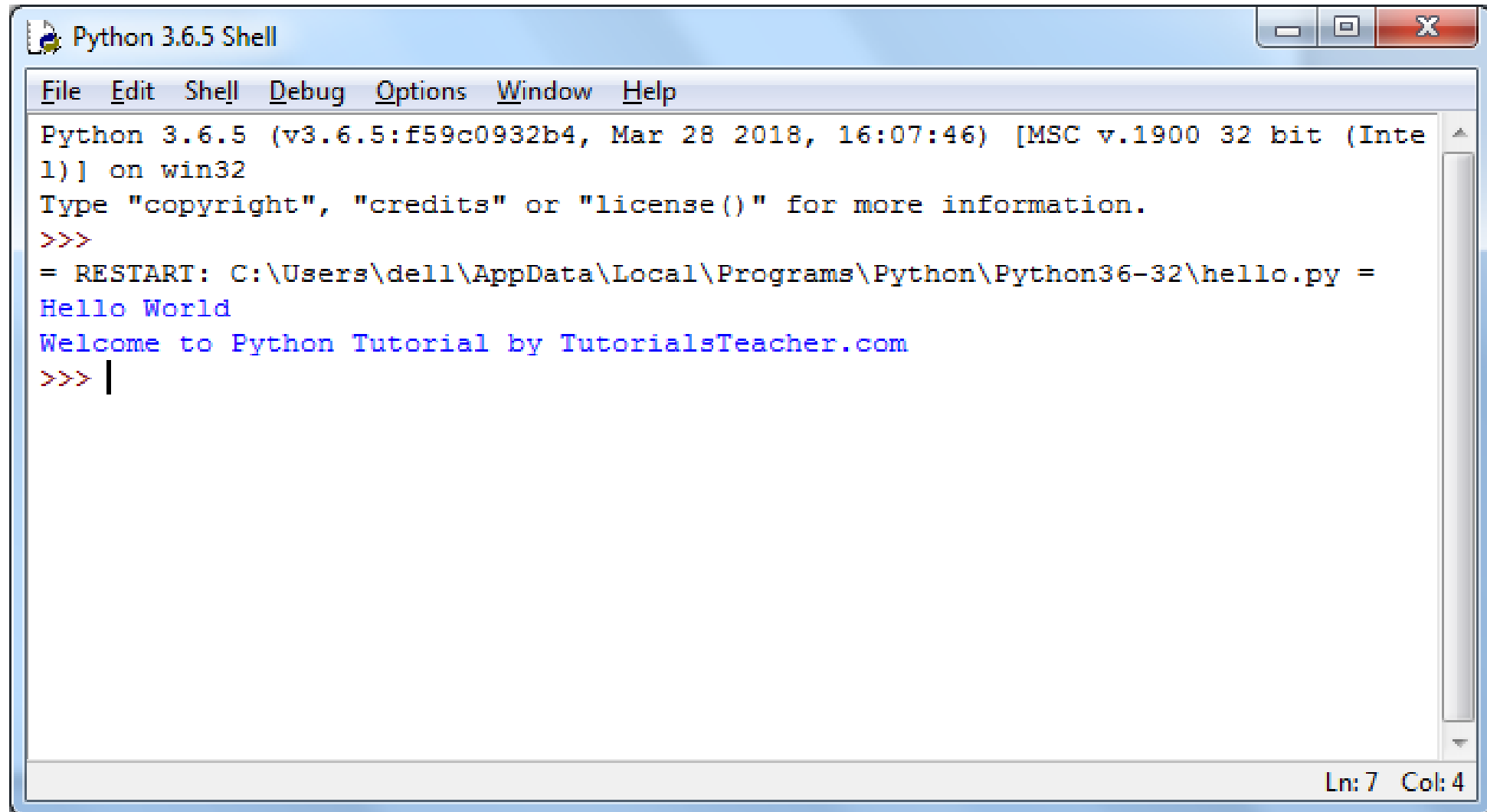


The image shows a screenshot of a Python IDE window titled "*Untitled*". The window has a menu bar with the following options: File, Edit, Format, Run, Options, Window, and Help. The main text area contains two lines of Python code:

```
print ("Hello World")  
print ("Welcome to Python Tutorial by TutorialsTeacher.com")
```

The status bar at the bottom right of the window indicates the current cursor position: "Ln: 2 Col: 60".





A screenshot of a Python 3.6.5 Shell window. The window has a title bar with the text "Python 3.6.5 Shell" and standard Windows window controls (minimize, maximize, close). Below the title bar is a menu bar with the following items: File, Edit, Shell, Debug, Options, Window, and Help. The main text area contains the following text: "Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 16:07:46) [MSC v.1900 32 bit (Intel)] on win32", "Type 'copyright', 'credits' or 'license()' for more information.", and three red prompt characters ">>>". Below the prompt characters, the text "= RESTART: C:\Users\dell\AppData\Local\Programs\Python\Python36-32\hello.py =" is displayed. This is followed by two lines of blue text: "Hello World" and "Welcome to Python Tutorial by TutorialsTeacher.com". The prompt characters ">>> |" are shown at the end of the line. A vertical scrollbar is visible on the right side of the text area. At the bottom right of the window, the status bar shows "Ln: 7 Col: 4".

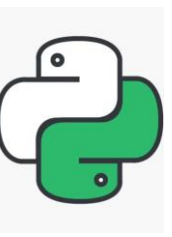
```
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 16:07:46) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:\Users\dell\AppData\Local\Programs\Python\Python36-32\hello.py =
Hello World
Welcome to Python Tutorial by TutorialsTeacher.com
>>> |
```

Ln: 7 Col: 4



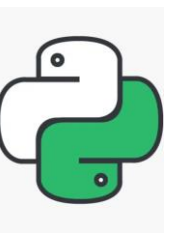
Basic Syntax

LECTURE 1



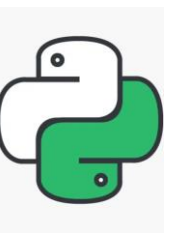
Python - Basic Syntax

- Vocabulary / Words - Variables and Reserved words
- Sentence structure - valid syntax patterns
- Program structure - constructing a program for a purpose



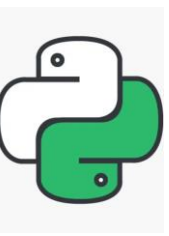
Python Identifiers

- A Python program can have variables, functions, classes, modules, packages etc.
- **Identifier** is the name given to these programming elements. An identifier should start with either an alphabet letter (lower or upper case) or an underscore (_). After that, more than one alphabet letters (a-z or A-Z), digits (0-9) or underscores may be used to form an identifier.
- No other characters are allowed.



Python Identifiers

- Conventionally, the name of the class begins with an uppercase alphabet letter. Others start with lowercase alphabet letters.
- Use of one or two underscore characters has a special significance when naming the instance attributes of a class. More about this will follow in the discussion about inheritance.
- Two leading and trailing underscores are used in the language itself for a special purpose. For example (e.g. `__add__`, `__init__`)

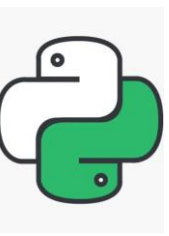


Python Statement

By default, the Python interpreter treats a piece of text terminated by hard carriage return (new line character) as one statement. It means each line in a Python script is a statement. (Just as in C/C++/C#, a semicolon ; denotes the end of a statement).

Example: Python Statements

```
msg="Hello World"  
code=123  
name="Steve"
```

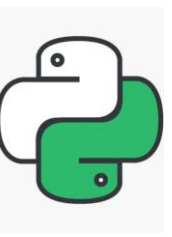


Python Statement

However, you can show the text spread over more than one lines to be a single statement by using the backslash (\) as a continuation character. Look at the following examples:

Example: Continuation of Statement

```
msg="Hello Pythonista \  
Welcome to Python Tutorial \  
from TutorialsTeacher.com"
```

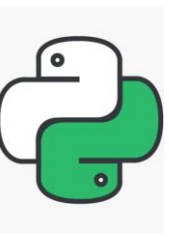


Python Statement

Similarly, use the semicolon ; to write multiple statements in a single line.

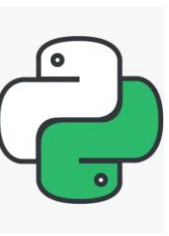
Example: Multiple Statements in Single Line

```
msg="Hello World";code=123;name="Steve"
```

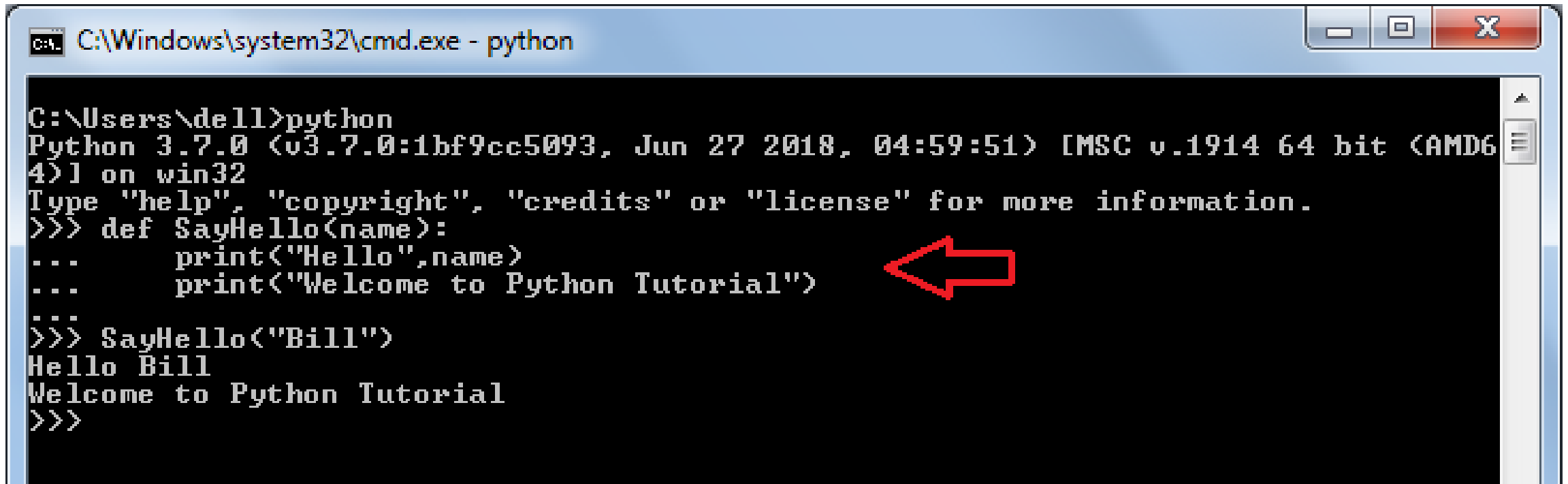
Indents in Python

- Many times it is required to construct a block of more than one statements. For example there are usually multiple statements that are part of the definition of a function. There can be one or more statements in a looping construct.
- Different programming languages use different techniques to define the scope and extent of a block of statements in constructs like class, function, conditional and loop. In C, C++, C# or Java, statements inside curly brackets { and } are treated as a block.



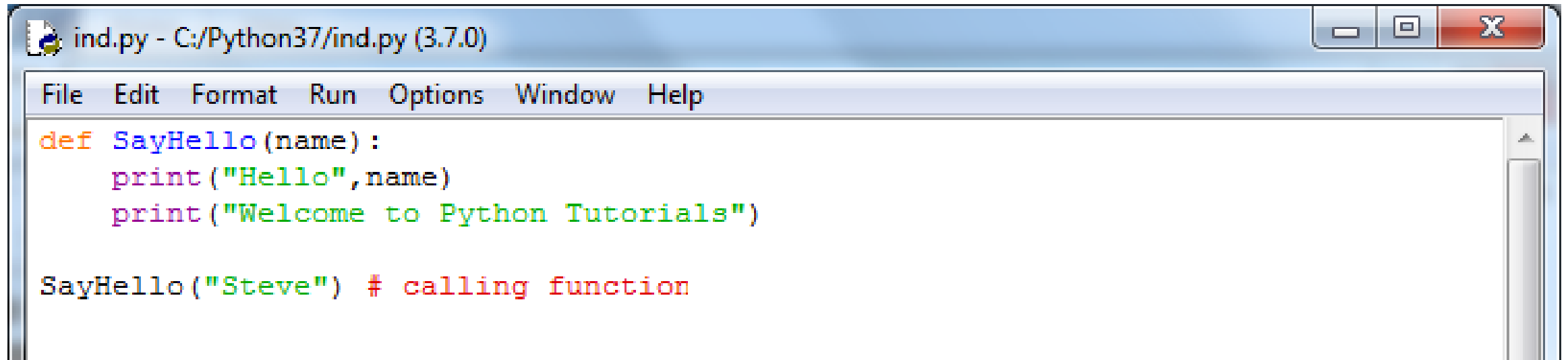
Indents in Python

- Python uses uniform indentation to denote a block of statements. When a block is to be started, type the exclamation symbol (:) and press Enter. Any Python-aware editor (like IDLE) goes to the next line leaving an additional whitespace (called indent). Subsequent statements in the block follow the same level of indent.
- In order to signal the end of a block, the whitespace is de-dented by pressing the backspace key. If your editor is not configured for Python, you may have to ensure that the statements in a block have the same indentation level by pressing the spacebar or Tab key. The Python interpreter will throw an error if the indentation level in the block is not same.



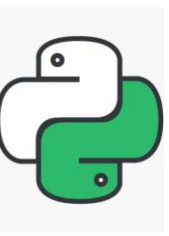
```
C:\Windows\system32\cmd.exe - python

C:\Users\dell>python
Python 3.7.0 (tags/3.7.0:1bf9cc5093, Jun 27 2018, 04:59:51) [MSC v.1914 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> def SayHello(name):
...     print("Hello",name)
...     print("Welcome to Python Tutorial")
...
>>> SayHello("Bill")
Hello Bill
Welcome to Python Tutorial
>>>
```



```
ind.py - C:/Python37/ind.py (3.7.0)
File Edit Format Run Options Window Help
def SayHello(name):
    print("Hello", name)
    print("Welcome to Python Tutorials")

SayHello("Steve") # calling function
```



Comments in Python

In a Python script, the symbol `#` indicates the start of a comment line. It is effective till the end of the line in the editor. If `#` is the first character of the line, then the entire line is a comment. It can be used also in the middle of a line. The text before it is a valid Python expression, while the text following is treated as a comment.

Example: Comments

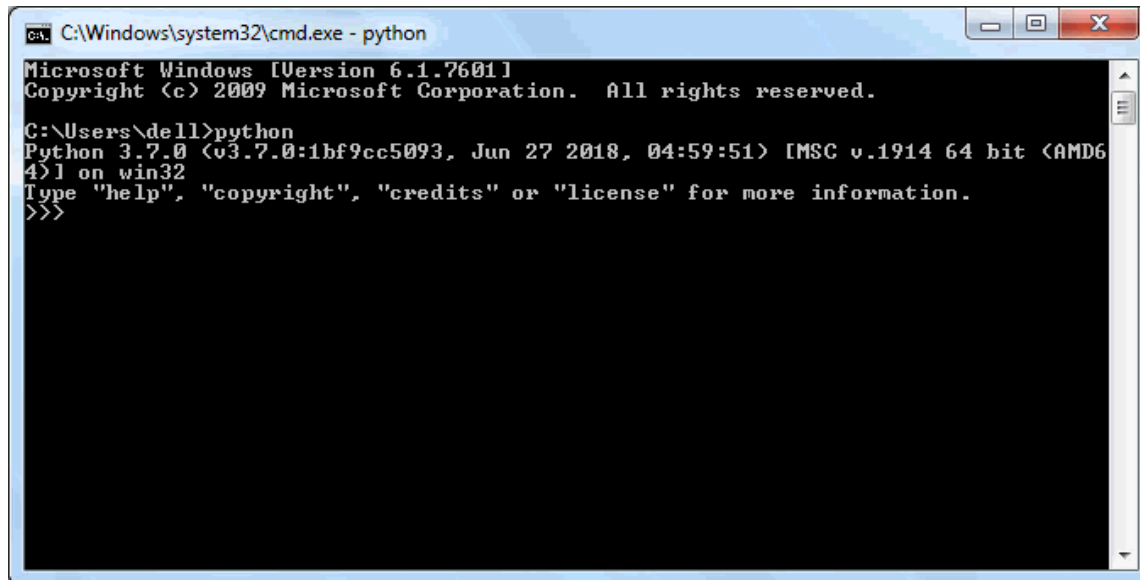
```
# this is a comment  
print ("Hello World")  
print ("Welcome to Python Tutorial") #this is also a comment but after a statement.
```

Example: Multi-line Comments

```
'''  
comment1  
comment2  
comment3  
'''  
print ("Hello World")
```

Getting the User's Input

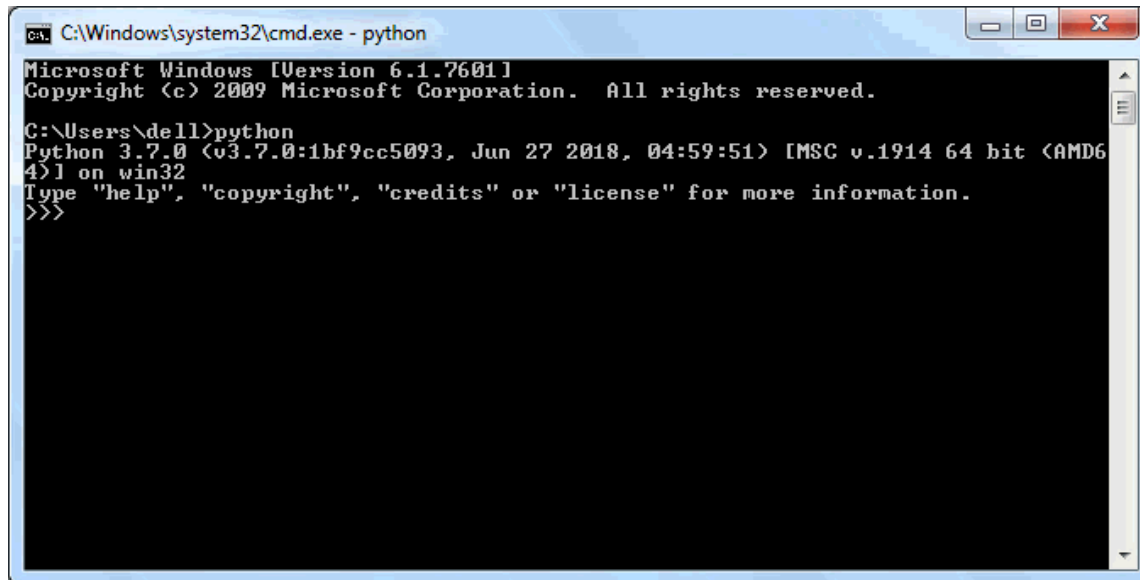
The **input()** function is a part of the core library of standard Python distribution. It reads the key strokes as a string object which can be referred to by a variable having a suitable name.

A screenshot of a Windows command prompt window titled "C:\Windows\system32\cmd.exe - python". The window shows the output of running the 'python' command. The text displayed is: "Microsoft Windows [Version 6.1.7601] Copyright (c) 2009 Microsoft Corporation. All rights reserved. C:\Users\dell>python Python 3.7.0 (tags/v3.7.0:1bf9cc5093, Jun 27 2018, 04:59:51) [MSC v.1914 64 bit (AMD64)] on win32 Type "help", "copyright", "credits" or "license()" for more information. >>>". The prompt is currently at the third set of greater-than signs, ready for user input.

```
C:\Windows\system32\cmd.exe - python
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\dell>python
Python 3.7.0 (tags/v3.7.0:1bf9cc5093, Jun 27 2018, 04:59:51) [MSC v.1914 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
```

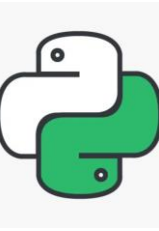
Getting the User's Input

A screenshot of a Windows command prompt window. The title bar reads 'C:\Windows\system32\cmd.exe - python'. The window content shows the following text:

```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\dell>python
Python 3.7.0 <v3.7.0:1bf9cc5093, Jun 27 2018, 04:59:51> [MSC v.1914 64 bit <AMD64>] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
```

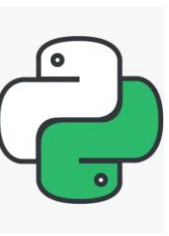
In the above example, the **input()** function takes the user's input from the next line, e.g. 'Steve' in this case. **input()** will capture it and assign it to a name variable. The **name** variable will display whatever the user has provided as the input.



Getting the User's Input

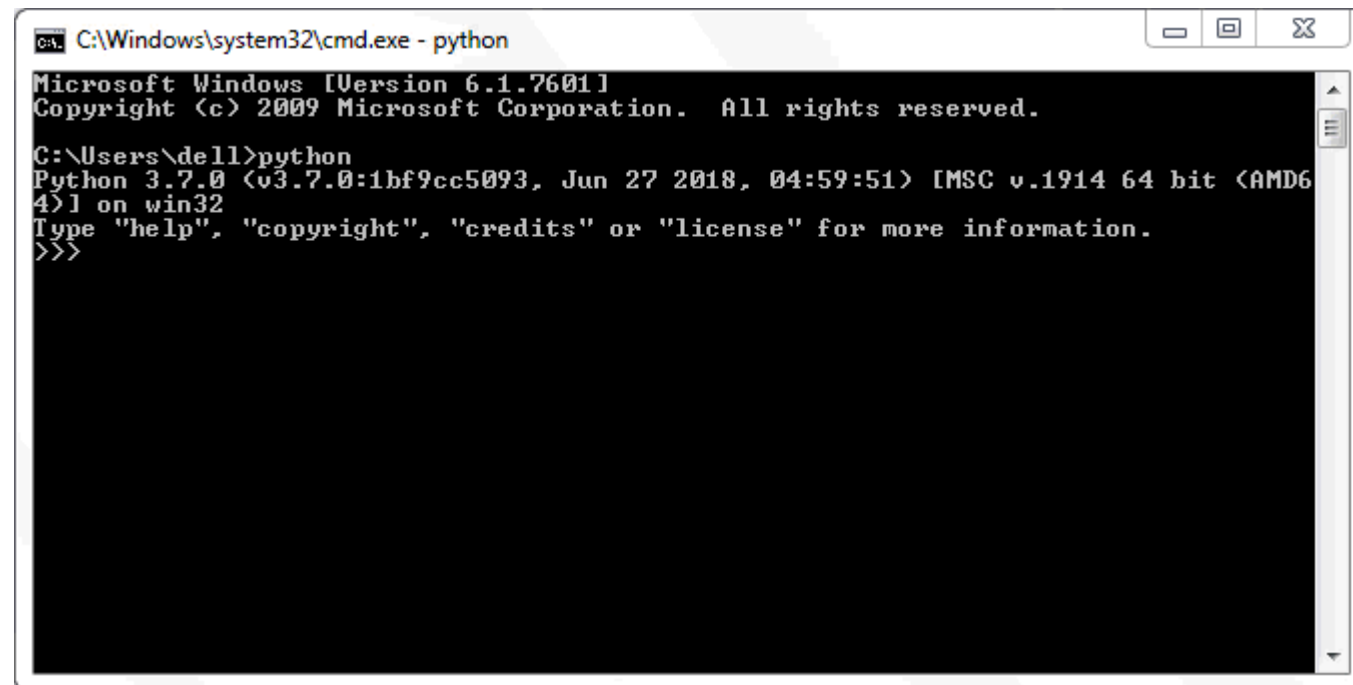
The `input()` function always reads the input as a string, even if it comprises of digits. The `type()` function used earlier confirms this behavior.

```
>>> name=input("Enter your name: ")
Enter your name: Steve
>>> type(name)
<class 'str'>
>>> age=input("Enter your age: ")
Enter your age: 21
>>> type(age)
<class 'str'>
```



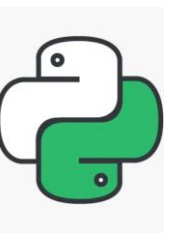
Display the Output

Another built-in function **print()** serves as an output statement in Python. It echoes the value of any Python expression on the Python shell.

A screenshot of a Windows command prompt window titled "C:\Windows\system32\cmd.exe - python". The window shows the output of running the "python" command. The text displayed is: "Microsoft Windows [Version 6.1.7601] Copyright (c) 2009 Microsoft Corporation. All rights reserved. C:\Users\dell>python Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:59:51) [MSC v.1914 64 bit (AMD64)] on win32 Type "help", "copyright", "credits" or "license" for more information. >>>". The prompt is now ">>>".

```
C:\Windows\system32\cmd.exe - python
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

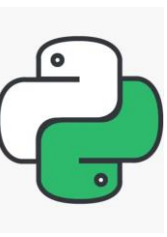
C:\Users\dell>python
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:59:51) [MSC v.1914 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```



Display the Output

Multiple values can be displayed by the single `print()` function separated by comma. The following example displays values of `name` and `age` variables using the single **`print()`** function.

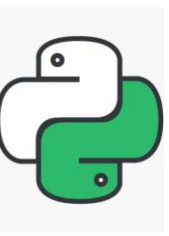
```
>>> name="Ram"
>>> age=21
>>> print("Name:", name, "Age:",age)
Name: Ram Age: 21
```



Display the Output

By default, a single space (' ') acts as a separator between values. However, any other character can be used by providing a sep parameter. In the following example, "=" is used as a separator character.

```
>>> name="Ram"
>>> age=21
>>> print(name,age)
Ram 21
>>> print(name,age,sep=",")
Ram,21
```



Display the Output

The output of the `print()` function always ends by a newline character. The `print()` function has another optional parameter `end`, whose default value is `"\n"`. This value can be substituted by any other character such as a single space (' ') to display the output of the subsequent `print()` statement in the same line. This is especially useful in a Python script like the one shown below:

Example: display.py

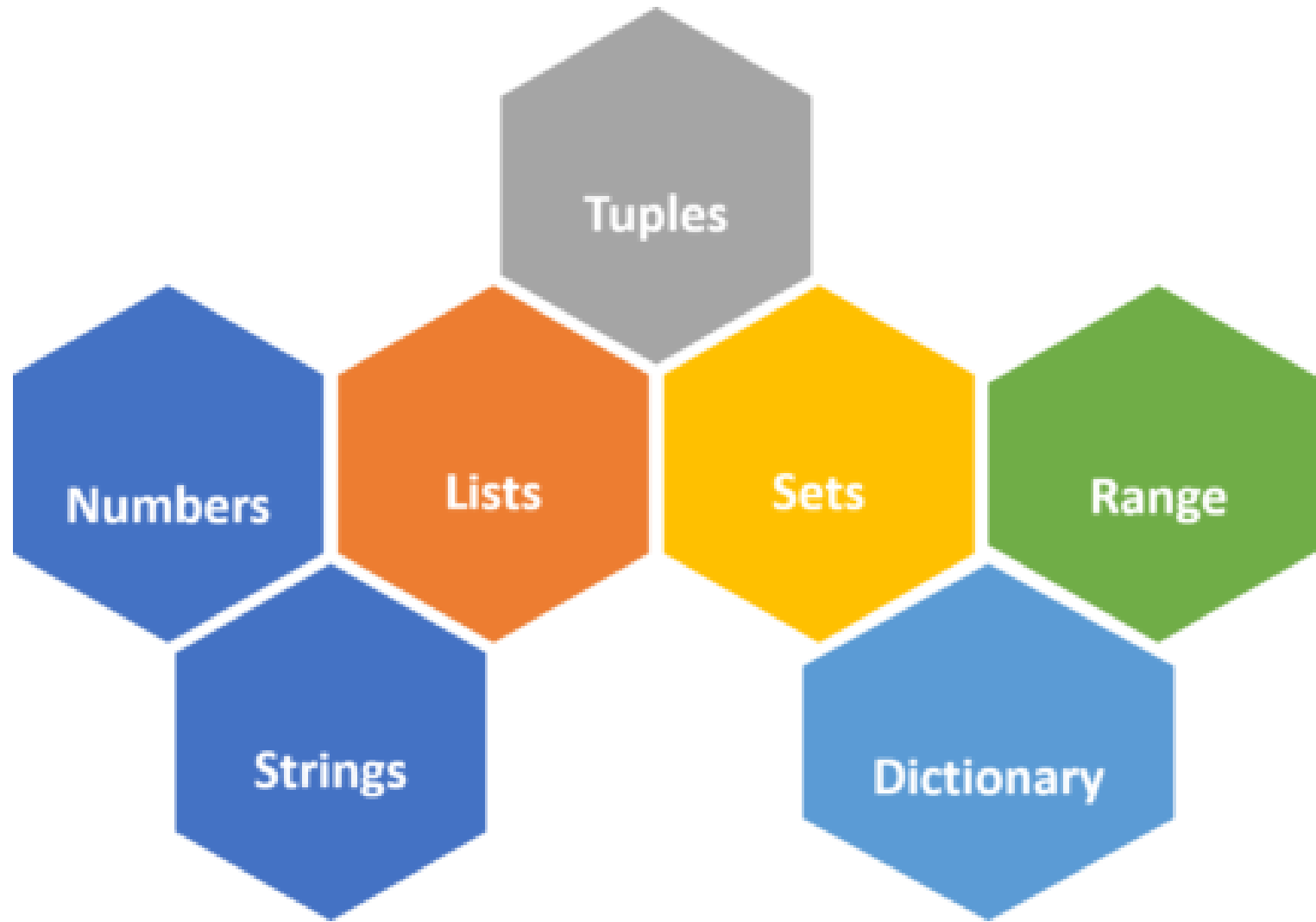
```
name="Amar"  
age=21  
print("Name:", name, end=" ")  
print("Age:", age)
```

Name: Amar Age: 21



Data Types

LECTURE 1



Basic Datatypes

- Integers (default for numbers)

- `z = 5 / 2` # Answer 2, integer division

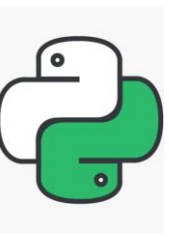
- Floats

- `x = 3.456`

- Strings

- Can use `""` or `' '` to specify with `"abc" == 'abc'`
 - Unmatched can occur within the string: `"matt's"`
 - Use triple double-quotes for multi-line strings or strings that contain both `'` and `"` inside of them:

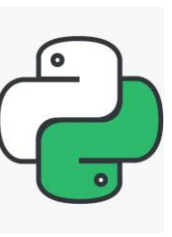
- `"""a'b'c"""`



Numeric

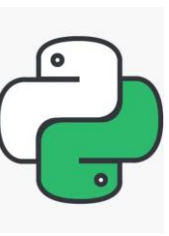
A numeric value is any representation of data which has a numeric value. Python identifies three types of numbers:

- **Integer:** Positive or negative whole numbers (without a fractional part)
- **Float:** Any real number with a floating point representation in which a fractional component is denoted by a decimal symbol or scientific notation
- **Complex number:** A number with a real and imaginary component represented as $x+yj$. x and y are floats and j is $\sqrt{-1}$



Boolean

- Data with one of two built-in values **True** or **False**. Notice that 'T' and 'F' are capital. true and false are not valid booleans and Python will throw an error for them.



Sequence Type

A sequence is an ordered collection of similar or different data types. Python has the following built-in sequence data types:

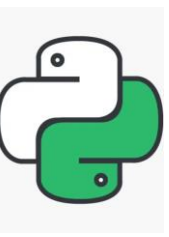
- **String**: A string value is a collection of one or more characters put in single, double or triple quotes.
- **List** : A list object is an ordered collection of one or more data items, not necessarily of the same type, put in square brackets.
- **Tuple**: A Tuple object is an ordered collection of one or more data items, not necessarily of the same type, put in parentheses.

```
>>> type(1234)
<class 'int'>
>>> type(55.50)
<class 'float'>
>>> type(6+4j)
<class 'complex'>
>>> type("hello")
<class 'str'>
>>> type([1,2,3,4])
<class 'list'>
>>> type((1,2,3,4))
<class 'tuple'>
>>> type({1:"one", 2:"two", 3:"three"})
<class 'dict'>
```



Variables

LECTURE 1



Constants

- Fixed values such as numbers, letters, and strings, are called “**constants**” because their value does not change
- Numeric constants are as you expect
- String constants use single quotes (')
or double quotes (")

```
>>> print(123)
```

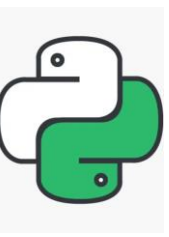
```
123
```

```
>>> print(98.6)
```

```
98.6
```

```
>>> print('Hello world')
```

```
Hello world
```



Variables

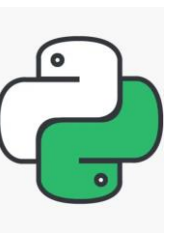
- A variable is a named place in the memory where a programmer can store data and later retrieve the data using the variable “name”
- Programmers get to choose the names of the variables
- You can change the contents of a variable in a later statement

x = 12.2

y = 14

x 12.2

y 14



Variables

- A variable is a named place in the memory where a programmer can store data and later retrieve the data using the variable “name”
- Programmers get to choose the names of the variables
- You can change the contents of a variable in a later statement

```
x = 12.2
```

```
y = 14
```

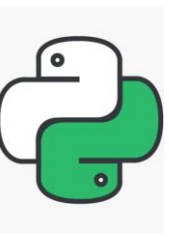
```
x = 100
```

x

~~12.2~~ 100

y

14



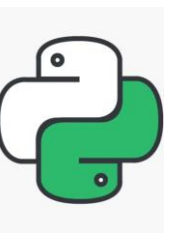
Python Variable Name Rules

- Must start with a letter or underscore _
- Must consist of letters, numbers, and underscores
- Case Sensitive

Good: spam eggs spam23 _speed

Bad: 23spam #sign var.12

Different: spam Spam SPAM



Mnemonic Variable Names

- Since we programmers are given a choice in how we choose our variable names, there is a bit of “best practice”
- We name variables to help us remember what we intend to store in them (“mnemonic” = “memory aid”)
- This can confuse beginning students because well-named variables often “sound” so good that they must be keywords

What is this bit of code doing?

```
x1q3z9ocd = 35.0  
x1q3z9afd = 12.50  
x1q3p9afd = x1q3z9ocd * x1q3z9afd  
print(x1q3p9afd)
```

What are these bits of code doing?

```
x1q3z9ocd = 35.0  
x1q3z9afd = 12.50  
x1q3p9afd = x1q3z9ocd * x1q3z9afd  
print(x1q3p9afd)
```

```
a = 35.0  
b = 12.50  
c = a * b  
print(c)
```

What are these bits of code doing?

```
x1q3z9ocd = 35.0  
x1q3z9afd = 12.50  
x1q3p9afd = x1q3z9ocd * x1q3z9afd  
print(x1q3p9afd)
```

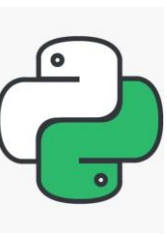
```
a = 35.0  
b = 12.50  
c = a * b  
print(c)
```

```
hours = 35.0  
rate = 12.50  
pay = hours * rate  
print(pay)
```



Expression

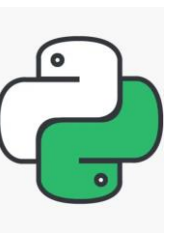
LECTURE 1



Numeric Expressions

- Because of the lack of mathematical symbols on computer keyboards - we use “computer-speak” to express the classic math operations
- Asterisk is multiplication
- Exponentiation (raise to a power) looks different than in math

Operator	Operation
+	Addition
-	Subtraction
*	Multiplication
/	Division
//	Integer Division
**	Power
%	Remainder



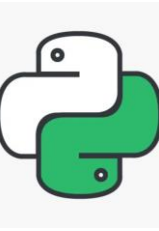
Numeric Expressions

```
>>> xx = 2
>>> xx = xx + 2
>>> print(xx)
4
>>> yy = 440 * 12
>>> print(yy)
5280
>>> zz = yy / 1000
>>> print(zz)
5.28
```

```
>>> jj = 23
>>> kk = jj % 5
>>> print(kk)
3
>>> print(4 ** 3)
64
```

```
      4 R 3
5 | 23
  | 20
  |---
  | 3
```

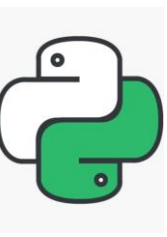
Operator	Operation
+	Addition
-	Subtraction
*	Multiplication
/	Division
//	Integer Division
**	Power
%	Remainder



Order of Evaluation

- When we string operators together - Python must know which one to do first
- This is called “operator precedence”
- Which operator “takes precedence” over the others?

`x = 1 + 2 * 3 - 4 / 5 ** 6`



Operator Precedence Rules

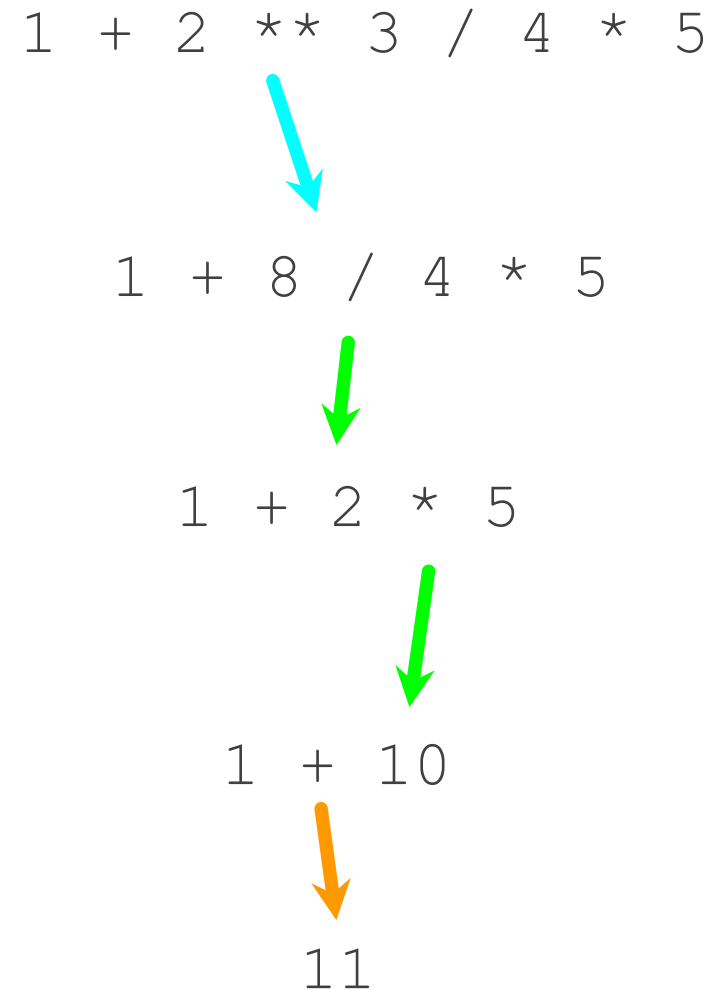
Highest precedence rule to lowest precedence rule:

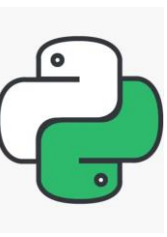
- Parentheses are always respected
- Exponentiation (raise to a power)
- Multiplication, Division, and Remainder
- Addition and Subtraction
- Left to right

Parenthesis
Power
Multiplication
Addition
Left to Right

```
>>> x = 1 + 2 ** 3 / 4 * 5
>>> print(x)
11.0
>>>
```

Parenthesis
Power
Multiplication
Addition
Left to Right

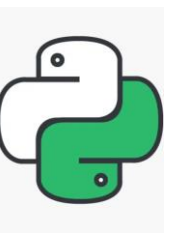




Operator Precedence

- Remember the rules top to bottom
- When writing code - use parentheses
- When writing code - keep mathematical expressions simple enough that they are easy to understand
- Break long series of mathematical operations up to make them more clear

Parenthesis
Power
Multiplication
Addition
Left to Right

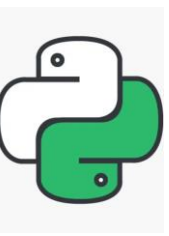


What Does “Type” Mean?

- In Python variables, literals, and constants have a “type”
- Python knows the difference between an integer number and a string
- For example “+” means “addition” if something is a number and “concatenate” if something is a string

```
>>> ddd = 1 + 4
>>> print(ddd)
5
>>> eee = 'hello ' + 'there'
>>> print(eee)
hello there
```

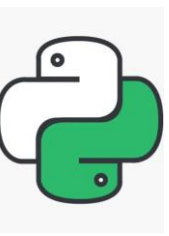
concatenate = put together



Type Matters

- Python knows what “type” everything is
- Some operations are prohibited
- You cannot “add 1” to a string
- We can ask Python what type something is by using the `type()` function

```
>>> eee = 'hello ' + 'there'
>>> eee = eee + 1
Traceback (most recent call last):
File "<stdin>", line 1, in
<module>TypeError: Can't convert
'int' object to str implicitly
>>> type(eee)
<class'str'>
>>> type('hello')
<class'str'>
>>> type(1)
<class'int'>
>>>
```



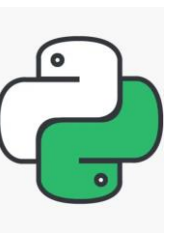
Several Types of Numbers

Numbers have two main types

- Integers are whole numbers:
-14, -2, 0, 1, 100, 401233
- Floating Point Numbers have
decimal parts: -2.5 , 0.0, 98.6, 14.0

There are other number types - they are variations on float and integer

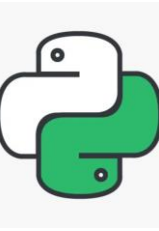
```
>>> xx = 1
>>> type (xx)
<class 'int'>
>>> temp = 98.6
>>> type(temp)
<class'float'>
>>> type(1)
<class 'int'>
>>> type(1.0)
<class'float'>
>>>
```



Type Conversions

- When you put an integer and floating point in an expression, the integer is implicitly converted to a float
- You can control this with the built-in functions `int()` and `float()`

```
>>> print(float(99) + 100)
199.0
>>> i = 42
>>> type(i)
<class 'int'>
>>> f = float(i)
>>> print(f)
42.0
>>> type(f)
<class 'float'>
>>>
```

Integer Division

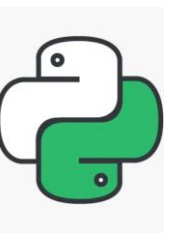
- Division produces a floating point result.
- Integer Division produces integer results

```
>>> print(10 / 2)
5.0
>>> print(9 / 2)
4.5
>>> print(99 / 100)
0.99
>>> print(10.0 / 2.0)
5.0
>>> print(99.0 / 100.0)
0.99
```



Statements

LECTURE 1



Sentences or Lines

```
x = 2
```

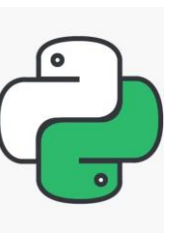
```
x = x + 2
```

```
print(x)
```

Assignment statement

Assignment with expression

Print statement

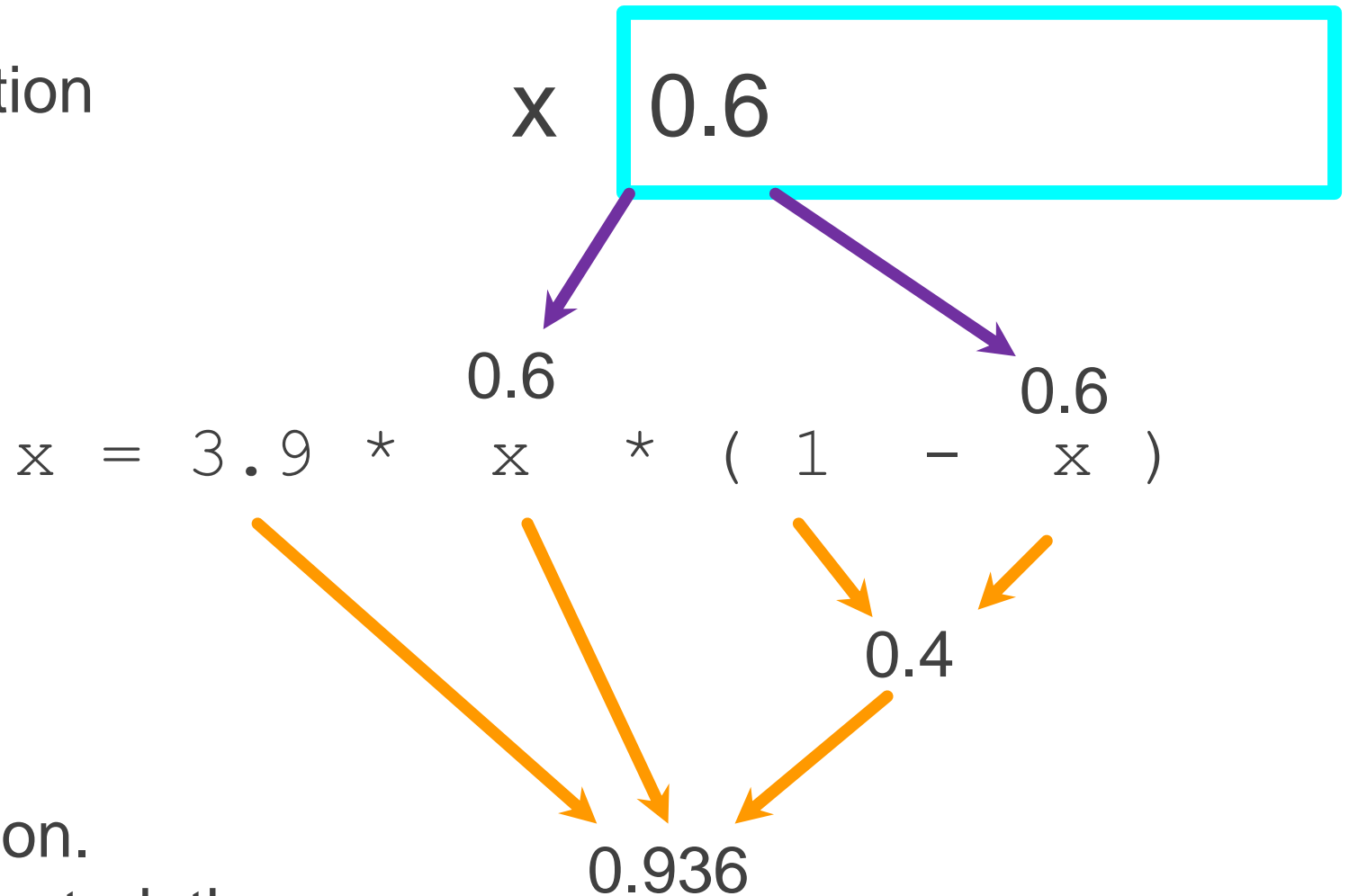


Assignment Statements

- We assign a value to a variable using the assignment statement (=)
- An assignment statement consists of an expression on the right-hand side and a variable to store the result

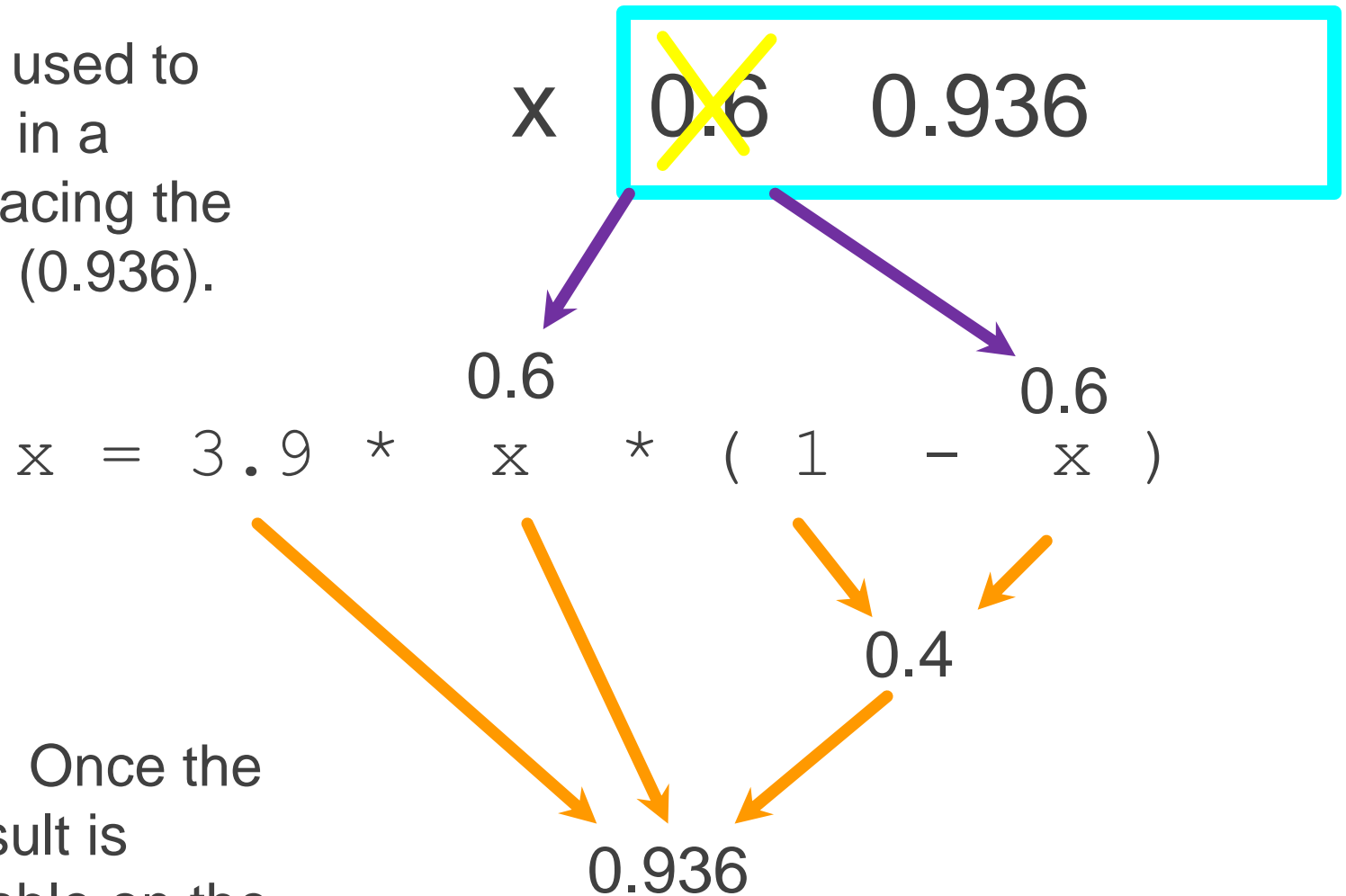
```
x = 3.9 * x * ( 1 - x )
```

A variable is a memory location used to store a value (0.6)



The right side is an expression.
Once the expression is evaluated, the
result is placed in (assigned to) `x`.

A variable is a memory location used to store a value. The value stored in a variable can be updated by replacing the old value (0.6) with a new value (0.936).

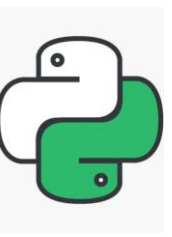


The right side is an expression. Once the expression is evaluated, the result is placed in (assigned to) the variable on the left side (i.e., x).



Type Conversion

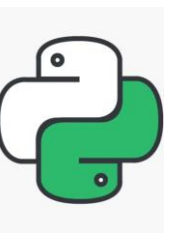
LECTURE 1



String Conversions

- You can also use `int()` and `float()` to convert between strings and integers
- You will get an error if the string does not contain numeric characters


```
>>> sval = '123'
>>> type(sval)
<class 'str'>
>>> print(sval + 1)
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
TypeError: Can't convert 'int' object
to str implicitly
>>> ival = int(sval)
>>> type(ival)
<class 'int'>
>>> print(ival + 1)
124
>>> nsv = 'hello bob'
>>> niv = int(nsv)
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
ValueError: invalid literal for int()
with base 10: 'x'
```



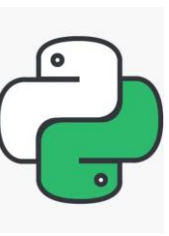
User Input

We can instruct Python to pause and read data from the user using the `input()` function

The `input()` function returns a string

```
nam = input('Who are you? ')\nprint('Welcome', nam)
```

```
Who are you? Chuck\nWelcome Chuck
```



Converting User Input

- If we want to read a number from the user, we must convert it from a string to a number using a type conversion function
- Later we will deal with bad input data

```
inp = input('Europe floor?')  
usf = int(inp) + 1  
print('US floor', usf)
```

Europe floor? 0

US floor 1

