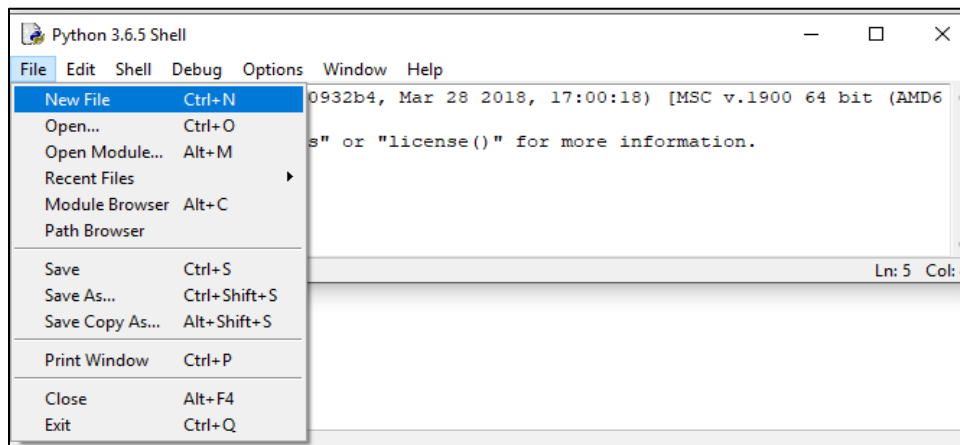
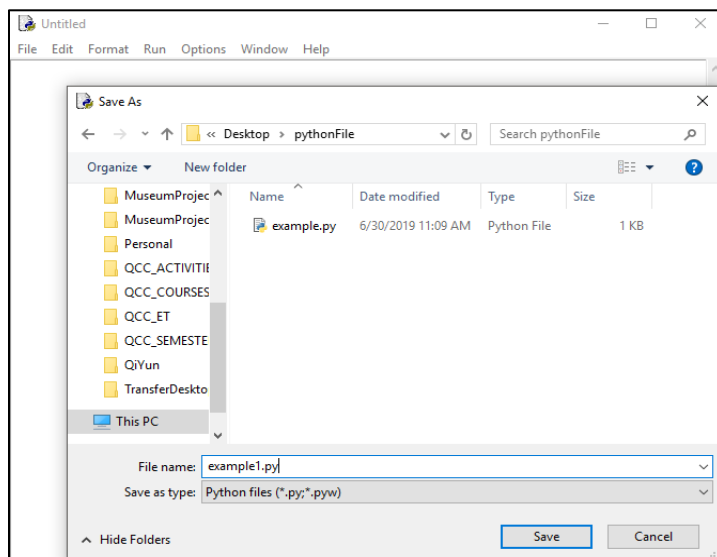


### **Saving a python program**

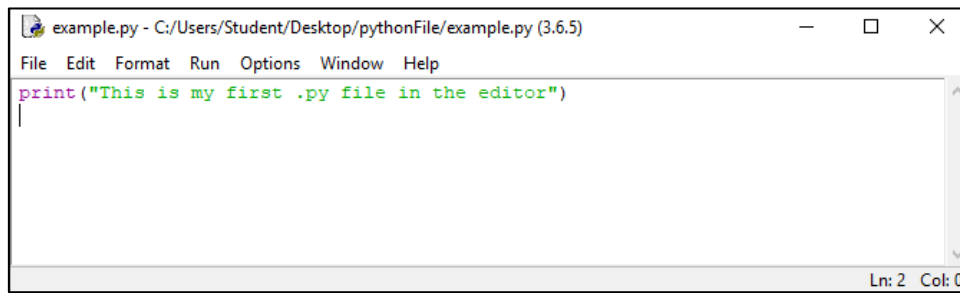
Python file is saved with the file extension ***py***. To create a python file, select a *New File* from the File menu:



Once the python file appears, we can save it first in a local folder and give a file name. During the saving process, make sure that the *Save As type* is selected as: ***Python file (.py)***

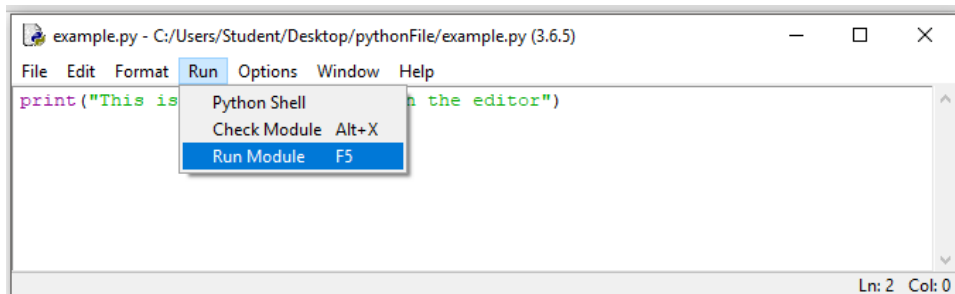


In the python file, we can write the code within the file:

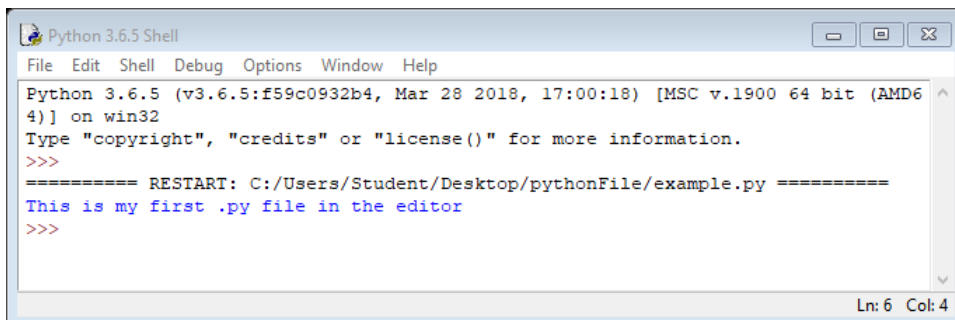


```
example.py - C:/Users/Student/Desktop/pythonFile/example.py (3.6.5)
File Edit Format Run Options Window Help
print("This is my first .py file in the editor")
Ln: 2 Col: 0
```

To run the file, we save the file, and click on the Run tab and select Run Module, or use the F5 function key from the keyboard.



```
example.py - C:/Users/Student/Desktop/pythonFile/example.py (3.6.5)
File Edit Format Run Options Window Help
print("This is my first .py file in the editor")
Ln: 2 Col: 0
```



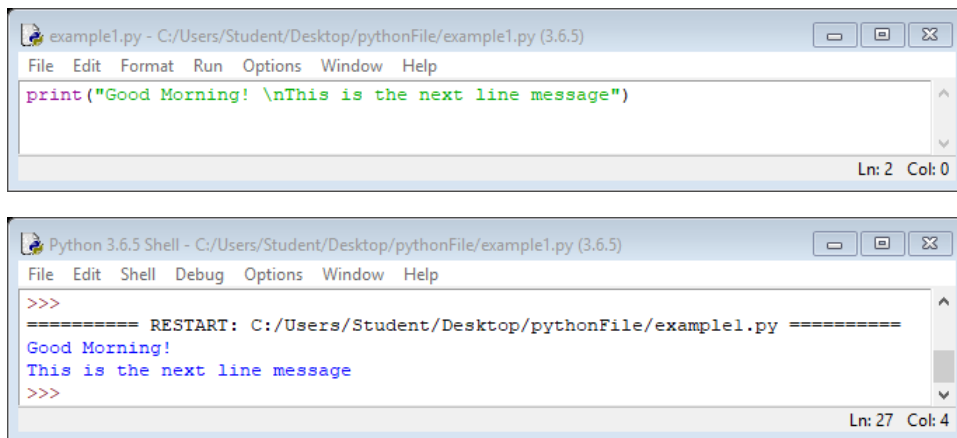
```
Python 3.6.5 Shell
File Edit Shell Debug Options Window Help
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 17:00:18) [MSC v.1900 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:/Users/Student/Desktop/pythonFile/example.py =====
This is my first .py file in the editor
>>>
Ln: 6 Col: 4
```

## In Python strings, the backslash "\"

**The backslash "\"** is a **special character**, also called the **"escape"** character. It is used in representing certain whitespace characters:

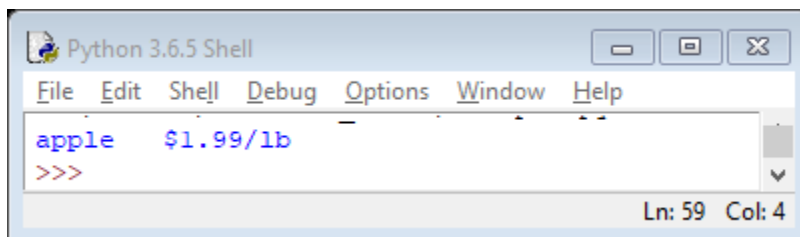
- **\t** is a tab
- **\n** is a newline

For example, if we want to make a new line in a message, we can use the command **\n**



You can also add a tab to a line as:

```
print('apple\t$1.99/lb');
```



Conversely, prefixing a special character with `\` turns it into an ordinary character. This is called "escaping". For example, `\` is the single quote character. `'It\'s raining'` therefore is a valid string and equivalent to `It's raining`.

```
print ("It\'s raining")
```

Prompt result:

```
It's raining
>>>
```

Likewise, `'"` can be escaped: `"\"hello\""` is a string begins and ends with the literal double quote character.

```
print ("\"hello\"")
```

Prompt result:

```
"hello"  
>>> |
```

Finally, `"\"` can be used to escape itself: `"\"` is the literal backslash character.

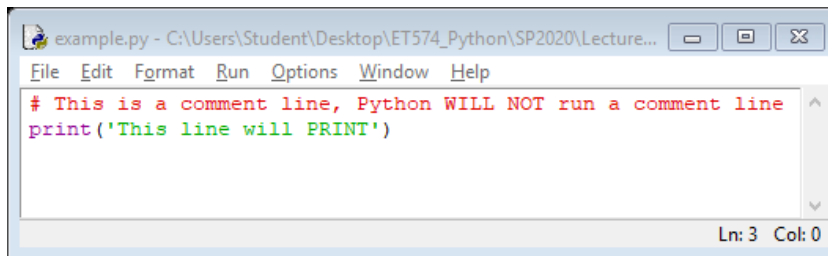
```
print ("\" is the backslash')
```

Prompt result:

```
"\" is the backslash  
>>> |
```

## Comments

**Comments in Python** start with the hash character, `#`, and extend to the end of the physical line. A **comment** may appear at the start of a line or following whitespace or code, but not within a string literal. A hash character within a string literal is just a hash character.



Prompt result

```
This line will PRINT  
>>> |
```

You can add a multiple comment lines by using triple-quotes from the beginning to the end of the comment.

```
"""  
This is a comment  
written in  
more than just one line  
"""  
print("Hello, World!")
```

Prompt result

```
Hello, World!  
>>> |
```

## Values and types

A value is one of the fundamental things—like a letter or a number—that a program manipulates. The values we have seen so far are 2(the result when we added 1 + 1), and 'Hello, World!'

These values belong to different types: 2 is an integer, and 'Hello, World!' is a string, so-called because it contains a “string” of letters. You (and the interpreter) can identify strings because they are enclosed in quotation marks.

The print statement also works for integers.

If you are not sure what type a value has, the interpreter can tell you.

Not surprisingly, strings belong to the type **str** and integers belong to the type **int**. Less obviously, numbers with a decimal point belong to a type called **float**, because these numbers are represented in a format called floating-point.

```
>>> type(3.56)
<class 'float'>
>>> type(160)
<class 'int'>
>>> type("Hello World!")
<class 'str'>
```

What about values like '165.89'? It look like an integer, but it is in between quotation marks like strings. Therefore 165.89 is a string.

```
>>> type('165.89')
<class 'str'>
```

You can also print the value type as:

```
print(type(1.5))
```

When you type a large integer, you might be tempted to use commas between groups of three digits, as in 1,000,000. This is not a legal integer in Python, but it is a legal expression:

```
>>> print(1,000,000)
1 0 0
```

Well, that’s not what we expected at all! Python interprets 1,000,000 as a comma-separated list of three integers, which it prints consecutively. This is the first example we have seen of a semantic error: the code runs without producing an error message, but it doesn’t do the “right” thing.

## Calculations and Variables

Now that you have Python installed and familiar with the IDLE Python shell, we are ready to do something with Python. We will start with simple mathematical operations and then move on to variables.

## Variables

Variable in programming describes a place to store information such as numbers, text, lists of numbers and text, and so on. Those information can be retrieved later for use by calling the variable's name.

For example, to create a variable named *number*, we use an equal sign ( = ) and then the type of information the variable should be the label for. If we want to assign number to 120 to variable *number*, we can write the code as: `number = 120`

### **Variable names**

Variable names can be made up of letters, numbers, and the underscore character `_`, but they:

- CAN'T start with a number
- CAN'T have symbols, except underscore character
- CAN'T have space.
- CAN'T use Python keywords

It turns out that `class` is one of the Python keywords. Keywords define the language's rules and structure, and they cannot be used as variable names.

Python has twenty-nine keywords:

and	def	exec	if	not	return
assert	del	finally	import	or	try
break	elif	for	in	pass	while
class	else	from	is	print	yield
continue	except	global	lambda	raise	

You might want to keep this list handy. If the interpreter displays an error about one of your variable names and you don't know why, see if it is on this list.

We can name our variable as we wish, but my recommendation is to name the variables according to its value or use.

### **Prompting a variable value**

To prompt a variable value, we can just type the variable name:

```
x = 90.5
print (x)
```

*Prompt result*

```
90.5
>>> |
```

### *Using variables*

Variables can also use amount them. For example, if we want to add to variables *number1* and *number2*, we can write the following code:

```
number1 = 20
number2 = 30
addition = number1 + number2
print ("The sum is: ", addition)
```

Also remember that variables are case sensible, for example, from the previous example, if we named the variable as *number1* and then we call the variable as *Number1*, we will have an error when we try to run the Python file, because *number1* and *Number1* is not the same variable.

### Assign value to multiple variables

- Python allows us to assign values to multiple variables in one line. Each variable and values must be separated by a comma:

```
item1, item2, item3 = "red", 28, 100.95;
print(item1, item3);
```

#### *Prompt result*

```
red 100.95
>>>
```

- You can assign the same value to multiple variables in one line:

```
number1=number2=number3= 12.5
print("variable number 1 is: ", number1)
print("variable number 2 is: ", number2)
print("variable number 3 is: ", number3)
```

#### *Prompt result*

```
variable number 1 is:  12.5
variable number 2 is:  12.5
variable number 3 is:  12.5
>>> |
```

### *input( ) function*

Python user **input** from the keyboard can be read using the **input( )** built-in function.

The **input( )** from the user is read as a string and can be assigned to a variable. After entering the value from the keyboard, we have to press the “Enter” button. Then the **input( )** function reads the value entered by the user.

*Syntax:*

```
variable_name = input('Display message: ')
```

For example, if we want to collect a first name from the computer keyword and save it in variable **msa**:

```
msa=input('Enter a first name: ')
```

Once we have the information saved in variable **msa**, we can then prompt the value in a message as:

```
print('The entered first name is: ',msa)
```

Running the code, the command window will prompt the input message first:

```
Enter a first name:
```

If we type **Peter** and hit **Enter**, Python will move to the next code line and print the value of variable **msa**, which the name that the user typed and entered:

```
Enter a first name: Peter
The entered first name is: Peter
>>> |
```

### Python Casting

In computer programming, casting is a source code of a program that tells the compiler to generate the machine code that perform as the actual conversion. In other words, casting means to convert from one data type into another. For example, when we collect a number from the keyword, the value from the keyword is actually a string and not an integer or float type. Therefore, if we perform operations with the number collected from the keyword, it will perform a string operation and not a numerical operation. Let's take a look at the following code:

```
number1 = input("Enter a number: ")
product = number1 * 2
print("The entered number times 2 is: ", product)
```

*Prompt result*

```
Enter a number: 25
The entered number times 2 is:  2525
>>> |
```

If we want to perform a numerical operation, then we need to cast the input type string to integer. To do so, we can use the Python casting **int()** right after the input function:

```
number1 = input("Enter a number: ")
→ number1 = int(number1)
product = number1 * 2
print("The entered number times 2 is: ", product)
```



### *Prompt result*

```
Enter a number: 25
The entered number times 2 is: 50
>>> |
```

or we can use Python casting within the **input** function

→ 

```
number1 = int(input("Enter a number: "))
product = number1 * 2
print("The entered number times 2 is: ", product)
```

### *Python Arithmetic Operators*

In Python, we can do multiplication, addition, subtraction, division, return the remainder of a two numbers, exponentiation, integer division, and parentheses using the basic operators:

Python Arithmetic Operators	
Symbol	Operations
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus (remainder)
//	Integer division
**	Exponentiation
()	Parentheses

### *Order of operation*

Like a calculator, any programming language perform operations due to their order of operations. An operation is anything that uses an operator. For example, multiplication and division have a higher order than addition and subtraction. It basically means that if we have an equation that has multiplication and subtraction, the program will calculate the multiplication first and then the subtraction.

**Example)** if we solve for the following operation:  $5 + 30 * 20$ , the operation will solve the multiplication first:  $5 + 600$ , after the product, it will solve the addition giving 605.

Use of parentheses in a programming language to control the flow of the operations. Order of operation will always solve the equation inside the parentheses before proceeding with the basic operators. If we take  $5 + 30 * 20$  and add parentheses to it:  $(5 + 30) * 20$ , the operation will solve the operation inside the parentheses first,  $35 * 20$ , after the sum, it will solve the multiplication giving 700

Parentheses can be nested, which means that there can be parentheses inside parentheses:

$((5 + 30) * 20) / 10$

In this case, the order of operation evaluates the innermost parentheses first:  $(35 * 20) / 10$ , then the outer parentheses:  $700 / 10$ , and then the final division operator giving: 70.

Example) perform the following operations:

```
x1 = (6-7)*8
x2 = (3+2)**(6-4)
x3 = 3**2*-1
x4 = 5%3
x5 = 7/2
x6 = 7//2
```

```
print("x1: ", x1)
print("x2: ", x2)
print("x3: ", x3)
print("x4: ", x4)
print("x5: ", x5)
print("x6: ", x6)
```

*Prompt result*

```
x1:  -8
x2:  25
x3:  -9
x4:  2
x5:  3.5
x6:  3
>>>
```

Example) ask the user to enter two sides of a right triangle as a float number and calculate the hypotenuse as

**hyp =  $(h^2 + w^2)^{0.5}$**

Prompt the two sides and the hypotenuse:

```
h = float(input("Enter the height: "))
w = float(input("Enter the width: "))
hyp = (h**2+w**2)**0.5
print("A triangle with sides: ", h, ", ", w, " has a hypotenuse of ", hyp)
```

*Prompt result*

```
Enter the height: 6.2
Enter the weight: 1.2
A triangle with sides: 6.2 and 1.2 has a hypotenuse of 6.315061361538778
>>> |
```

### Python Assignment Operators

The Python assignment operators are used to assign values to the declared variables

Python Assignment Operators				
Symbol	Operations	Python Example	Arithmetic equality	Value
=	Assign a value	x = 5	x = 5	5
+=	Self-addition	x+=3	x = x+3	8
-=	Self-subtraction	x-=3	x = x-3	2
*=	Self-multiplication	x*=3	x = x*3	15
/=	Self-division	x/=3	x = x/3	1.6 $\bar{6}$
//=	Self-division, returning only the quotient	x//=3	x = x//3	1
%=	Returning remainder of a self-division	x%=3	x = x%3	2
**=	Self-exponential	x**=3	x = x**3	125