PYTHON PROGRAMMING - BASICS

Mrs.R.Chithra Devi

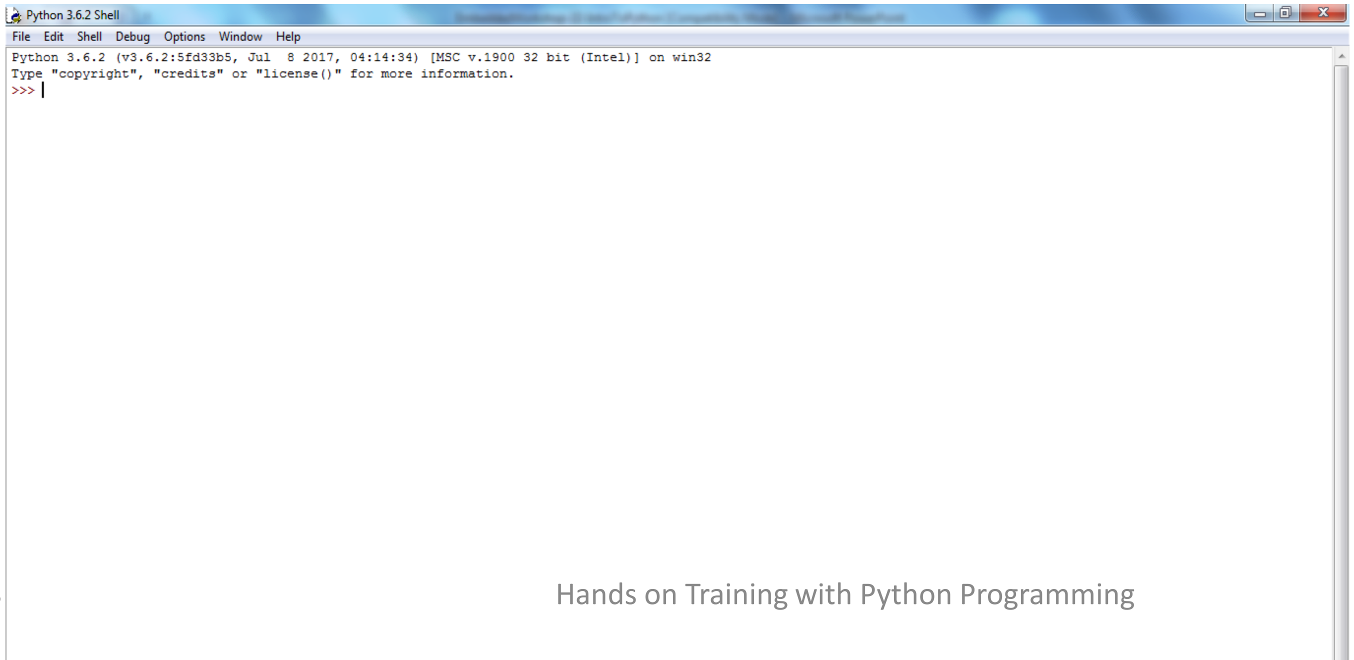
AP/IT, Dr.SACOE

Data and Expressions – Values and Types, Operators

Python Interpreter

Python Interpreter is a program that reads and execute Python Code.

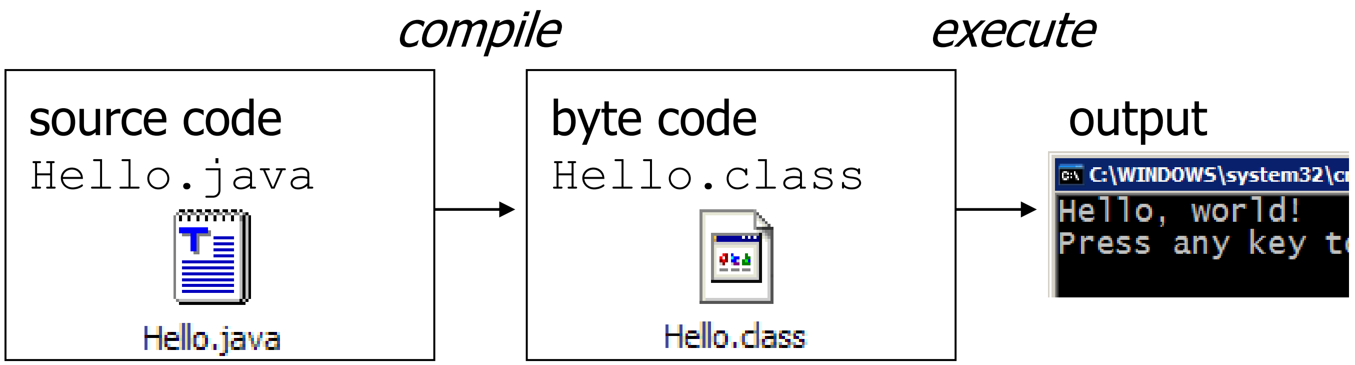
- **On Linux, the Python interpreter is installed as /usr/local/bin/python3.6**
- **On Windows, the Python installation is usually placed in C:\Python36**
- **Depending on your environment, you might start the interpreter by clicking on an icon or by typing python on a command line.**
- **When it start, you should see output like this:**



```
Python 3.6.2 (v3.6.2:5fd33b5, Jul  8 2017, 04:14:34) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> 
```

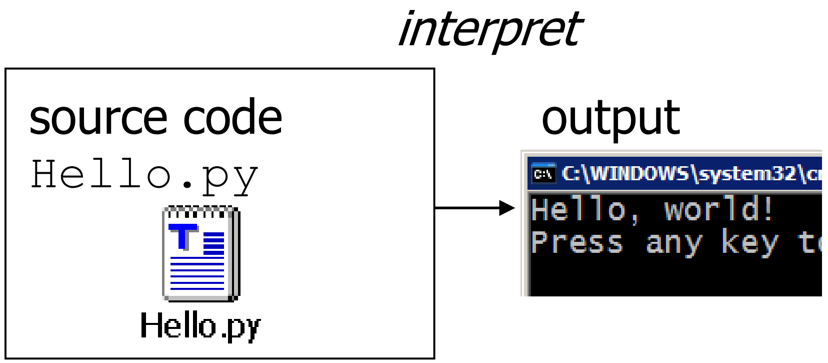
Condt.

Many languages require you to compile (translate) your program into a form that the machine understands.



```
Hello, world!
Press any key t
```

Python is instead directly interpreted into machine instructions.

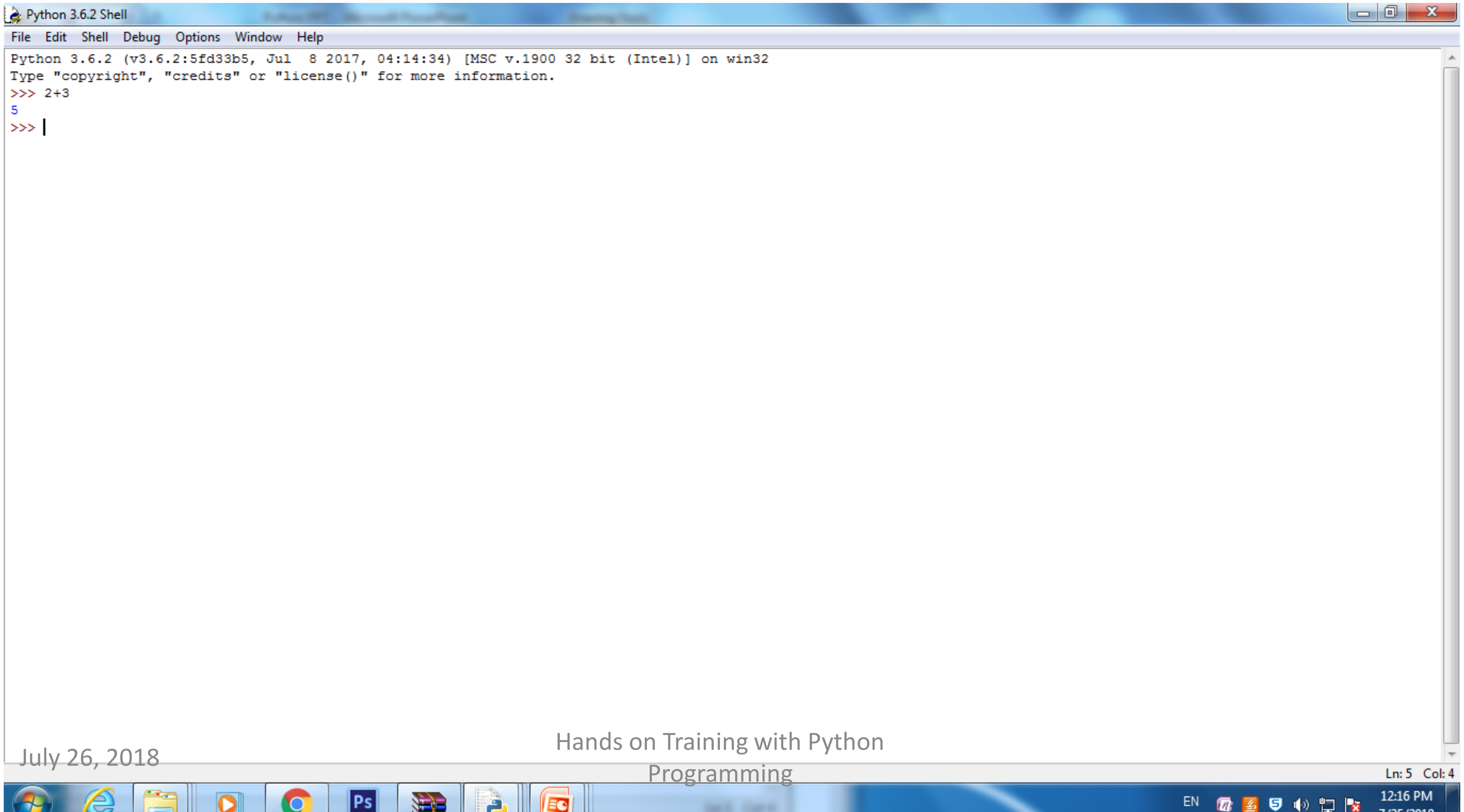


```
Hello, world!
Press any key t
```

Interactive Mode

Two basic Modes: Normal and Interactive

Interactive Mode: A command line shell which gives immediate feedback for each statement.

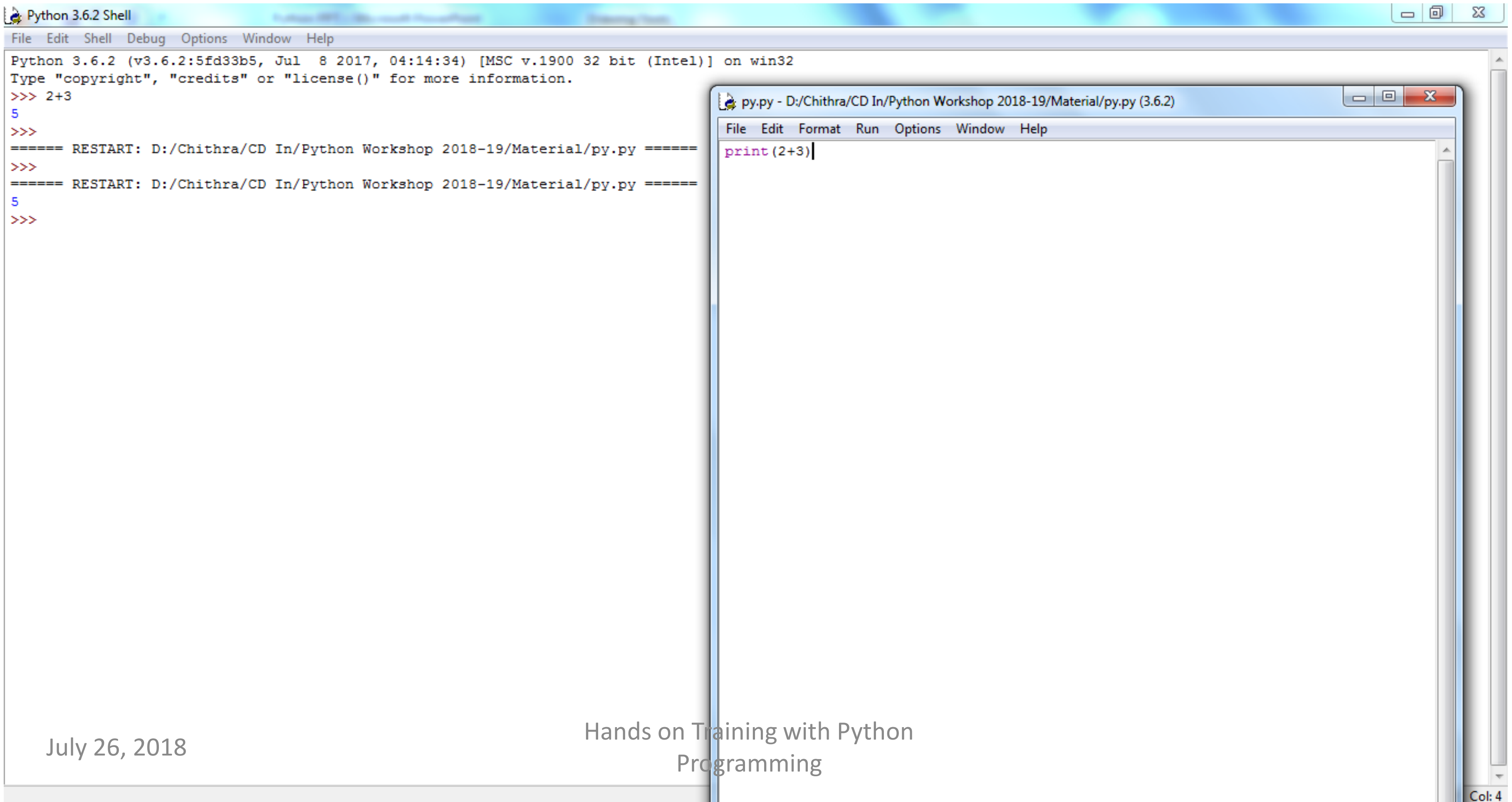


```
Python 3.6.2 (v3.6.2:5fd33b5, Jul  8 2017, 04:14:34) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> 2+3
5
>>> 
```

Script Mode

Normal Mode (Script Mode): The python commands are stored in a file and saved using the extension .py.

Files are run in the Python Interpreter.



Python 3.6.2 Shell
File Edit Shell Debug Options Window Help
py.py - D:/Chithra/CD In/Python Workshop 2018-19/Material/py.py (3.6.2)
File Edit Format Run Options Window Help
Col: 4

```
Python 3.6.2 (v3.6.2:5fd33b5, Jul  8 2017, 04:14:34) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> 2+3
5
>>>
====== RESTART: D:/Chithra/CD In/Python Workshop 2018-19/Material/py.py ======
>>>
====== RESTART: D:/Chithra/CD In/Python Workshop 2018-19/Material/py.py ======
5
>>>
```

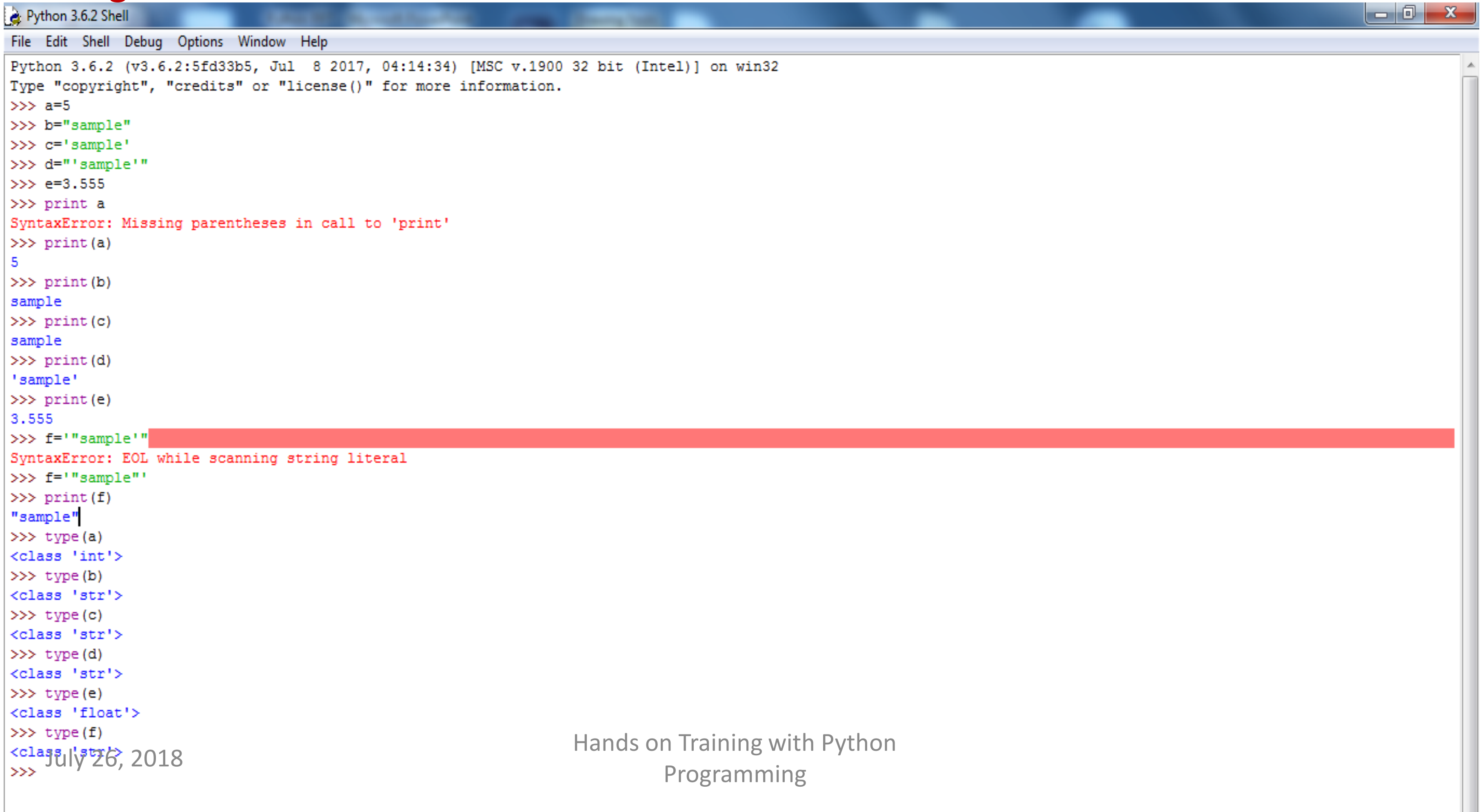
```
print(2+3)
```

Values and Types

Integer – value or number without any decimal point.

Float – value with some decimal point.

String – collection of characters.



```
Python 3.6.2 (v3.6.2:5fd33b5, Jul  8 2017, 04:14:34) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> a=5
>>> b="sample"
>>> c='sample'
>>> d="'sample'"
>>> e=3.555
>>> print a
SyntaxError: Missing parentheses in call to 'print'
>>> print(a)
5
>>> print(b)
sample
>>> print(c)
sample
>>> print(d)
'sample'
>>> print(e)
3.555
>>> f='"sample'"
SyntaxError: EOL while scanning string literal
>>> f='"sample"'
>>> print(f)
"sample"
>>> type(a)
<class 'int'>
>>> type(b)
<class 'str'>
>>> type(c)
<class 'str'>
>>> type(d)
<class 'str'>
>>> type(e)
<class 'float'>
>>> type(f)
<class 'str'>
>>>
```

Variables, Expressions and Statements

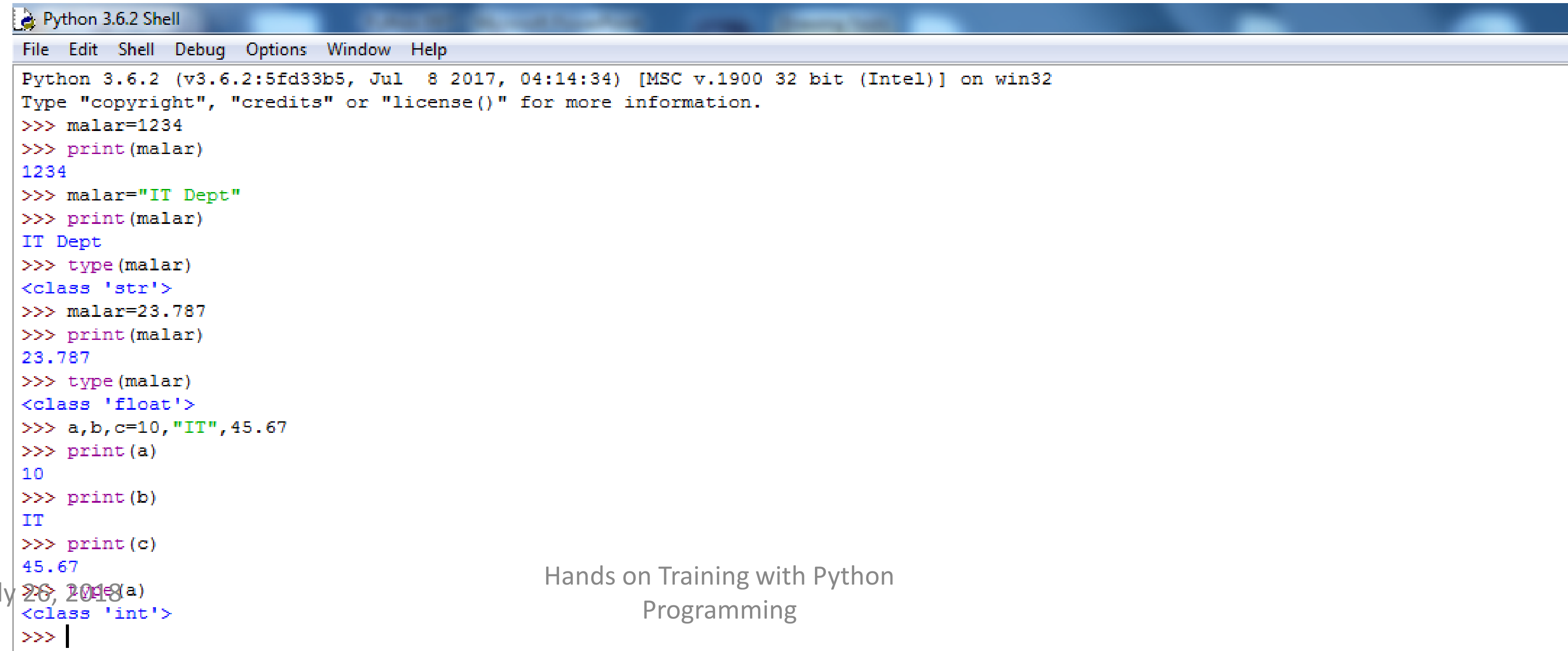
Variables:

It is a reserved memory location to store values. The programmer can assign some values. Programmer choose the variable name which is meaningful.

Assignment Statements:

Creates new variables and then corresponding value can be assigned to it.

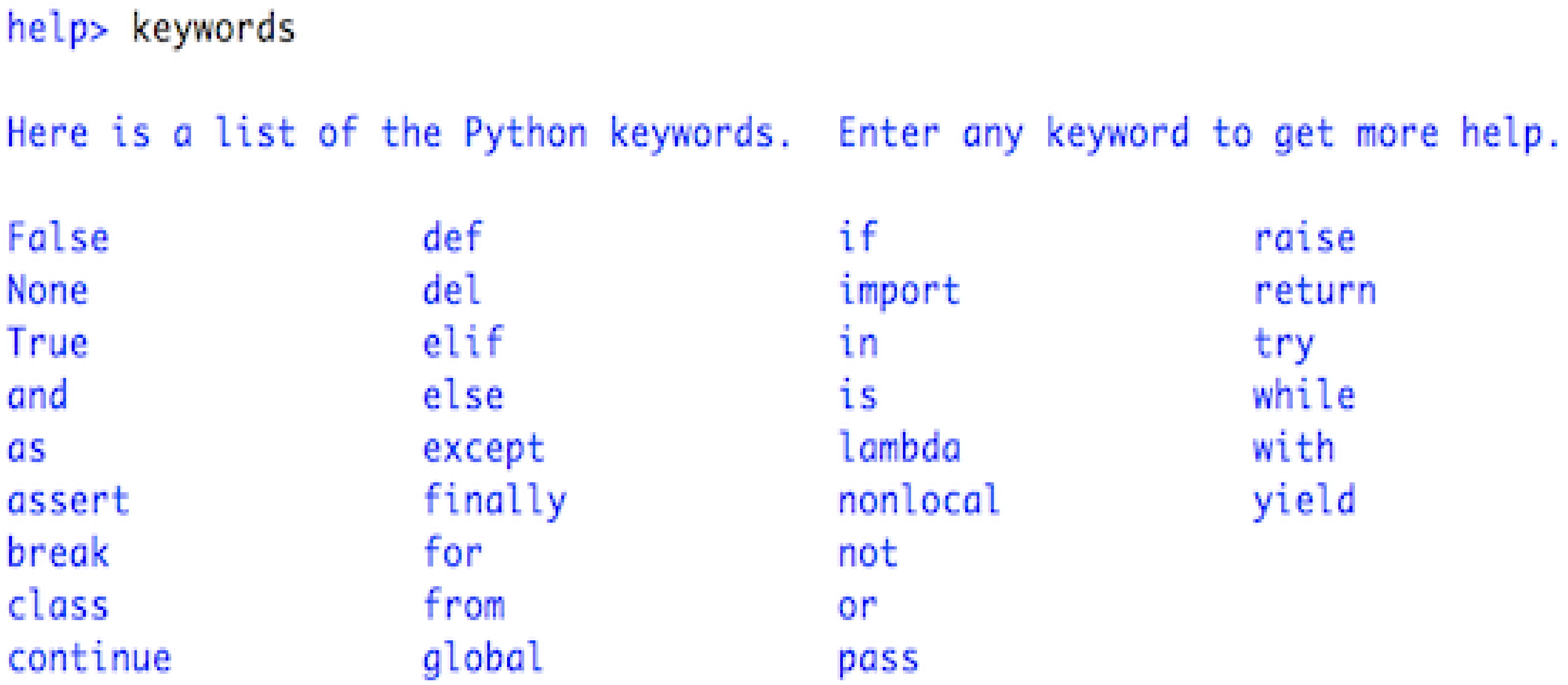
Assignment Operator = (equal)



```
Python 3.6.2 (v3.6.2:5fd33b5, Jul  8 2017, 04:14:34) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> malar=1234
>>> print(malar)
1234
>>> malar="IT Dept"
>>> print(malar)
IT Dept
>>> type(malar)
<class 'str'>
>>> malar=23.787
>>> print(malar)
23.787
>>> type(malar)
<class 'float'>
>>> a,b,c=10,"IT",45.67
>>> print(a)
10
>>> print(b)
IT
>>> print(c)
45.67
>>> type(a)
<class 'int'>
>>>
```

Keywords

The keywords are special words reserved for some purpose.



```
help> keywords

Here is a list of the Python keywords.  Enter any keyword to get more help.

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

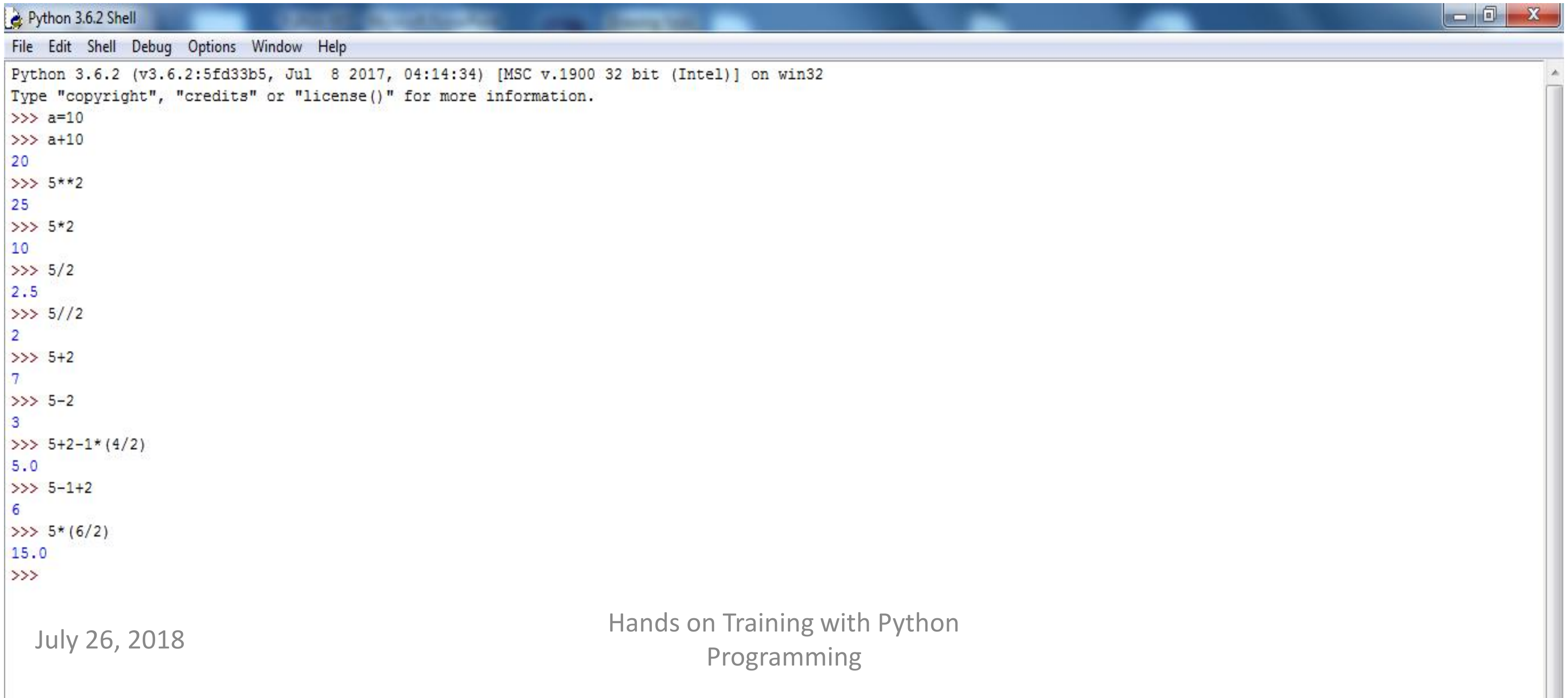
Expressions and Statements, Precedence Operator

Expression is a combination of values, variables and operators.

Precedence Operator: PEMDAS

P – Parentheses, E – Exponential, M – Multiplication, D – Division, A – Addition, S – Subtraction.

Operators with the same precedence are evaluated from Left to Right.



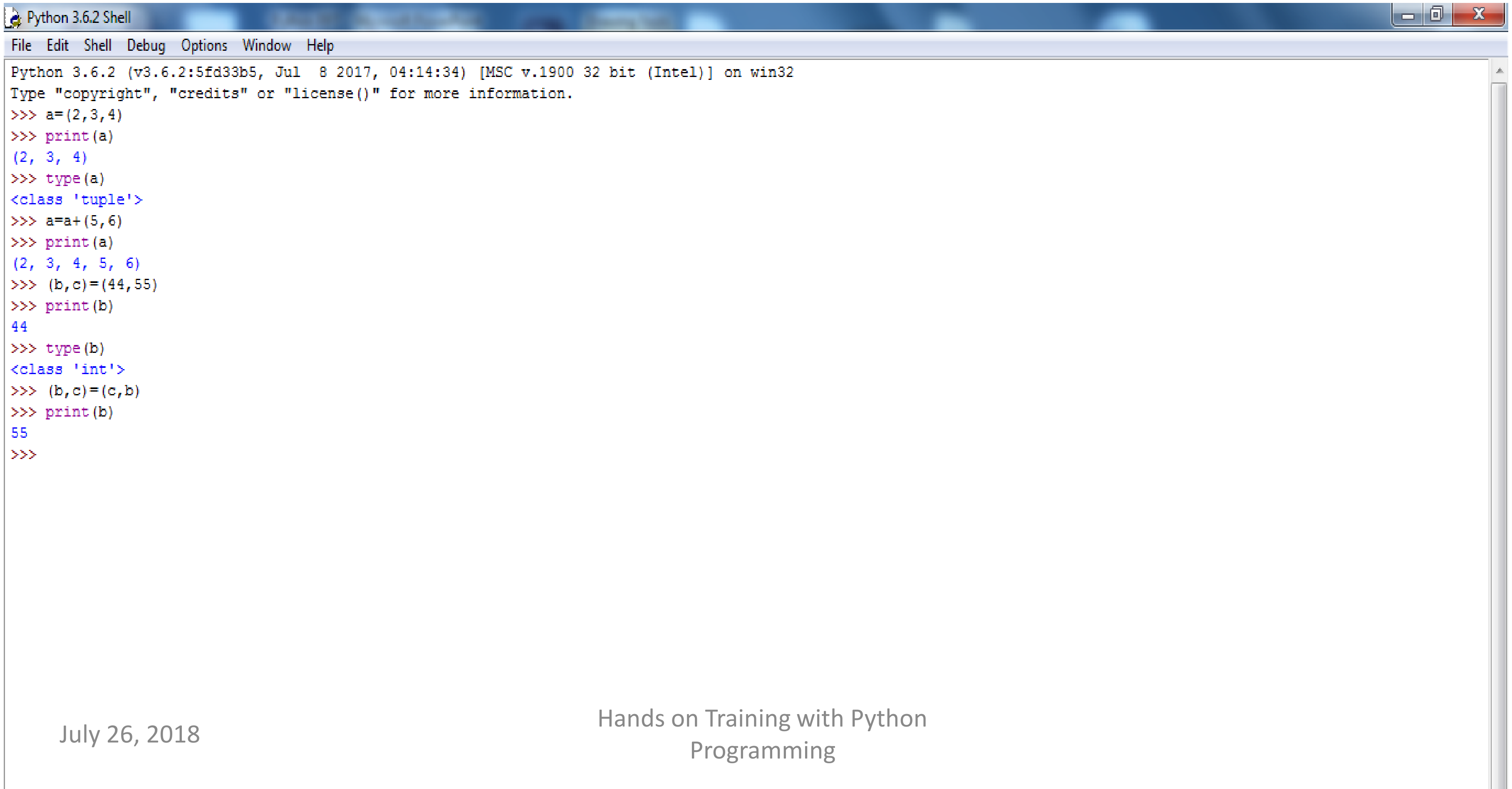
```
Python 3.6.2 (v3.6.2:5fd33b5, Jul  8 2017, 04:14:34) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> a=10
>>> a+10
20
>>> 5**2
25
>>> 5*2
10
>>> 5/2
2.5
>>> 5//2
2
>>> 5+2
7
>>> 5-2
3
>>> 5+2-1*(4/2)
5.0
>>> 5-1+2
6
>>> 5*(6/2)
15.0
>>>
```

Tuple Assignment

Tuple is a sequence of items of any type.

Comma separated list of values and read-only lists.

Enclosed within the parenthesis (()) and their value cannot be modified.



```
Python 3.6.2 (v3.6.2:5fd33b5, Jul  8 2017, 04:14:34) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> a=(2,3,4)
>>> print(a)
(2, 3, 4)
>>> type(a)
<class 'tuple'>
>>> a=a+(5,6)
>>> print(a)
(2, 3, 4, 5, 6)
>>> (b,c)=(44,55)
>>> print(b)
44
>>> type(b)
<class 'int'>
>>> (b,c)=(c,b)
>>> print(b)
55
>>>
```

Lists

Lists are the most versatile data type of Python language. A list consist of items separated by commas and enclosed within square brackets The values stored in a list are accessed using indexes. The index of the first element being 0 and n-1 as that of the last element, where n is the total number of elements in the list. Like strings, you can also use the slice, concatenation and repetition operations on lists.

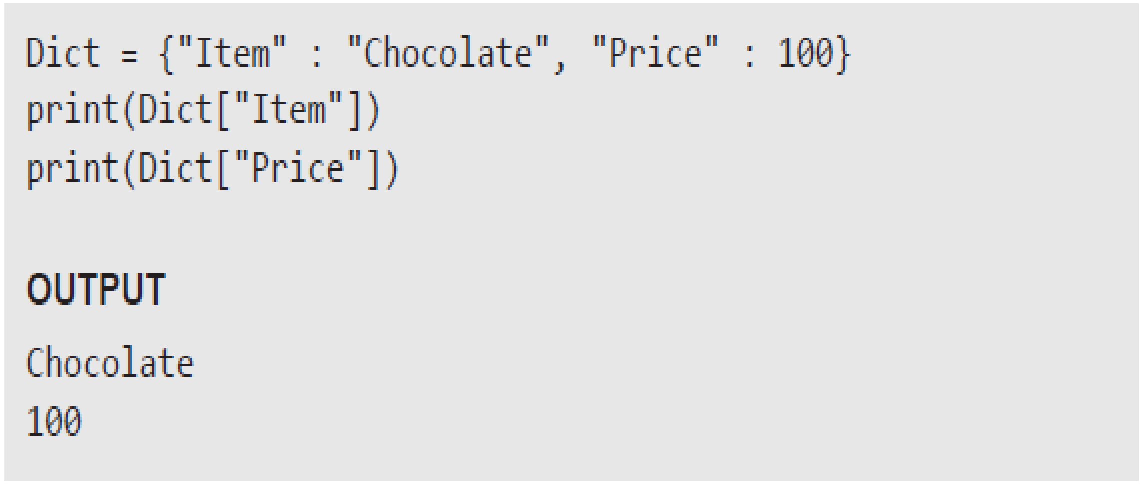
```
list = ['a', 'bc', 78, 1.23]
list2 = ['d', 78]
print(list)
print(list[0]       # Prints first element of the list
print(list[1:3]     # Prints elements starting from 2nd till 3rd
print(list[2:]      # Prints elements starting from 3rd element
print(list *2)       # Repeats the list
print(list + list2)  # Concatenates two lists
```

OUTPUT

```
['a', 'bc', 78, 1.23]
a
['bc', 78]
[78, 1.23]
['a', 'bc', 78, 1.23, 'a', 'bc', 78, 1.23]
['a', 'bc', 78, 1.23, 'd', 78]
```

Dictionary

Python's dictionaries stores data in key-value pairs. The key values are usually strings and value can be of any data type. The key value pairs are enclosed with curly braces ({ }). Each key value pair separated from the other using a colon (:). To access any value in the dictionary, you just need to specify its key in square braces ([]).Basically dictionaries are used for fast retrieval of data



```
Dict = {"Item" : "Chocolate", "Price" : 100}
print(Dict["Item"])
print(Dict["Price"])
```

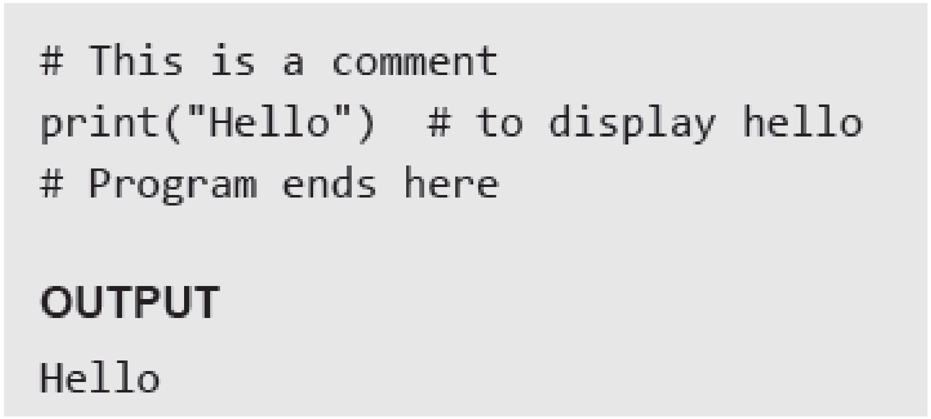
OUTPUT

```
Chocolate
100
```

Comments

Comments are the non-executable statements in a program. They are just added to describe the statements in the program code. Comments make the program easily readable and understandable by the programmer as well as other users who are seeing the code. The interpreter simply ignores the comments.

In Python, we use the hash sign (#) to writing a comment. All characters following the # and up to the end of the line are part of the comment



```
# This is a comment
print("Hello")  # to display hello
# Program ends here
```

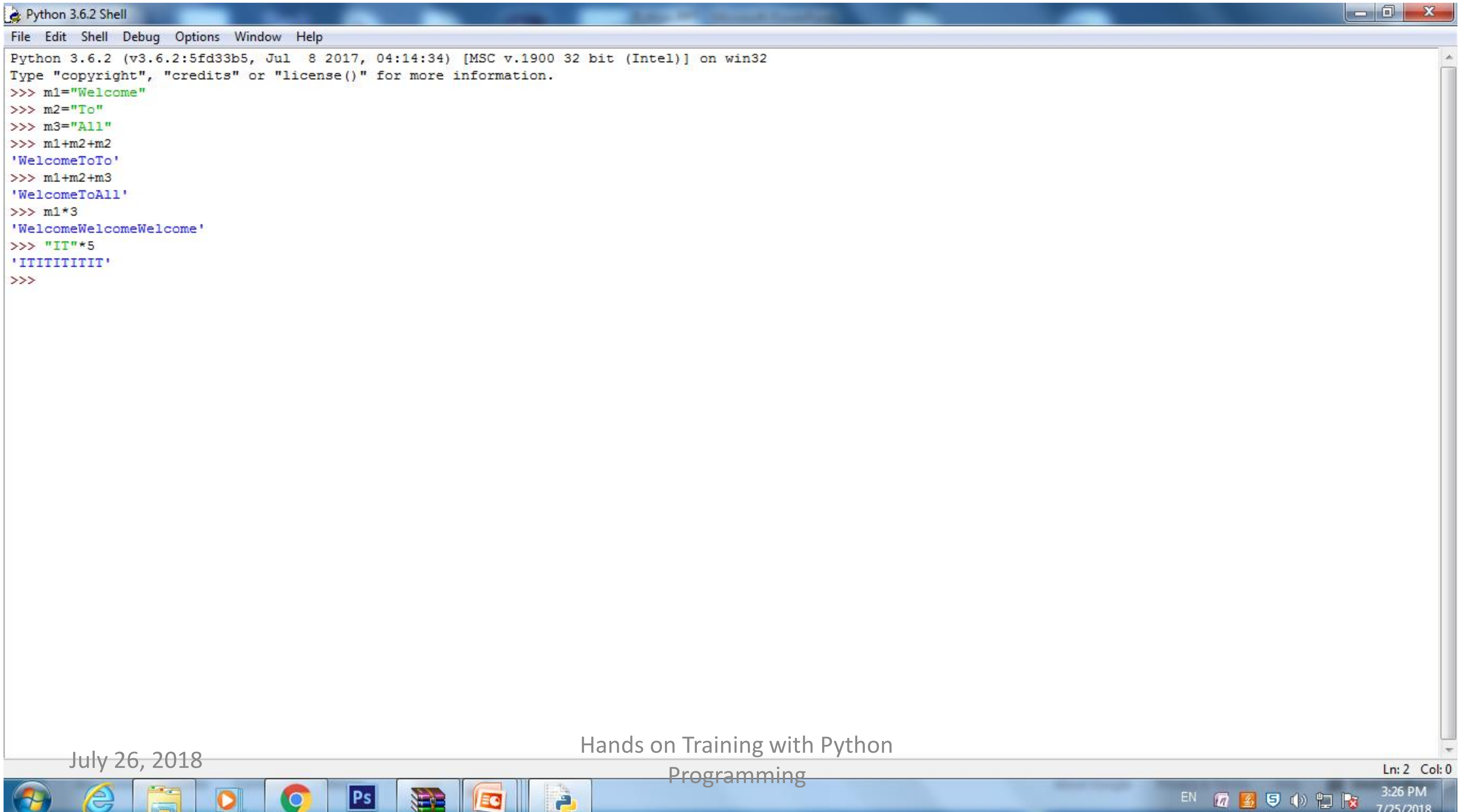
OUTPUT

```
Hello
```

String Operations

String is collection of characters.

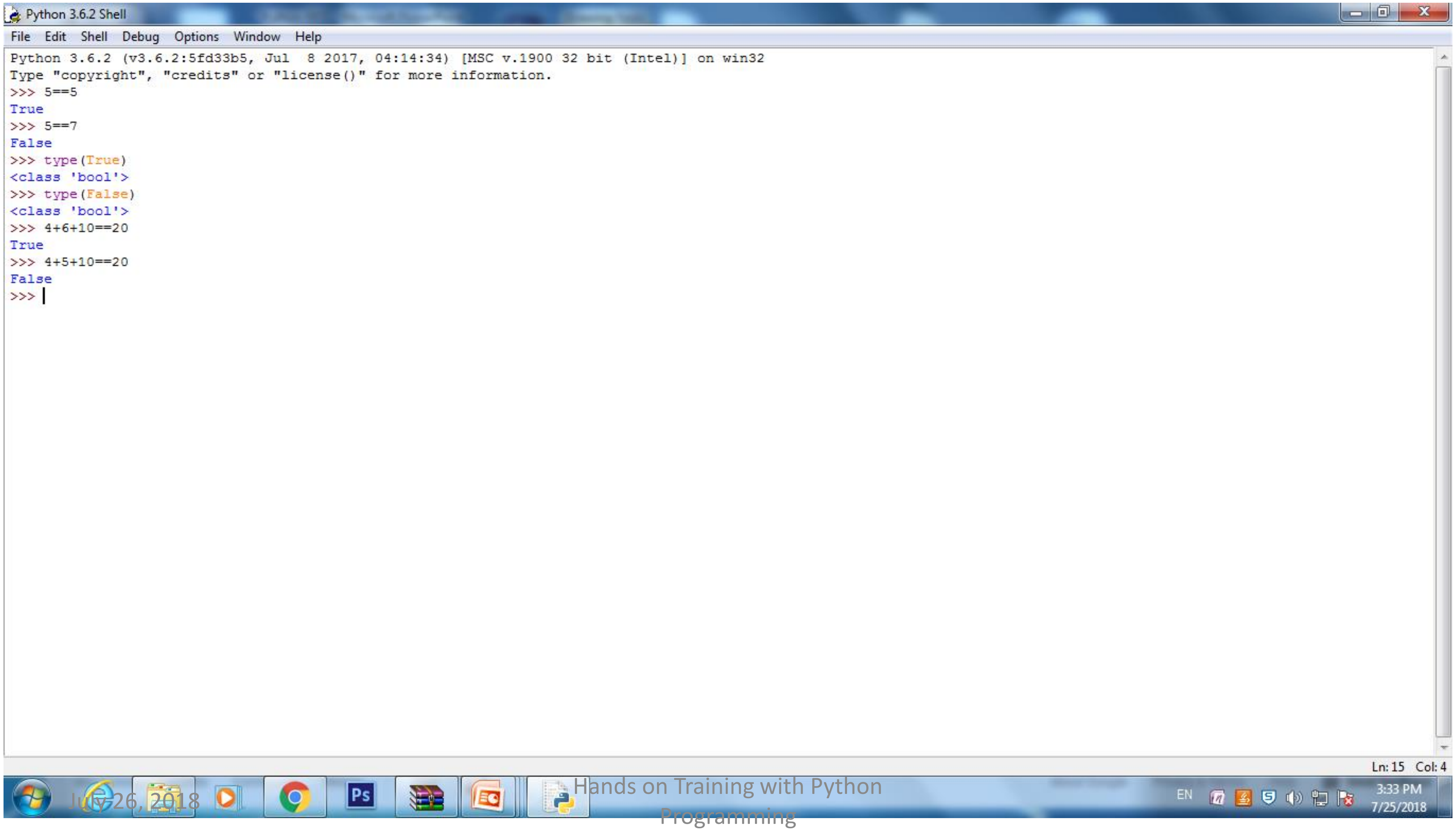
The concatenation and repetition operations on strings using the operators like + and *.



```
Python 3.6.2 (v3.6.2:5fd33b5, Jul  8 2017, 04:14:34) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> m1="Welcome"
>>> m2="To"
>>> m3="All"
>>> m1+m2+m2
'WelcomeToTo'
>>> m1+m2+m3
'WelcomeToAll'
>>> m1*3
'WelcomeWelcomeWelcome'
>>> "IT"*5
'ITITITITIT'
>>>
```

Boolean Values

Two types of boolean values – true or false.



```
Python 3.6.2 (v3.6.2:5fd33b5, Jul  8 2017, 04:14:34) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> 5==5
True
>>> 5==7
False
>>> type(True)
<class 'bool'>
>>> type(False)
<class 'bool'>
>>> 4+6+10==20
True
>>> 4+5+10==20
False
>>> 
```

Operators

1. Arithmetic Operators:

Operator	Description	Example	Output
+	Addition: Adds the operands	`>>> print(a + b)`	300
-	Subtraction: Subtracts operand on the right from the operand on the left of the operator	`>>> print(a – b)`	-100
*	Multiplication: Multiplies the operands	`>>> print(a * b)`	20000
/	Division: Divides operand on the left side of the operator with the operand on its right. The division operator returns the quotient.	`>>> print(b / a)`	2.0
%	Modulus: Divides operand on the left side of the operator with the operand on its right. The modulus operator returns the remainder.	`>>> print(b % a)`	0

//	Floor Division: Divides the operands and returns the quotient. It also removes the digits after the decimal point. If one of the operands is negative, the result is floored (i.e.,rounded away from zero towards negative infinity).	`>>> print(12//5)`
		`>>> print( 12.0//5.0)`
		`>>> print(-19//5)`
		`>>> print(-20.0//3)`
			2
			2.0
			
			-4
			-7.0

**	Exponent: Performs exponential calculation, that is, raises operand on the right side to the operand on the left of the operator.	`>>> print(a**b)`	100^{200}

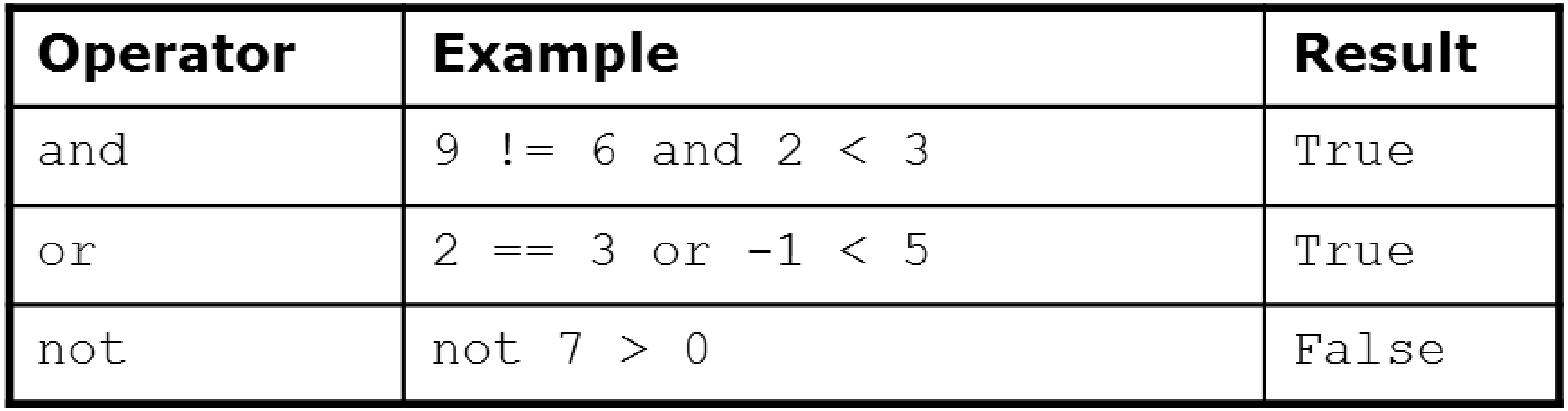
Operators

2. Relational Operators:

Operator	Meaning	Example	Result
`==`	equals	`1 + 1 == 2`	`True`
`!=`	does not equal	`3.2 != 2.5`	`True`
`<`	less than	`10 < 5`	`False`
`>`	greater than	`10 > 5`	`True`
`<=`	less than or equal to	`126 <= 100`	`False`
`>=`	greater than or equal to	`5.0 >= 5.0`	`True`

Operators

3. Logical Operators:



Operator	Example	Result
and	9 != 6 and 2 < 3	True
or	2 == 3 or -1 < 5	True
not	not 7 > 0	False

Operators

4. Bitwise Operators:

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

Operator	Meaning	Example
&	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)

Operators

5. Assignment Operators:

Operator	Example	Equivatent to
=	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

Special Operators

1. Identity operators: is and is not are the identity operators in Python. They are used to check if two values are located on the same part of the memory.

Operator	Meaning	Example
is	True if the operands are identical (refer to the same object)	x is True
is not	True if the operands are not identical (do not refer to the same object)	x is not True

```
x1 = 5
y1 = 5
x2 = 'Hello‘
y2 = 'Hello‘
print(x1 is not y1)
Output: False
print(x2 is y2)
Output: True
```

Special Operators

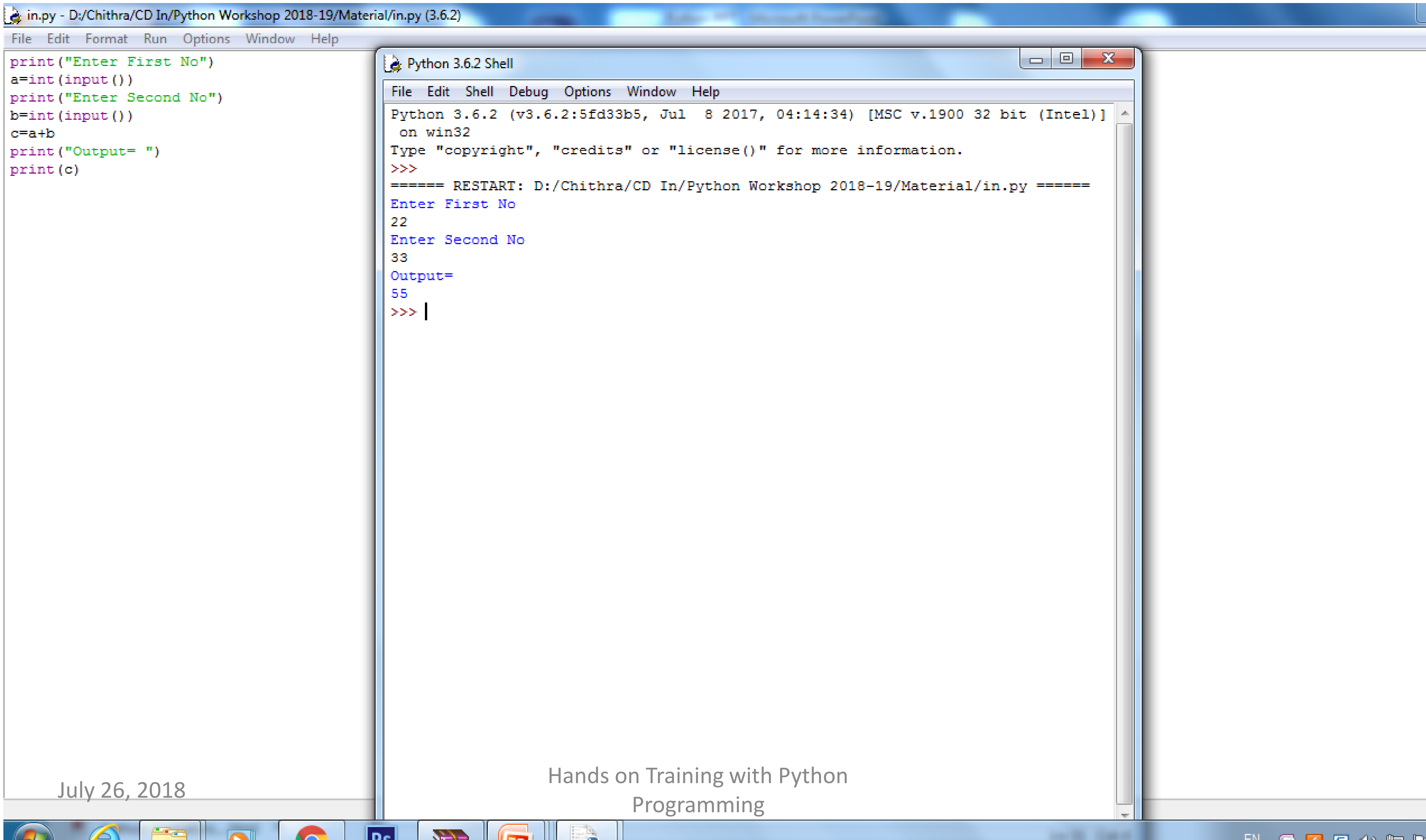
2. Membership operators: in and not in are the membership operators in Python. They are used to test whether a value or variable is found in a sequence.

Operator	Meaning	Example
in	True if value/variable is found in the sequence	5 in x
not in	True if value/variable is not found in the sequence	5 not in x

```
x1 = 'Hello‘
Print(‘H’ is not x1)
Output: False
Print(‘e’ is x1)
Output: True
```

Input and Output

The function input()

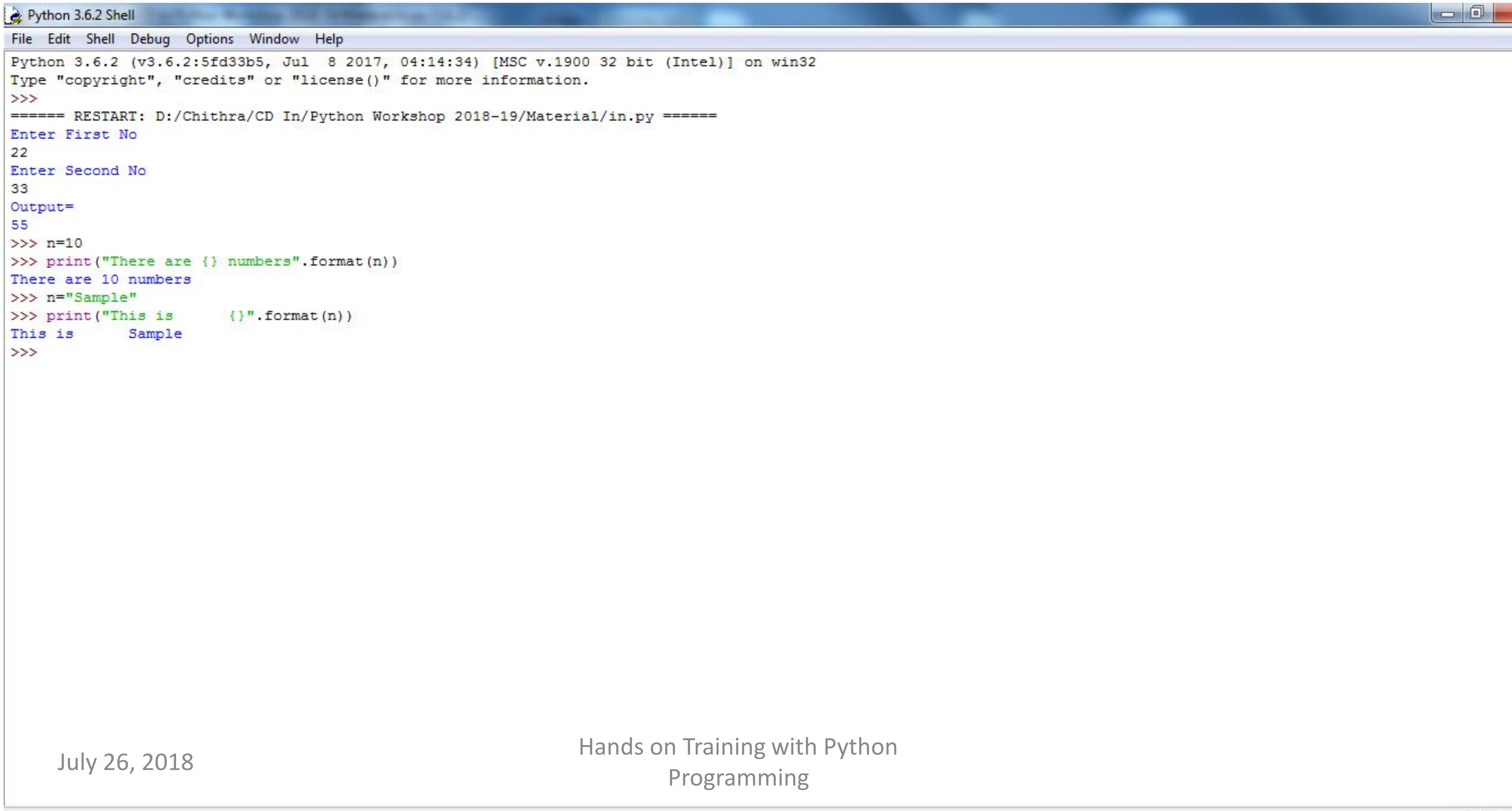

in.py - D:/Chithra/CD In/Python Workshop 2018-19/Material/in.py (3.6.2)
File Edit Format Run Options Window Help
Python 3.6.2 Shell
File Edit Shell Debug Options Window Help

```
print("Enter First No")
a=int(input())
print("Enter Second No")
b=int(input())
c=a+b
print("Output= ")
print(c)
```

```
Python 3.6.2 (v3.6.2:5fd33b5, Jul  8 2017, 04:14:34) [MSC v.1900 32 bit (Intel)]
 on win32
Type "copyright", "credits" or "license()" for more information.
>>>
====== RESTART: D:/Chithra/CD In/Python Workshop 2018-19/Material/in.py ======
Enter First No
22
Enter Second No
33
Output=
55
>>>
```

How to Display Output on Console using Format?

Using .format the data can be displayed on the console. For that purpose { } and .format is used.

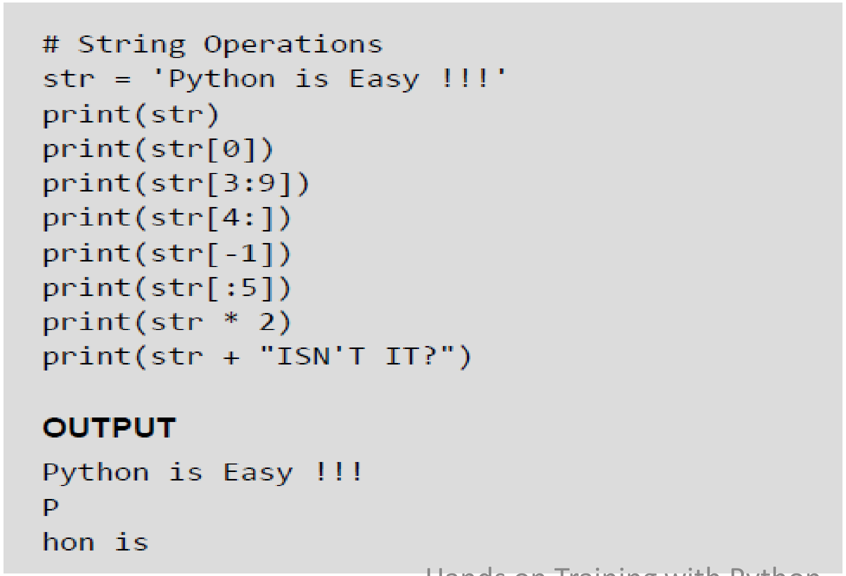


```
Python 3.6.2 (v3.6.2:5fd33b5, Jul  8 2017, 04:14:34) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
====== RESTART: D:/Chithra/CD In/Python Workshop 2018-19/Material/in.py ======
Enter First No
22
Enter Second No
33
Output=
55
>>> n=10
>>> print("There are {} numbers".format(n))
There are 10 numbers
>>> n="Sample"
>>> print("This is      {}".format(n))
This is      Sample
>>>
```

Slice Operations on Strings

You can extract subsets of strings by using the slice operator ([ ] and [:]). You need to specify index or the range of index of characters to be extracted. The index of the first character is 0 and the index of the last character is n-1, where n is the number of characters in the string.

If you want to extract characters starting from the end of the string, then you must specify the index as a negative number. For example, the index of the last character is -1.



```
# String Operations
str = 'Python is Easy !!!'
print(str)
print(str[0])
print(str[3:9])
print(str[4:])
print(str[-1])
print(str[:5])
print(str * 2)
print(str + "ISN'T IT?")
```

OUTPUT

```
Python is Easy !!!
P
hon is
```

Sample Programs

- Printing the Pattern.
- Circulate the Values.
- Test for Leap Year.
- Distance between Two Points.

Control Structures

Conditional Statement – if/else

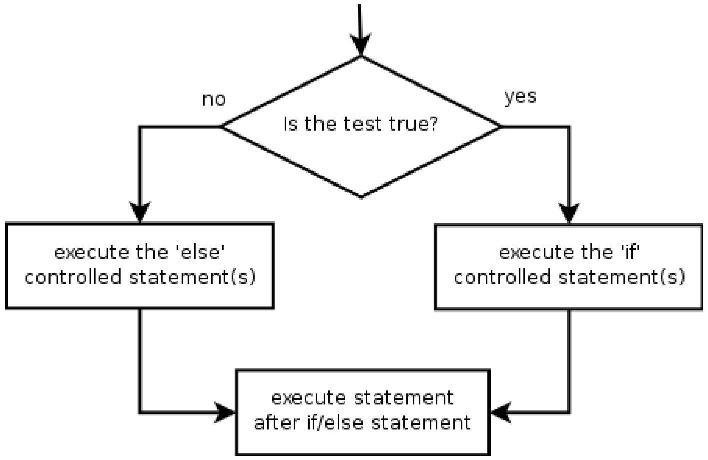
if/else statement: Executes one block of statements if a certain condition is True, and a second block of statements if it is False.

Syntax:

```
if condition:
    statements
else:
    statements
```

Example:

```
gpa = 1.4
if gpa > 2.0:
    print ("Welcome to Mars University!“)
else:
    print ("Your application is denied.“)
```



no
Is the test true?
yes
execute the 'else' controlled statement(s)
execute the 'if' controlled statement(s)
execute statement after if/else statement

Multiple conditions can be chained with elif ("else if"):

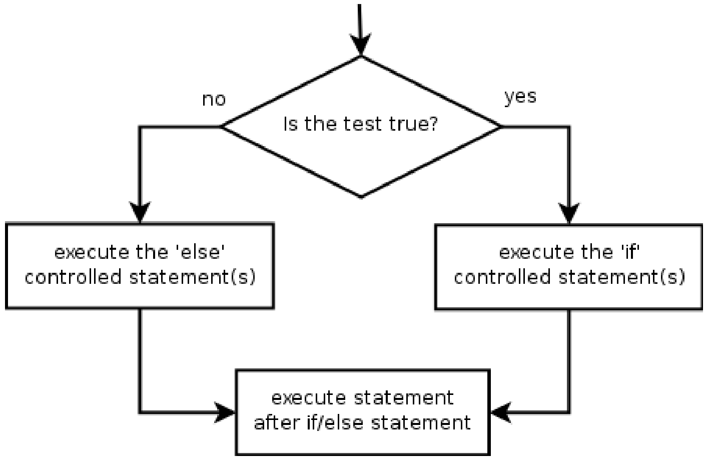
```
if condition:
    statements
elif condition:
    statements
else:
    statements
```

Iteration – while

- **while loop**: Executes a group of statements as long as a condition is True.
  - good for indefinite loops (repeat an unknown number of times)

- Syntax:

```
while condition:
    statements
```



no
yes
Is the test true?
execute the 'else' controlled statement(s)
execute the 'if' controlled statement(s)
execute statement after if/else statement

- Example:

```
number = 1
while number < 200:
    print (number)
    number = number * 2
```

  - Output:

```
1 2 4 8 16 32 64 128
```

Iteration – for

- **for loop**: Repeats a set of statements over a group of values.
  - Syntax:

    ```
    for variableName in groupofValues:
        statements
    ```

    - We indent the statements to be repeated with tabs or spaces.
    - variableName gives a name to each value, so you can refer to it in the statements.
    - groupOfValues can be a range of integers, specified with the range function.
  - Example:

    ```
    for x in range(1, 6):
        print (x, "squared is", x * x)
    ```

    Output:

    ```
    1 squared is 1
    2 squared is 4
    3 squared is 9
    4 squared is 16
    5 squared is 25
    ```

Iteration – for - range

The range function specifies a range of integers:

range(start, stop) - the integers between start (inclusive) and stop (exclusive)

It can also accept a third value specifying the change between values.

range(start, stop, step) - the integers between start (inclusive) and stop (exclusive) by step

Example:

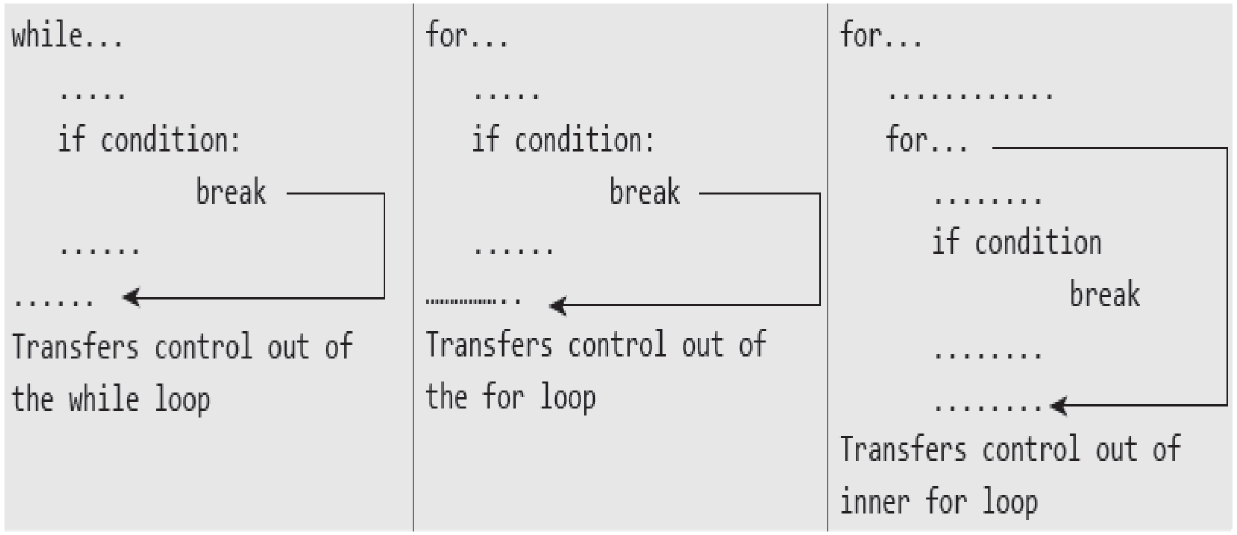
```
for x in range(5, 0, -1):
    print (x)
print (“End“)
```

Output:

```
5
4
3
2
1
End
```

Break Statement

The *break* statement is used to terminate the execution of the nearest enclosing loop in which it appears. The break statement is widely used with for loop and while loop. When compiler encounters a break statement, the control passes to the statement that follows the loop in which the break statement appears.



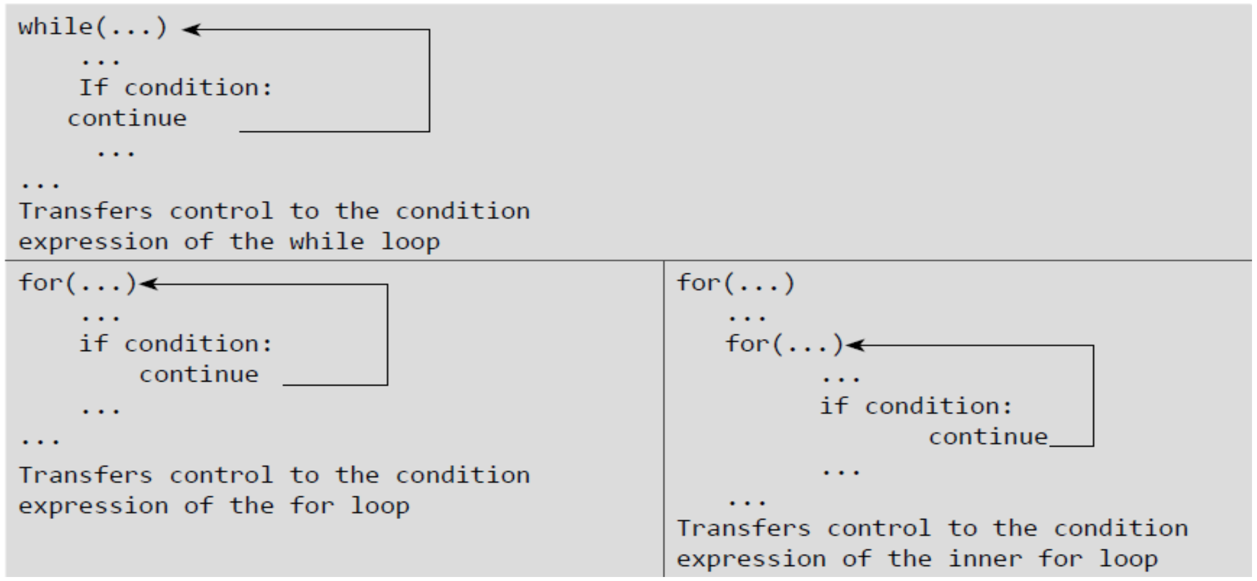
```
i = 1
while i <= 10:
        print(i, end=" ")
        if i==5:
                break
        i = i+1
print("\n Done")
```

OUTPUT

```
1 2 3 4 5
Done
```

Continue Statement

Like the break statement, the continue statement can only appear in the body of a loop. When the compiler encounters a continue statement then the rest of the statements in the loop are skipped and the control is unconditionally transferred to the loop-continuation portion of the nearest enclosing loop.



```
while(...)
    ...
    If condition:
   continue
     ...
...
Transfers control to the condition
expression of the while loop
```

```
for(...)
    ...
    if condition:
        continue
    ...
...
Transfers control to the condition
expression of the for loop
```

```
for(...)
   ...
   for(...)
         ...
         if condition:
                continue
         ...
   ...
Transfers control to the condition
expression of the inner for loop
```

```
for i in range(1,11):
    if(i==5):
        continue
    print(i, end=" ")
print("\n Done")
```

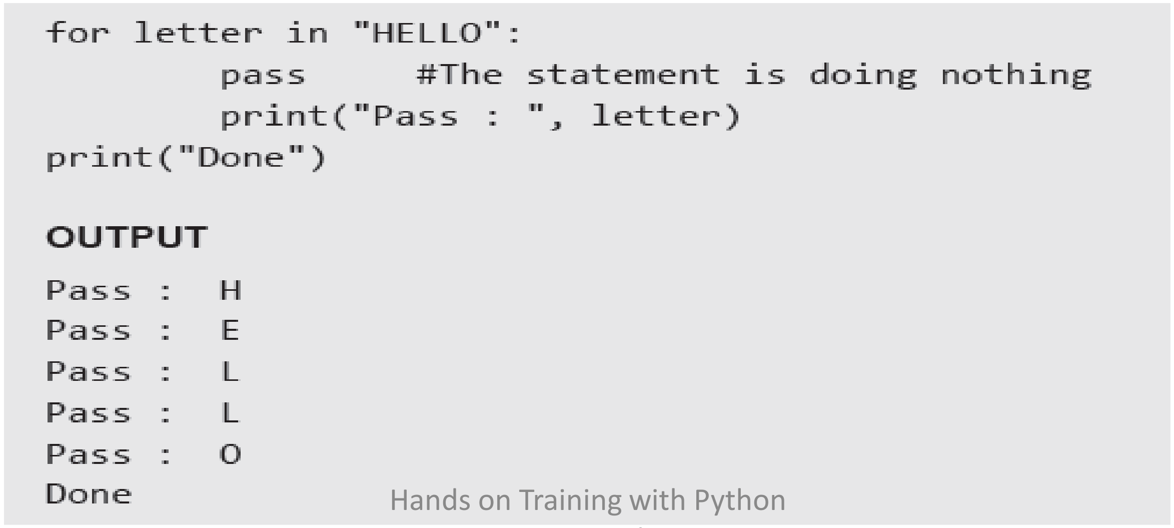
OUTPUT

```
1 2 3 4 6 7 8 9 10
Done
```

Pass Statement

Pass statement is used when a statement is required syntactically but no command or code has to be executed. It specified a *null* operation or simply No Operation (NOP) statement. Nothing happens when the pass statement is executed.

Difference between comment and pass statements In Python programming, pass is a null statement. The difference between a comment and pass statement is that while the interpreter ignores a comment entirely, pass is not ignored. Comment is not executed but pass statement is executed but nothing happens.



```
for letter in "HELLO":
        pass      #The statement is doing nothing
        print("Pass : ", letter)
print("Done")
```

OUTPUT

```
Pass :  H
Pass :  E
Pass :  L
Pass :  L
Pass :  O
Done
```

The Else Statement Used With Loops

Unlike C and C++, in Python you can have the *else* statement associated with a loop statements. If the else statement is used with a *for* loop, the *else* statement is executed when the loop has completed iterating. But when used with the *while* loop, the *else* statement is executed when the condition becomes false.

```
for letter in "HELLO":
       print(letter, end=" ")
else:
        print("Done")
```

OUTPUT

```
H E L L O Done
```

```
i = 1
while(i<0):
        print(i)
        i = i - 1
else:
        print(i, "is not negative so
loop did not execute")
```

OUTPUT

```
1 is not negative so loop did not execute
```