Python Mini Lessons

From http://www.onlineprogramminglessons.com

These Python mini lessons will teach you all the Python Programming statements you need to know, so you can write 90% of any Python Program.

Lesson 1   **Input and Output**

Lesson 2   **Functions**

Lesson 3   **Classes**

Lesson 4   **Operators, Lists, Tuples and Dictionaries**
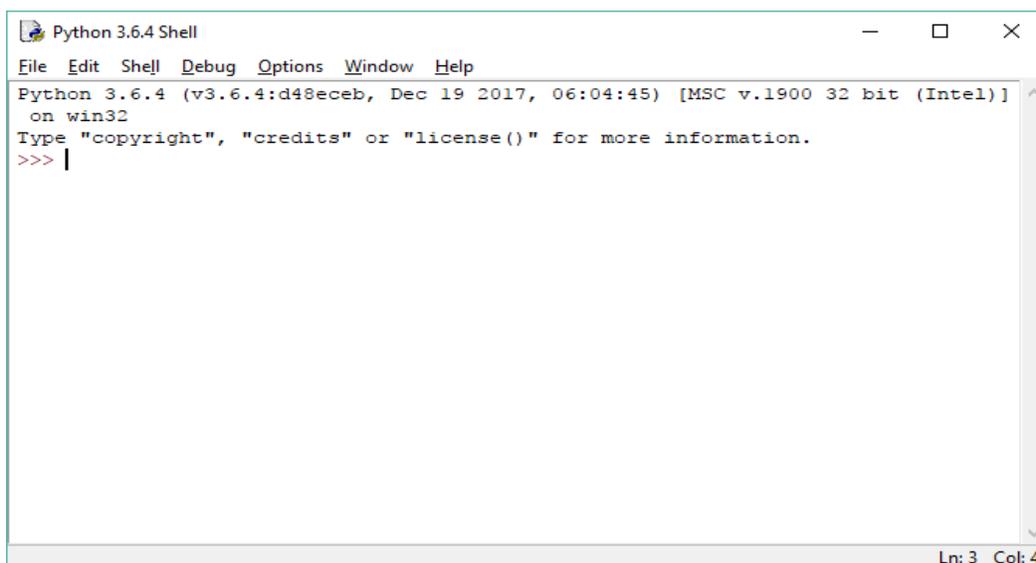
Lesson 5   **Programming Statements**

Let's get started!

You first need a Python interpreter to run Python Programs.

Download from this site: https://www.python.org/downloads/

Choose Python version 3.6.4

Once you download and run you will get this screen known as the Python interpreter Shell:
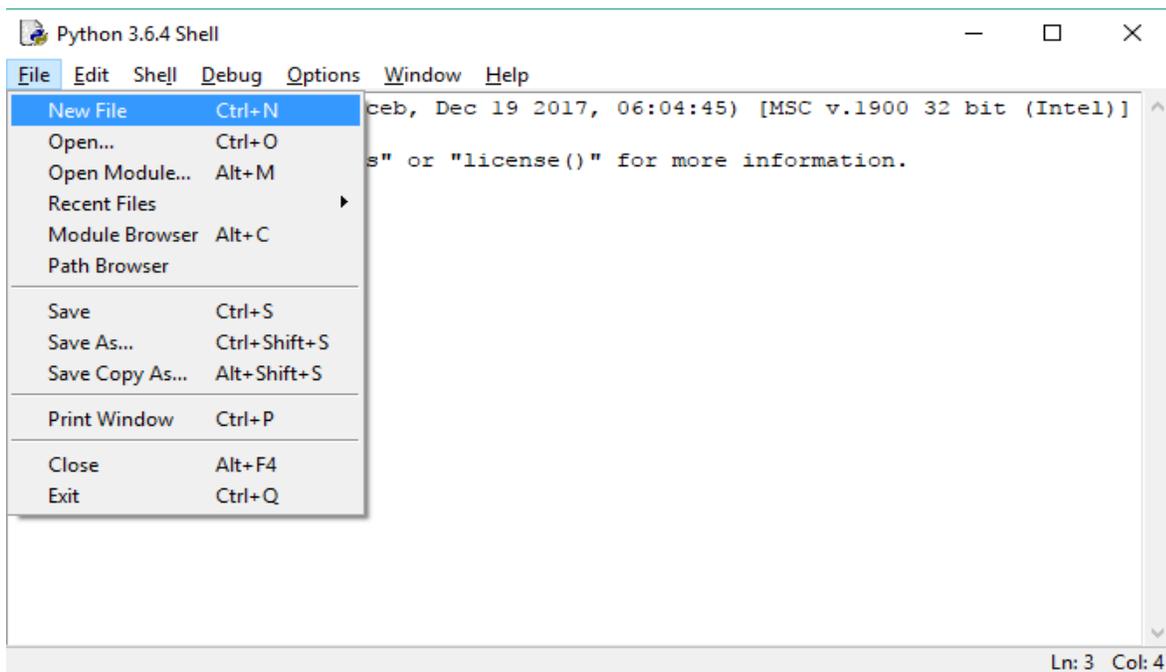


```
Python 3.6.4 Shell                                          —    □    ×
File  Edit  Shell  Debug  Options  Window  Help
Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017, 06:04:45) [MSC v.1900 32 bit (Intel)]
 on win32
Type "copyright", "credits" or "license()" for more information.
>>> |
                                                                Ln: 3  Col: 4
```
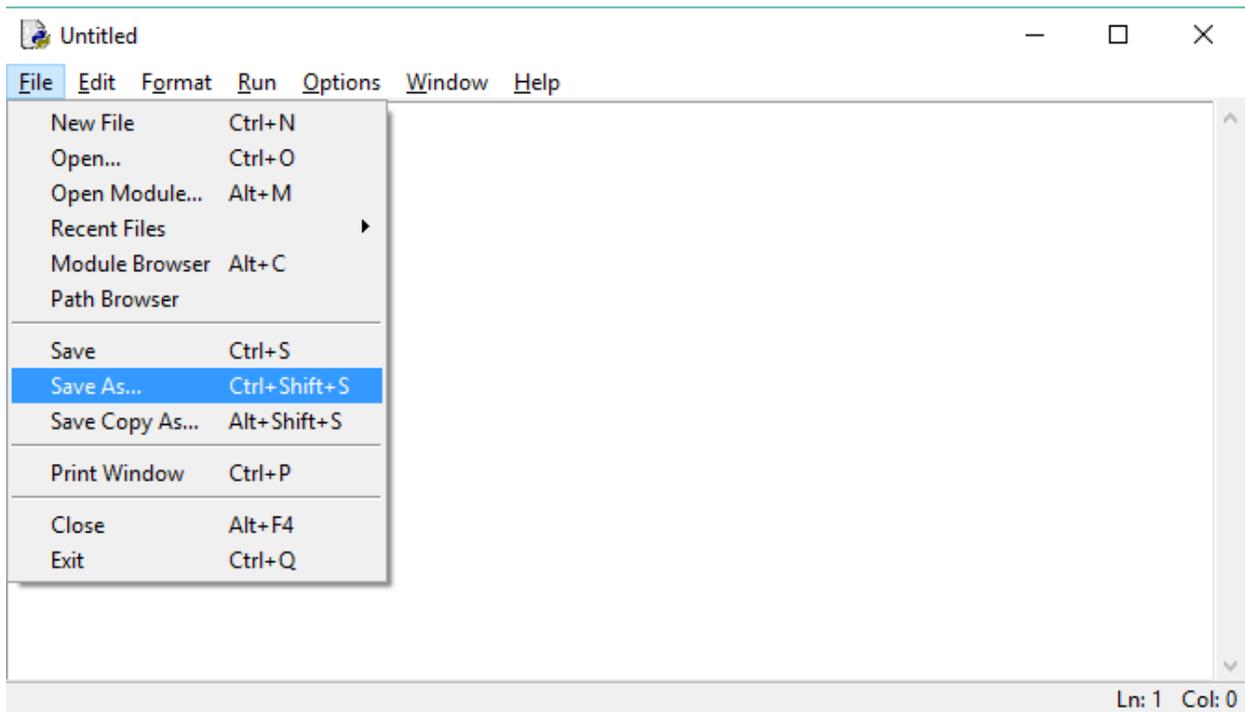
Lesson 1   Input and Output

It is best to store all your lesson programs in a file rather than using the Python interpreter shell. Python program are also known as scripts, because it is an interpretive language, meaning the python programming statements are executed one by one as they appear in the file. From the File Menu select New File.
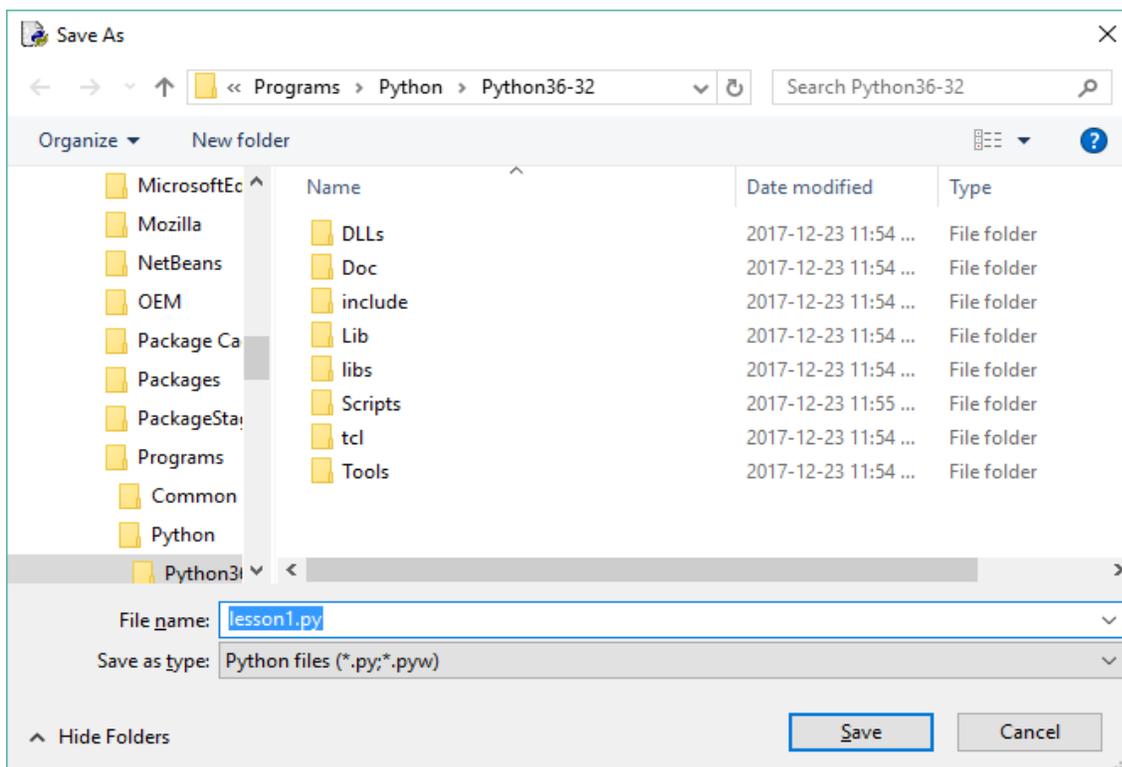

The editor window appears where you can type in Python programming statements that you can save and run.

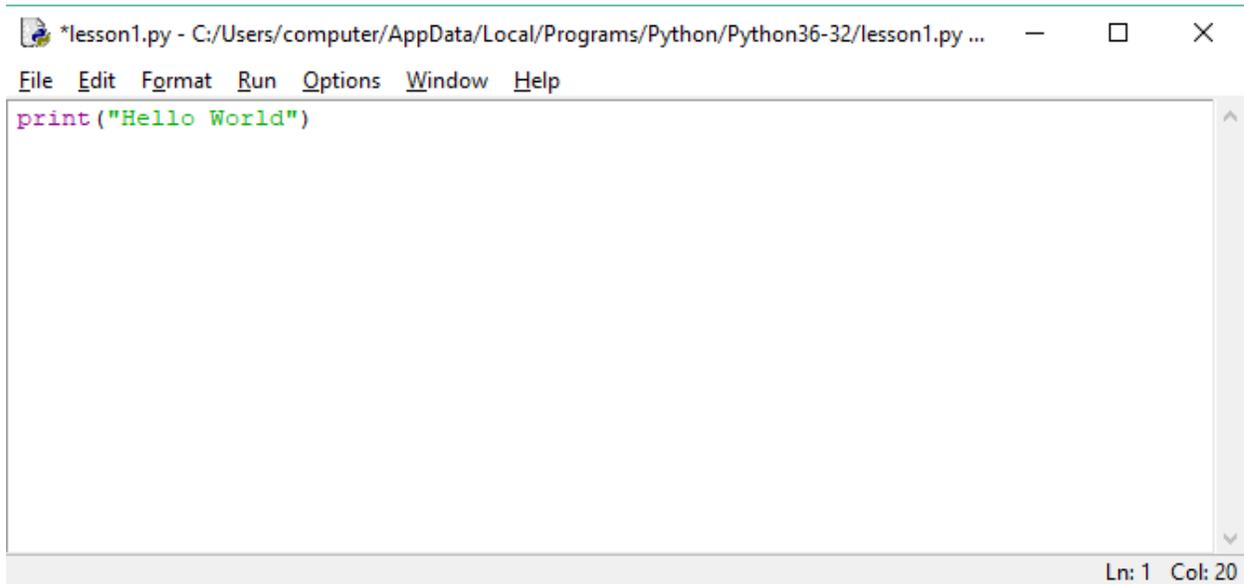From the File menu select File Save As

Save file as lesson1.py
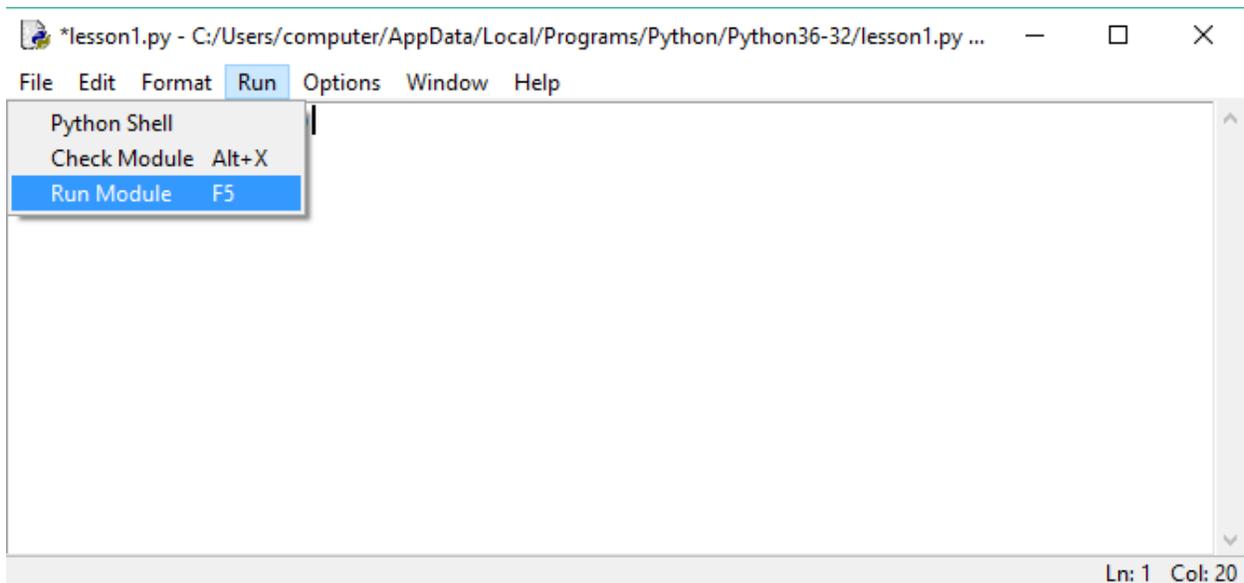
It's now time to write your first Python Programming Statement. You will print Hello World on the screen. In the Python Editor type:

print("Hello World")

To run your program select Run Module from the Run Menu.

Select OK

Hello World is now printed on the screen in another Python Shell. The **print** statement was used to print a message on the screen like Hello World

The next thing we need to do is get values from the keyboard. We will ask the user to type in their name and then greet them.

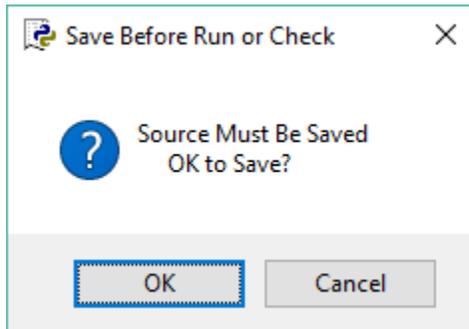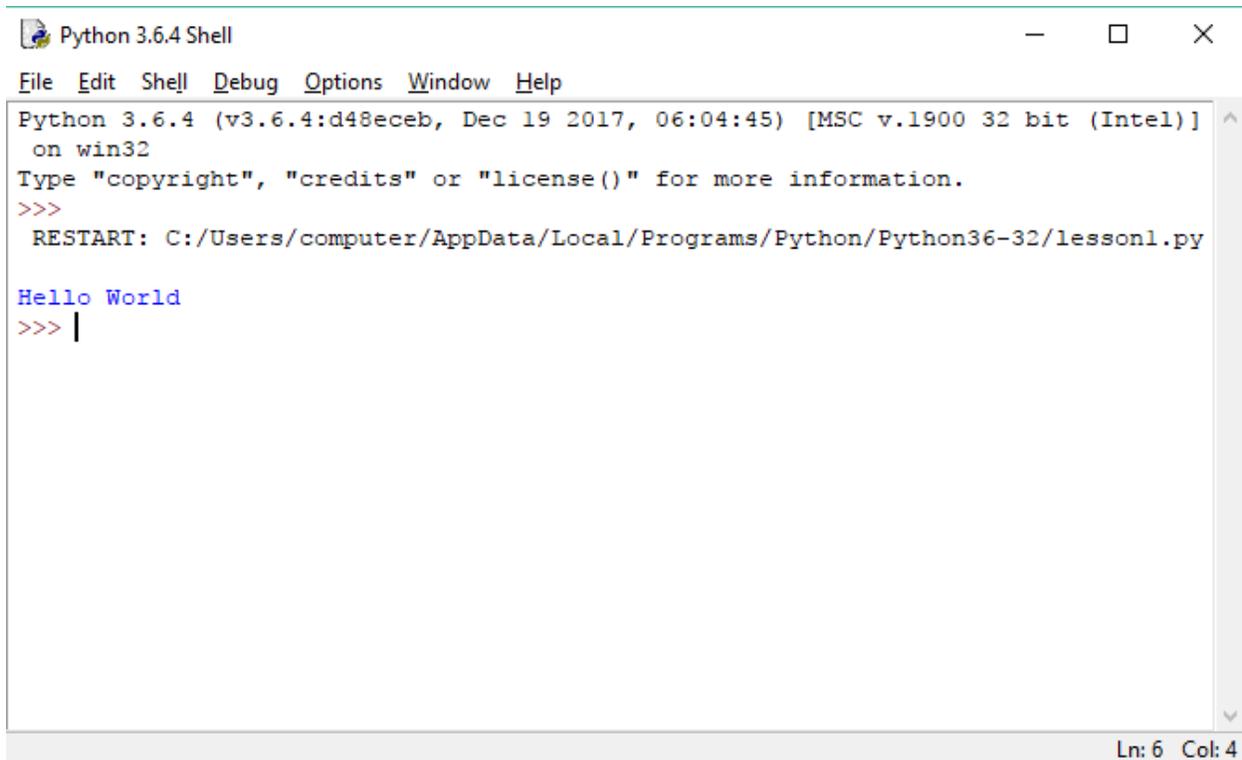Type in the following statements in the python editor right after the Hello World statement. The **input** statement is used to read values from the keyboard.

> **name = input("Please type in your name: ")**
>
> **print("Nice to meet you ", name)**
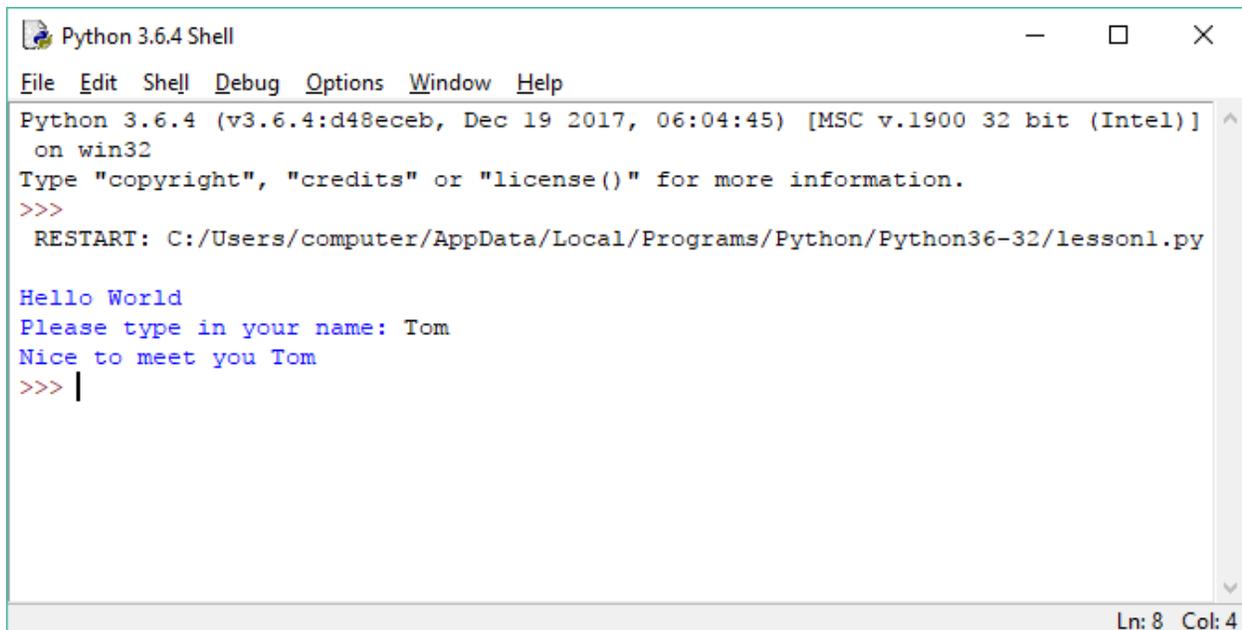
```
*lesson1.py - C:/Users/computer/AppData/Local/Programs/Python/Python36-32/lesson1.py ...    —    □    ✕

File  Edit  Format  Run  Options  Window  Help

print("Hello World")
name = input("Please type in your name: ")
print("Nice to meet you ", name)
|




                                                                      Ln: 4  Col: 0
```

Now run your program, you will get something like this:

(You may want to close the previous Python interpreter shell before running.)

```
Python 3.6.4 Shell                                                  —    □    ✕

File  Edit  Shell  Debug  Options  Window  Help

Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017, 06:04:45) [MSC v.1900 32 bit (Intel)]
 on win32
Type "copyright", "credits" or "license()" for more information.
>>>
 RESTART: C:/Users/computer/AppData/Local/Programs/Python/Python36-32/lesson1.py

Hello World
Please type in your name: Tom
Nice to meet you Tom
>>> |




                                                                      Ln: 8  Col: 4
```
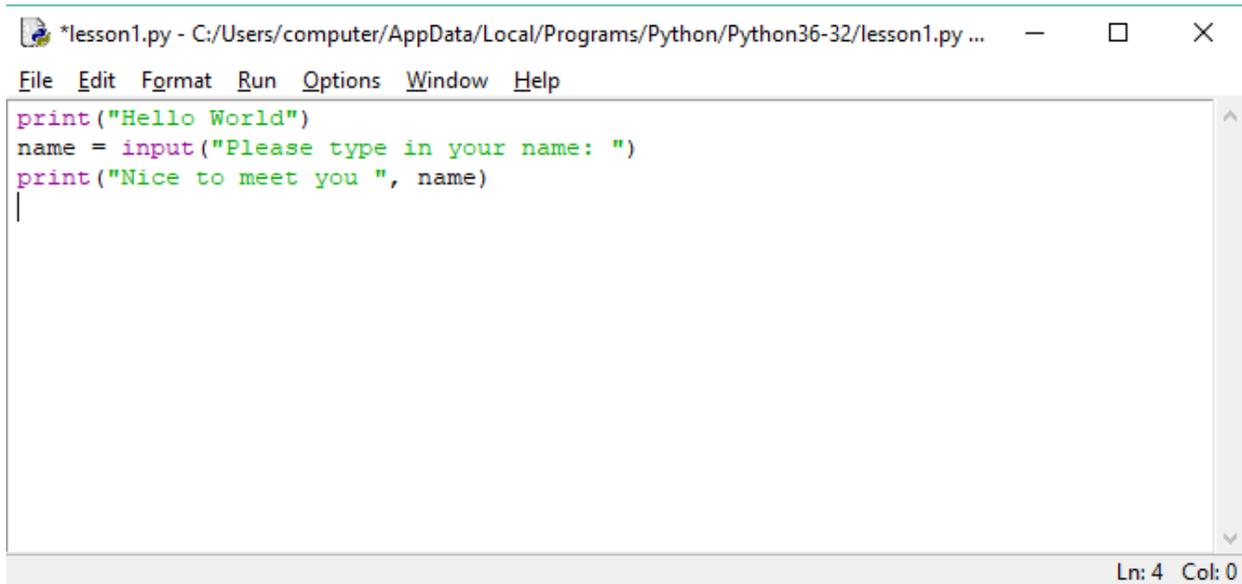
Recapping: The **print** statements writes messages on the screen the **input** statement reads values from the key board. Python has two types of values **string** values and **numeric** values. String values are messages enclosed in double or single quotes like "Hello World" or 'Hello World' where as numeric values are numbers like 5 and 10.5 Numeric values without decimal points like 5 are known as an **int** and numbers with decimal points like 10.5 are known as a **float**. Variable store string or numeric values that can be used later in your program. The variable **name** stores the string value entered from the keyboard, in this case the persons name.

        name = input("Please type in your name: ")

The print statement prints out the string message "Nice to meet you" and the name of the user stored in the variable name.

        print("Nice to meet you ", name)
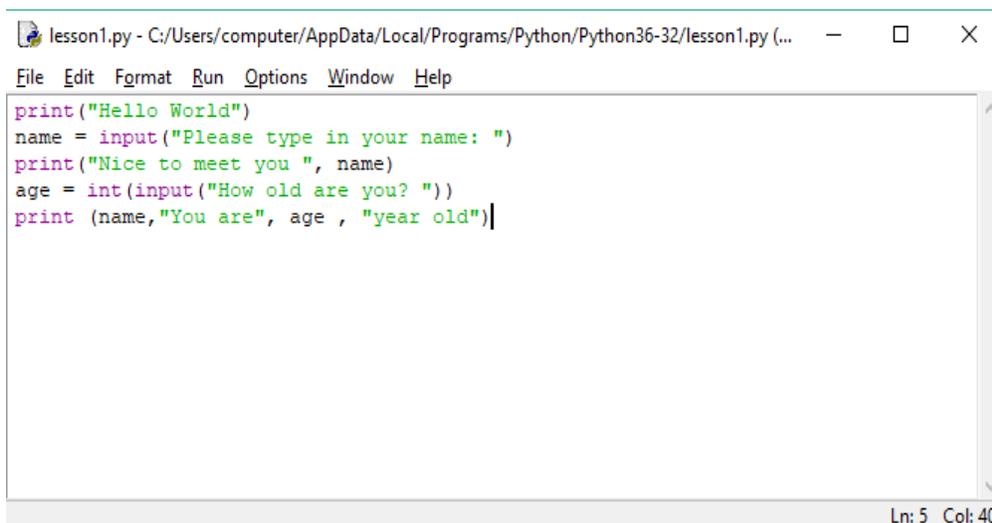
Note inside the print statement the string message and variable name are separated by commas.

We now ask the user how old they are.

Type in the following statements at the end of your program and then run the program.

        age = int(input("How old are you? "))
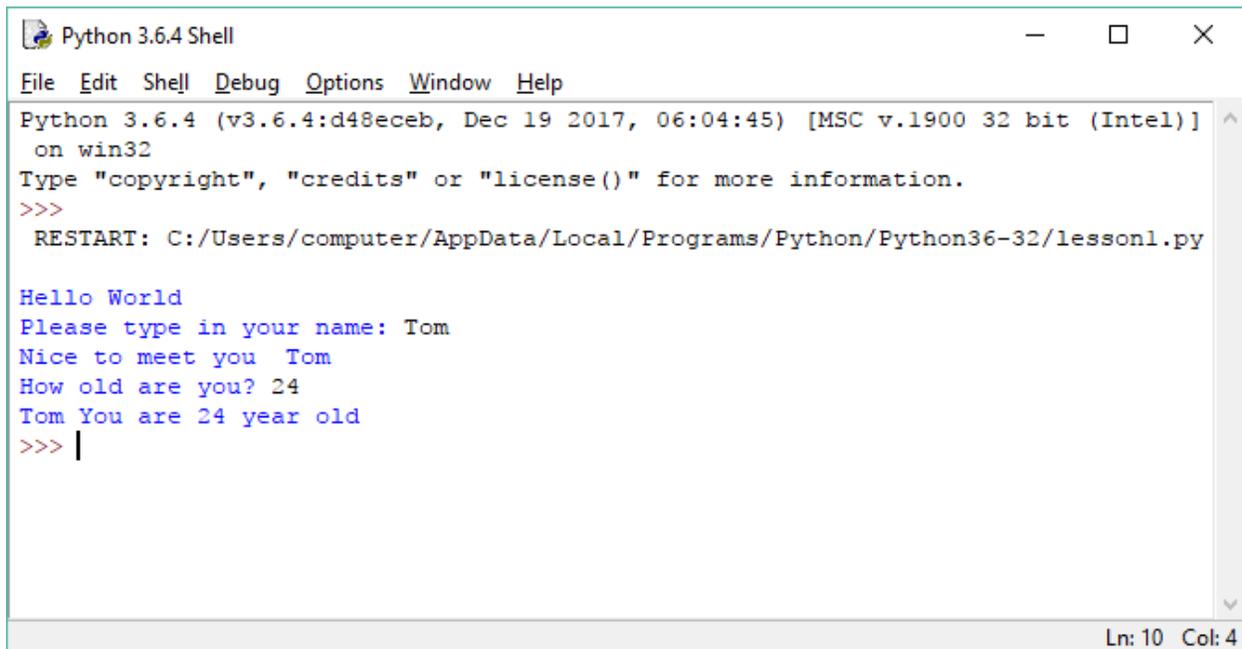
        print (name,"You are", age , "year old")

```
lesson1.py - C:/Users/computer/AppData/Local/Programs/Python/Python36-32/lesson1.py (...    —    □    ✕
File  Edit  Format  Run  Options  Window  Help
print("Hello World")
name = input("Please type in your name: ")
print("Nice to meet you ", name)
age = int(input("How old are you? "))
print (name,"You are", age , "year old")
                                                                                  Ln: 5  Col: 40
```

Run the program and enter Tom for name and 24 for age

```
Python 3.6.4 Shell                                         —    □    ✕

File  Edit  Shell  Debug  Options  Window  Help
Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017, 06:04:45) [MSC v.1900 32 bit (Intel)]  ^
 on win32
Type "copyright", "credits" or "license()" for more information.
>>>
 RESTART: C:/Users/computer/AppData/Local/Programs/Python/Python36-32/lesson1.py

Hello World
Please type in your name: Tom
Nice to meet you  Tom
How old are you? 24
Tom You are 24 year old
>>> |




                                                                       Ln: 10  Col: 4
```

Recapping: The **input** statement asks the used to enter the age. The **int** statement
converts the entered age to a **int** number and the variable age holds the age value.
We need to convert the string input to a numeric value using the **int** statement.

> **age = int (input ("How old are you? "))**

The print statement is used to print out the messages, the persons name and age.

> **print (name,"You are", age , "year old")**

If you have got this far then you will be a great python programmer soon.

Most people find Programing difficult to learn. The secret of learning program is to
figure out what you need to do and then choose the right program statement to
use. If you want to print messages and values to the screen you use a **print**
statement. If you want to get values from the user, you use an **input** statement. If
you need a numeric value, you use a **int** statement or **float** statement on the **input**
statement to convert the input value to a numeric value.

Lesson 2   Functions

Functions allow you to group many programming statements together so that you can reuse them repeatedly in your Python Program. The most common function to use is the main function. We will now group our previous programming statements in a main function and then call the main function from the python script.

Start a new python File from the Python shell called lesson2.py. Then type in the following python statements.

```python
def main():

    print("Hello World")

    name = input("Please type in your name: ")

    print("Nice to meet you ", name)

    age = int(input("How old are you? "))

    print (name,"You are", age , "year old")


main()
```

Now run your program, it will do the same thing as the previous program.

All functions in Python start with the key word **def**. Function name's end with 2 rounds brackets (). The round brackets distinguish a function name from a variable name. After the function name and the round brackets () there is a colon : the colon states there are programming statements to follow that belong to the current function name. Function statements are indented with a tab. All indented statements following **def** and the function name belong to the function. In the above function our function name is main.

The last program statement is unindented indicating the end of the main function. This statement calls our main function. It has the same name as the main function and also includes the round brackets. This indentation makes python very awkward to use and is the most cause of many program errors and unexpected program executions. Fortunately, you will get use to indentation and realize its importance in a python program to define the python program structure.

If you get your program running them you are doing good.

Python has many built in functions that you can use, that make python easier to use. You already used some of them in lesson 1, **print**, **input**, **int** and **float**. As we proceed with these lessons you will learn and use many more functions.

It is now time to add more functions to our lesson 2 program.

We will make a welcome, getName ad getAge functions.

Functions usually are defined at the top of the program in order as they are used. The main function is the last one because it will call all the proceeding functions. When a function is called in a programming statement it means it is executed. In a Python script the functions must be defined before they are used.

Here is our program now divided into functions. So, upgrade your program file and run it.

def welcome():

  print("Hello World")

```python
def getName():
    name = input("Please type in your name: ")
    return name


def getAge():
    age = int(input("How old are you? "))
    return age


def printDetails(name, age):
    print("Nice to meet you ", name)
    print (name,"You are", age , "year old")


def main():
    welcome()
    name = getName()
    age = getAge()
    printDetails(name, age)


main()
```

Functions make your program more organized and manageable to use. Functions have three different purposes. Functions can receive values, return values both or neither.   The welcome function just prints a statement and receives no values or returns no value.

def welcome():

  print("Hello World")

The get Name() and getAge() function both return a value using the return statement.

def getName():

  name = input("Please type in your name: ")

  return name

def getAge():

  age = int(input("How old are you? ")

  return age

The printDetails function receive a name and age value to print out, but return's no value.

def printDetails(name, age):

  print("Nice to meet you ", name)

  print (name,"You are", age , "year old")

The **name** and **age** inside the round brackets of the **printDetails** function definition statement are known as **parameters** and contain values to be used by the function. The parameters just store values from the calling function and are not the same variables that are in the calling function. Although the parameter names and values may be same as in the calling function variable names, they are different memory locations. The main purpose of the parameters is to transfer values to the function. The main functions call the preceding functions to run them and store the values in variables and pass the stored variable values to the functions. Calling a function means to execute the function. The values that are passed to the called function from the calling function are known as arguments. Variables inside a function are known as local variables and are known to that function only. Name and age are local variables in the main function but are also arguments to the printDetails function.

```
def main():
    welcome()
    name = getName()
    age = getAge()
    printDetails(name, age)
```

Its now time to comment you program All programs need to be commented so that the user knows what the program is about. Just by reading the comments in your program you will know exactly what the program is supposed to do. We have two types of comments in python. Header comments that are at the start of a program or a function. They start with 3 double quotes and end with three double quotes and can span multiple lines like this.

"""

Program to read a name and age from a user and
print the details on the screen

"""

Other comments are for one line only and explain what the current or proceeding program statement it is to do,

The one-line comment starts with a # like this:

 # function to read a name from the key board are return the value

We now comment the program as follow. Please add all these comments to your program.

"""

Program to read a name and age from a user and print

the details on the screen

"""

 # function to print a welcome message

def welcome():

  print("Hello World")

 # function to read a name from the key board are return the value

def getName():

  name = input("Please type in your name: ")

  return name

```python
# function to read an age from the key board are return the value
def getAge():
    age = int(input("How old are you? "))
    return age


# function to print out a user name and age
def printDetails(name, age):
    print("Nice to meet you ", name)
    print (name,"You are", age , "year old"


# main function to run program
def main():
    welcome()  # welcome user
    name = getName() # get user name
    age = getAge() # get user age
    printDetails(name, age) # print user name an age


main() # call main function to run program
```

Lesson 3   Classes

We now take a big step in Python Programing. This is very important step to take. **Classes** represent another level in program organization. They represent programming units that contain variables to store values and contain functions to do operations on these variables that store the values. This concept is known as Object Oriented Programming, and is a very powerful concept. It allows these programming units to be used over again in other programs. The main benefit of a class is to store values and do operations on them transparent from the user of the class. It is very convenient for the programmers to use classes. The are like building blocks that are to create many sophisticated programs with little effort.

A class starts with the keyword **class** and the class name. The class uses another keyword **self** that indicates which variables and functions belong to this class. The keyword **self** is a little awkward to use, but we have no choice but to accept and use properly. Class definitions are little more automatic in other programming languages. Classes in Python are just probably an add on hack. All functions in a Python class must contain the **self** keyword.

We now convert our previous program to use a class. We will have a Person class that has variables to stores a name and age and includes functions to do operations on them, like initializing retrieval, assignment and output.

"""

Person Class to store a person's name and age

A main function to read a name and age from a user and print

the details on the screen using the Person class

"""

```python
# define a class Person
class Person:
    # initialize Person
    def __init__(self, name, age):
        self.name = name
        self.age = age

    # return name
    def getName(self):
        return self.name

    # return age
    def getAge(self):
        return self.age

    # assign name
    def setName(self,name):
        self.name = name

    # assign age
    def setAge(self, age):
        self.age = age
```

 # return person info as a string

def __str__(self):

    sout = "Nice to meet you " + self.name + "\n";

    sout += self.name + " You are " + str(self.age) + " years old"

    return sout

The Person class definition starts with the class key word and class name Person

class Person:

A class contains an **__init__()** function that initializes the class. This **__init__()** function is also known as a **constructor**. (__ is 2 under scores) The mechanism that allocates memory in the computer for the variables and function code defined in the class, is known as **instantiation**.

 # initialize Person

    def __init__(self, name, age):

      self.name = name

      self.age = age

Notice the **self** keyword in the constructor parameter list. The **self** keyword is passed to every function in the Person class and represents the memory location of the Person class variables. The programming statements inside the constructor define the Person class variables name and age and are assigned values from the parameters name and age.

    self.name = name

    self.age = age

The keyword **self** specifies which variables belongs to the Person class. The parameter name and age just store values to be assigned to the Person class variables and are not the same ones in the Person class.

The get functions also known as **getters** and just return values of the variables stored in the Person class. Again, you notice the self keyword.

```
# return name

  def getName(self):

    return self.name


 # return age

  def getAge(self):

    return self.age
```

We also have set functions known as **mutators** that allow the user of the class to assign new values to the stored Person class variables.

```
  # assign name

  def setName(self,name):

    self.name = name


  # assign age

  def setAge(self, age):

    self.age = age
```

You will notice the parameter list has the keyword self and an additional parameter to assign the name or age value. Again, the **self** keyword distinguishes the person variables from the parameters since they both have the same names.

All classes should have a **__str__()** function so that it can easily return the class variables as a string message.

```
# return person info as a string

def __str__(self):

    sout = "Nice to meet you " + self.name + "\n";

    sout += self.name + " You are " +  str(self.age) +  " years old"

    return sout
```

Notice we have no print statement in our **__str__** function. We assign information to the local variable sout and return the sout value. A local variable is just known to the function it resides in. The sout variable must use the + operator to join values together as a message. Unfortunately, the + operator only joins string variables in case of numeric values the **str** function is used to convert a numeric value to string value. The **str** function is a common built in python function that return a string value.

The class definition must not contain any input or output statements. A class must be a reusable program unit not dependent on any input or output print statements. The purpose of the class is to contain information that can be easily accessed.

Therefore, our main function must provide all the input and output print statements. We will use the input and output functions from our previous program.

```
# function to print a welcome message

def welcome():

    print("Hello World")


# function to read a name from the key board are return the value

def getName():

    name = input("Please type in your name: ")

    return name
```

```python
# function to read an age from the key board are return the value
def getAge():
    age = int(input("How old are you? "))
    return age


# main function to instantiate class Person and run program
def main():
    welcome()  # welcome user
    name = getName() # get user name from keyboard
    age = getAge() # get user age from key board
    p = Person(name, age) # create Person class
    print(p) # print user name and age from person class


main() # call main function
```

Notice we create the Person class with the following statement:

p = Person(name, age)   # create Person class

This statement calls the **__init__()** function of the person class to create the person object and initialized with the values name and age. When a class is instantiated in computer memory it is known as an **object**, when a class is written in a program then it is still known as a class not an object.
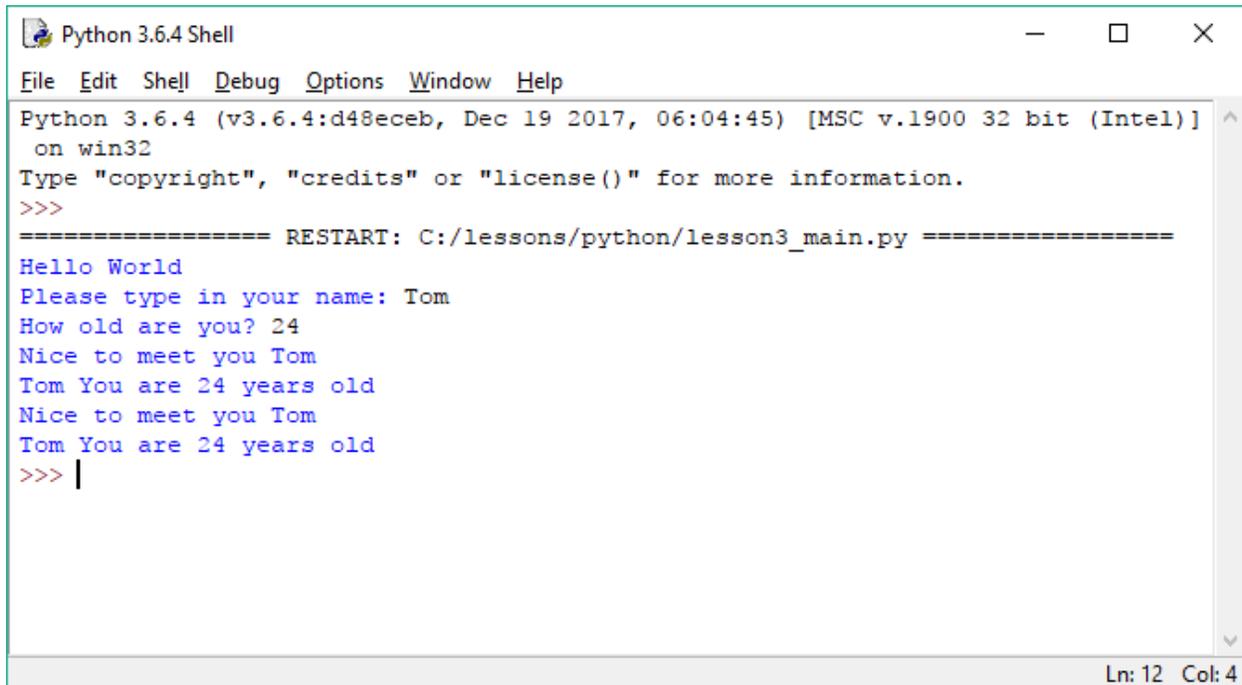
In the print function the print statement automatically calls **__str__()** function

print(p)

Type in the class and main function in a python program lesson3.py. and run the program. You will get the same output as the previous program.

Once you got this program running make a new person called p2. Copy the values from person p using the getters and print out the person p2 details,

You should get something like this:

```
Python 3.6.4 Shell                                          —   □   ✕

File  Edit  Shell  Debug  Options  Window  Help
Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017, 06:04:45) [MSC v.1900 32 bit (Intel)] ^
 on win32
Type "copyright", "credits" or "license()" for more information.
>>>
================= RESTART: C:/lessons/python/lesson3_main.py =================
Hello World
Please type in your name: Tom
How old are you? 24
Nice to meet you Tom
Tom You are 24 years old
Nice to meet you Tom
Tom You are 24 years old
>>> |



                                                          Ln: 12  Col: 4
```

If you can do this, you are almost great python programmer!

Classes are usually put into their own files So put all the Person class code in a file called person.py. Put all the main functions in a file called lesson3_main.py. Lesson3_main.py needs some extra statements. We need to tell file lesson3_main.py to use the Person class. Put this statement just below the header comment of lessone3_main.py file near the top of the file.

from person import Person

This means from file person.py use the code from the Person class. You also need to tell the python interpreter that lesson3_main.py is the file to run. You cannot run a class without a main program. Classes do not run by them selves. Put this statement just above the main() statement at the bottom of the file.

```
if __name__ == "__main__":

    main() # call main function
```

The beauty of classes is that they can be extended to increase their functuality. We can make a Student class using the variables and functions from the Person class. This is known as **inheritance**.

A Student class will have an additional variable called idnum that will represent a string student id number. Using inheritance, the student class will be able to use the variables and functions of the Person class. The Person class is known as the **super** class and the Student class is known as the **derived** class. The Person class knows nothing about the Student class where as the Student class knows all about the Person class.

Open a new python file called student.py. Create a class called Student using this statement

Class Student (Person)

The above statement means to define a class Student that inherits the Person class.

On the top of the student.py file just below the header comment you need to tell the python interpreter to use the Person class

from person import Person

This means from the file person.py use code from the Person class**.** Now make a constructor that will initialize the student name, age and idnum.

```
# initialize Student

def __init__(self, name, age, idnum):

    self.name = name

    self.age = age

    self.idnum = idnum
```

Notice how you can easily use the variables of the super Person class.

You should now be able to make the getID and setID getters and setters without our help.

The last thing you need to make the **__str__()** function. By using the super() function you can call functions directly from the super Person class inside the Student derived class. Here is the Student **__str__()** function

 # return student info as a string

def __str__(self):

sout = super().__str__()

sout += " having Student ID: " + self.idnum;

return sout

Once you got the Student class made then add programming statements to the lessons3_main.py file to obtain a student name, age and idnum. You will have to make an additional getID() function to obtain a student id number from the key board.

You will also need an additional import statement at the top of the file for the Student class.

from student import Student

Then make a student object and use the obtained name, age and idnum then print out the student details. You should get something like this:

```
Python 3.6.4 Shell                                              —    □    ×

File  Edit  Shell  Debug  Options  Window  Help

Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017, 06:04:45) [MSC v.1900 32 bit (Intel)]
 on win32
Type "copyright", "credits" or "license()" for more information.
>>>
================= RESTART: C:/lessons/python/lesson3_main.py =================
Hello World
Please type in your name: Tom
How old are you? 24
Nice to meet you Tom
Tom You are 24 years old
Nice to meet you Tom
Tom You are 24 years old
Please type in your name: Sue
How old are you? 36
What is your Student ID? S1234
Nice to meet you Sue
Sue You are 36 years old having Student ID: S1234
>>> |


                                                               Ln: 17  Col: 4
```

Lesson 4 Operators, Lists, Tuples and Dictionaries

Operators

Operators do operations on variables like addition + , subtraction -comparisons > < etc. You can test Python programing statements directly on the Python shell, which is very convenient to use. You just type in the program statement and it gets executed automatically. To see the value of a variable you just type in the variable name and the value is displayed automatically.

```
Python 3.6.4 Shell                                               —    □    ✕
File  Edit  Shell  Debug  Options  Window  Help
Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017, 06:04:45) [MSC v.1900 32 bit (Intel)]
 on win32
Type "copyright", "credits" or "license()" for more information.
>>> x = 3 + 4
>>> x
7
>>> x = 3 * 4
>>> x
12
>>> x = 3 > 4
>>> x
False
>>> x = 3 < 4
>>> x
True
>>> |
                                                             Ln: 15  Col: 4
```

We now present all the Python operators. You can type all the examples in the python shell or put in a file called operators.py

If you use a file, then you will have to use print statements to see the result on the screen.

Arithmetic Operators

Arithmetic operators are used to do operations on numbers like addition and subtraction.

Operator	Description	Example	Result
+	Add two operands or unary plus	x = 3 +2	5
-	Subtract right operand from the left or unary minus	x -= 3-2 x = -2	1 -2
*	Multiply two operands	x = 3 * 2	6
/	Divide left operand by the right one	x = 5 / 2	2.5
%	Modulus - remainder of the division of left operand by the right	x = 5 % 2	3
//	Floor division - division that results into whole number	x = 5 // 2	2
**	Exponent - left operand raised to the power of right	x = 5**2	25

Comparison Operators

Comparison operators are used to compare values. It either returns True or False according to the condition. In python true and false start with capital letters True or False**.**

Operator	Description	Example	Result
>	Greater than - True if left operand is greater than the right	5 > 3	True
<	Less than - True if left operand is less than the right	3 < 5	True
==	Equal to - True if both operands are equal	5 == 5	True
!=	Not equal to - True if operands are not equal	5!= 5	True
>=	Greater than or equal to - True if left operand is greater than or equal to the right	5 >= 3	True
<=	Less than or equal to - True if left operand is less than or equal to the right	5 <= 3	True

Logical Operators

Logical operators are the **and**, **or**, *no*t boolean operators.

Operator	Description	Example	Result
and	True if both the operands are true	True and True	True
or	True if either of the operands is true	True or False	True
not	True if operand is false (complements the operand)	Not False	True

Bitwise Operators

Bitwise operators act on operands as if they were binary digits. It operates bit by bit. Binary numbers are base 2 and contain only 0 and 1's. Every decimal number has a binary equivalent. Every binary number has a decimal equivalent. For example, decimal 2 is 0010 in binary and decimal 7 is binary 0111.

In the table below: Let x = 10 (0000 1010 in binary) and y = 4 (0000 0100 in binary)

Operator	Description	Example and Result
&	Bitwise AND	x & y = 0 (0000 0000)
|	Bitwise OR	x | y = 14 (0000 1110)
~	Bitwise NOT	~x = -11 (1111 0101)
^	Bitwise XOR	x ^ y = 14 (0000 1110)
>>	Bitwise right shift	x>> 2 = 2 (0000 0010)
<<	Bitwise left shift	x<< 2 = 40 (0010 1000)

Assignment Operators

Assignment operators are used in Python to assign values to variables.

x = 5 is a simple assignment operator that assigns the value 5 on the right to the variable a on the left.

There are various compound operators in Python like x += 5 that adds to the variable and later assigns the same. It is equivalent to x = x + 5.

Operator	Compound	Equivalent
=	x = 5	x = 5
+=	x += 5	x = x + 5
-=	x -= 5	x = x - 5
*=	x *= 5	x = x * 5
/=	x /= 5	x = x / 5
%=	x %= 5	x = x % 5
//=	x //= 5	x = x // 5
**=	x **= 5	x = x ** 5
&=	x &= 5	x = x & 5
|=	x |= 5	x = x | 5
^=	x ^= 5	x = x ^ 5
>>=	x >>= 5	x = x >> 5
<<=	x <<= 5	x = x << 5

Identity Operators

is and **is not** are the identity operators in Python. They are used to check if two values (or variables) are located on the same part of the memory. Two variables that are equal does not imply that they are identical.

Operator	Description	Example
is	True if the operands are identical	x is y
is not	True if the operands are not identical	x is not y

As stated earlier you should type in all the examples and try them out. You will be using them soon in the next lesson.

String Operators

```
# join two strings together

s1 = "hello"
s2 = "there"
s3 = s1 + s2
print(s3)    # hellothere


# get a character from string

c = s3[0]
print (c)    # h


# get a substring

s4 = s3[0:5]
print (s4) # hello


# add a character to a string

s5 = s3[0:5] + 'X' + s3[5:]
print (s5) # helloXthere


# reverse a string

s6 = s5[::-1]
print (s6) # erehtXolleh
```

make string upper case

s7 = s6.lower() # erehtxolleh
print(s7) # EREHTXOLLEH

make string upper case

s8 = s6.upper()
print(s8) # EREHTXOLLEH

change a character to a ASCII number

x = ord('A')
print(x) #65

#change a ASCII number to a character

c = chr(x)|
print(c) # A

Lists

Lists store many sequential values together. Lists are analogous to arrays in other programing languages. Lists in python are very powerful and can do many things. We now present many list examples. Put the following programing statements in a python file called lesson4.py and run it.

```
# To create an empty list
list1 = []
print(list1)

# To create a list with preinitialized values
list2 = [1,2,3,4, 5]
print(list2)
```

```python
# get the number of elements in an list
x = len(list2)
print (x)

# Create a list preinitialized with one values of a specified length.
# an list of 10 0's is created
list3 = [0] * 10
print(list3)

# Add a value to a list
list1.append(5)
print(list1)

# Print a list
print(list2)

# Get a value from a list at a specified location
x = list2[0]
print(x)

# Get part of a list
x = list2[1:3]
print(x)

# Get last value of list
x = list2[-1:]
print(x)

# Get last 2 values of list
x = list2[-2:]
print(x)

# get all values except last value
x = list2[:-1]
print(x)
```

```python
# get all values except last 2 values
x = list2[:-2]
print(x)

# Reverse a list
x = list2[::-1]
print(x)

# add 2 lists together
x = list1 + list2;
print(x)

# remove a value from a list
list2.remove(3);
print(list2)

# remove a list item by index
del list2[0]
print(list2)

# test if a value is in a list returns True or False
x = 2 in list2
print (x)

x = 1 in list2
print (x)

# join 2 lists together (result is list inside a list)
list1.append(list2);
print(list1)
```
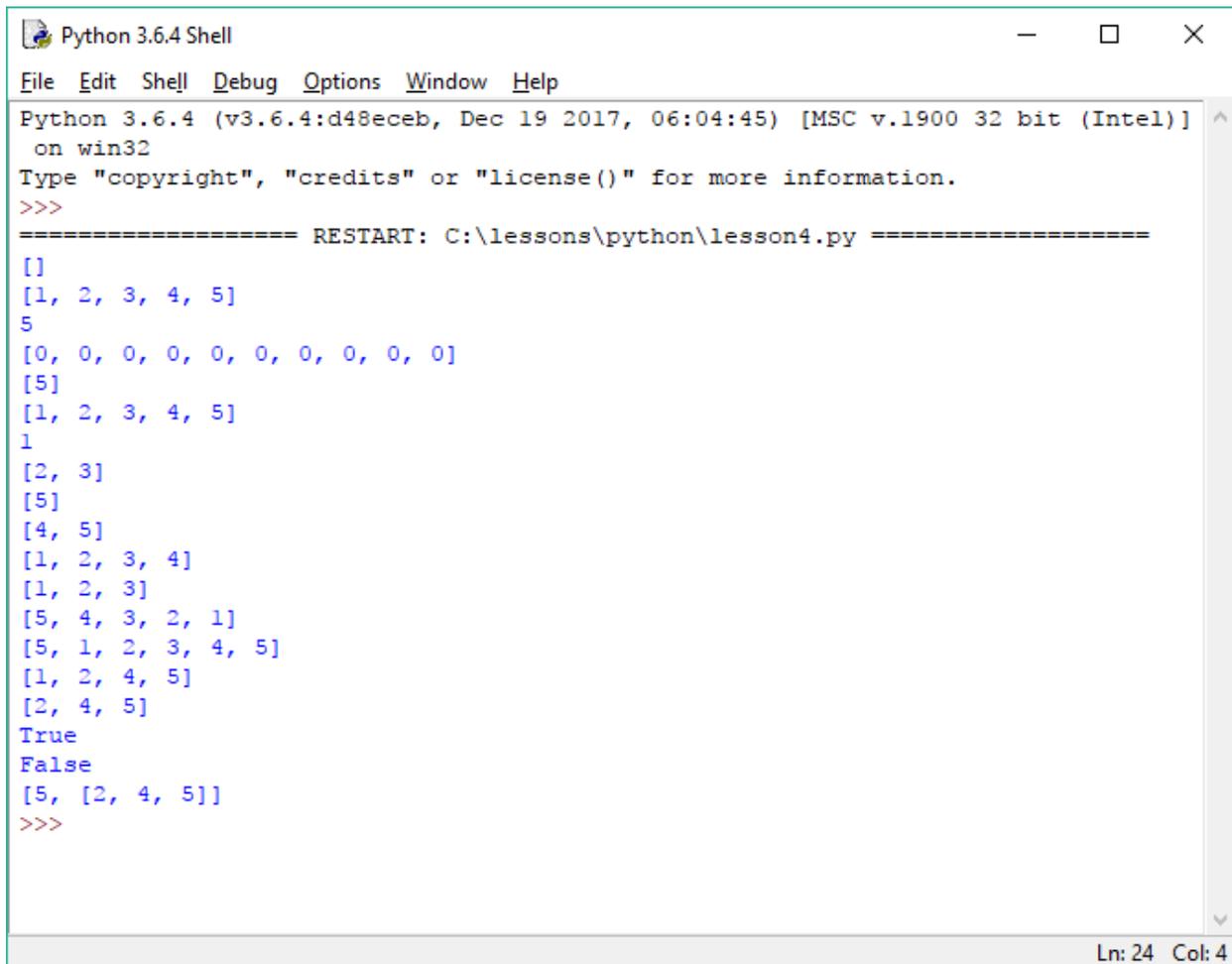
This should be your program output

```
Python 3.6.4 Shell                                               −    □    ×

File  Edit  Shell  Debug  Options  Window  Help

Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017, 06:04:45) [MSC v.1900 32 bit (Intel)]
 on win32
Type "copyright", "credits" or "license()" for more information.
>>>
================== RESTART: C:\lessons\python\lesson4.py ==================
[]
[1, 2, 3, 4, 5]
5
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[5]
[1, 2, 3, 4, 5]
1
[2, 3]
[5]
[4, 5]
[1, 2, 3, 4]
[1, 2, 3]
[5, 4, 3, 2, 1]
[5, 1, 2, 3, 4, 5]
[1, 2, 4, 5]
[2, 4, 5]
True
False
[5, [2, 4, 5]]
>>>

                                                            Ln: 24  Col: 4
```

Two Dimensional Lists

Two dimensional lists are analogues to 2 dimensional arrays in other programming languages. To make a two dimensional list you make a one dimensional list and then assign additional one dimensional lists to it.

 # make a one-dimensional list of size 3

List3 = [None] * 3;

We use **None** as a value because the one dimensional will include another one-dimensional array. (None means nothing or null in other programming languages)

assign one dimensional list's of size 5 to the one-dimensional list

```
List3[0] = [0] * 5

List3[1] = [0] * 5

List3[2] = [0] * 5

Print(list3)  # [[0, 0, 0], [0, 0, 0], [0, 0, 0]]
```

assign value by row and column

```
list3[1][2] = 5
```

retrieve value by row and column

```
x = list3[1][2]

print(x)  # 5
```

Tuples

Tuples store a group of values. They are not as powerful as lists and their main purpose is to return multiple value from functions or represent multiple values when they are needed. They are read only, meaning you cannot change their values once they are created.

```
# make a tuple

t = (1,2,3)

print(t)    # (1,2,3)
```

```
# retrieve a value from a tuple

x = t[0]

print(x) # 1
```

```python
# print values from a tuple

print(t[0])  # 1

print(t[1])  # 2

print(t[2])  # 3
```

Dictionaries

Dictionaries contain a key and a value. A dictionary can have many keys and corresponding values. Think of a dictionary is like a telephone book with the name as the key and the telephone number as the value.

```python
 # make empty dictionary

d = {}

print (d)  # {}


# add values to a dictionary

d["name"] = "Tom"

d["age"] = 24

d["idnum"]= "S1234"

print(d)   # {'name': 'Tom', 'age': 24, 'idnum': 'S1234'}

# make dictionary with values

d2 = {"name": "tom", "age": 24, "idnum": "S1234"}

print (d2);   #  {'name': 'tom', 'age': 24, 'idnum': 'S1234'}

# get values from a dictionary

print(d2["name"])   # tom

print(d2["age"])    # 24

print(d2["idnum"])  # S1234
```

#print keys of a dictionary

keys = d2.keys()

print(keys)

```
dict_keys(['name', 'age', 'idnum'])
```

 # print values of a dictionary

values = d2.values()

print(values)

```
dict_values(['tom', 24, 'S1234'])
```

Type all the above examples in your file lesson4.py and, make sure you get the same results. We will be using the lists, tuples and dictionaries in our next lesson.

Lesson 5   Programming Statements

Programming statements allow you to write complete Python Program scripts. We have already looked at simple input, print and assignment statements. We now need to control the flow of our program. Branch control statements allow certain program statements to execute and other not. Loop control statements allow program statements to repeat themselves.

Start a new python program lesson5.py to test al the control statements

"""

lesson5.py

Programming statements

"""

Branch Control Statements

The **if** branch **control** statements use conditional operators from the previous lesson to direct program flow.

If condition :

  Statement(s)

When the condition is evaluated to be true the statements belonging to the if statement will execute.

> **# if statement**
>
> **x = 5**
>
> **if x == 5:**
>
> > **print("x is 5")**
>
> **print("I like Python Programming")**

```
x is 5

I like Python Programming
```

We now add an else statement. An if-else control construct is a two-way branch operation.

If condition :

   statements

else:

   statements

 # if – else statement

x = 2

if x == 5:

   print("x is 5")

else:

   print ("x is not 5")

print("I like Python Programming")


```
x is not 5

I like Python Programming
```

We can also have extra else if statements to make a multi-branch. Python contracts else if to **elif**

```
# multi if else

x = 10

if x == 5:

    print("x is 5")

elif x < 5:

    print("x less than  5")

elif x > 5:

    print("x greater than  5")

print("I like Python Programming")
```

```
x greater than  5

I like Python Programming
```

Our multi branch if-else can also end with an else statement.

```
# multi if-else else

x = 5

if x < 5:

        print("x less than  5")

elif x > 5:

    print("x greater than  5")

else: print("x is 5")
print("I like Python Programming")
```

```
x is  5

I like Python Programming
```

if statements can be nested to make complicated conditions simpler

 # nested if statement

x = 5

if x >= 0:

  if x > 5:

    print("x greater than 5")

  else:

    print("x less than equal 5")

```
x less than equal 5
I like Python Programming
```

While loop

Our next control statement is the while loop

while condition:

    statement(s)

The while loop allows you to repeat programming statements repeatedly until some condition is satisfied

 # while loop

x = 5

while x > 0:

  print(x)

  x-=1

```
5
4
3
2
1
```

For Loop

The other loop is the for loop. Ii is much more powerful then the while loop but more difficult to use.

All loops much have counter mechanism. The for loop uses the range functuin that supplies' the couther stat value step value and end value for the for loop.

for counter in range(start_value, increment, end_value-1, increment):

    Statement(s)

The default value for increment is 1

Using a for loop to loop 1 to 5 using the range function. The i variable cannot be changed and belongs to the for loop.

 # for loop using range

for i in range(1,5+1,1):

    print (i)

```
1
2
3
4
5
```

Here is a for lop that counts backwards using a negarive increment

 # for loop using range counting backward

for i in range(5,1-1,-1):

   print (i)

```
5
4
3
2
1
```

Nested for loops

Nested for loops are used to print out 2 dimensional objects by row and column

 # nested for loop

for r in range(1,5+1):

   print(r, ":", end="")

   for c in range(1,5+1):

      print("",c,end="")

   print("")

Note we use the end directive so we do not start a new line every time we print out a column value

```
1 : 1 2 3 4 5

2 : 1 2 3 4 5

3 : 1 2 3 4 5

4 : 1 2 3 4 5
```

Loops can also be used to print out characters in a string variable

 # print out characters in a string

s = "Hello"

for c in s:

    print(c)

```
H
e
l
l
o
```

We also can print out the values in a list

For loops can also print out values from lists

 # print out values in a list

list1 = [1,2,3,4,5]

for x in list1:

 print (x)

```
1
2
3
4
5
```

Here we print out values from a two-dimensional array

```
# print out two-dimensional list
list2 = [[1,2,3],[4,5,6],[7,8,9]]
for r in list2:
    for c in r:
print (c,end=" ")
print("")
```

```
1 2 3
4 5 6
7 8 9
```

For loops can also print out dictionaries. The main purpose is to print out the dictionary in order by key or order by values

```
# print dictionary by key
# make dictionary
d = {"name": "tom", "age": "24", "idnum": "S1234"}
for key, value in d.items() :
        print (key, value)
```

```
name tom

age 24

idnum S1234
```

print dictionary by value

for key, value in d.items() :

  print (value, key)

```
tom name

24 age

S1234 idnum
```

print dictionary sorted by key

for key in sorted(d.keys()):

  print ("%s: %s" % (key, d[key]))

```
age: 24

idnum: S1234

name: tom
```

print dictionary sorted by value

```python
for v in sorted( str(d.values()) ):
    for key in d:
        if d[ key ] == v:
            print (key, v)
            break
```

```
age: 24

idnum: S1234

name: tom
```

File Access

input.txt

happy days are here again
we will see you tomorrow

```python
# Open the file for read

f = open('input.txt', "r")

# read all lines in the file to a list

lines = f.readlines()

print(lines)

# close the file

f.close()
```

['happy days are here again\n', 'we will see you tomorrow\n']

```python
# Open the file for read

f = open('input.txt')

## Read the first line

line = f.readline()

# read line one at a time

# till the file is empty

while line:

    print (line)

    line = f.readline()

f.close()
```

```
happy days are here again

we will see you tomorrow
```

input.csv

```
one,two,three,four
```

```python
# read csv file

# Open the file for read

f = open('input.csv')

## Read the first line

line = f.readline()

# read line one at a time

# till the file is empty

while line:

    # strip removes '\n'
    # split separates line into tokens

    tokens = line.strip().split(".")

    print (tokens)

    line = f.readline()

f.close()
```

```
['one,two,three,four']
```

 # write lines to a file

f = open("output.txt", "w")

f.write("hello")

f.write("\n")

f.close()

```
hello
```

 # write lines to end of a file

f = open("output.txt", "a")

f.write("hello")

f.write("\n")

f.close()

```
hello
hello
```

END