



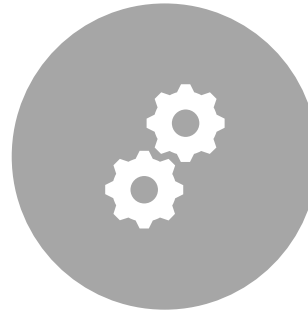
1



Python Quick Tips

Data Types

Future Videos



MAKING A CUSTOM
NODE



MAKING A CUSTOM
PACKAGE



PYTHON
FUNDAMENTALS



PYTHON IN
DYNAMO (SERIES)



TLDR
github

<https://github.com/aussieBIMguru>



Python Quick Tips #1

Data Types and variables

Data Types



In Python, Data is referred to as **objects**

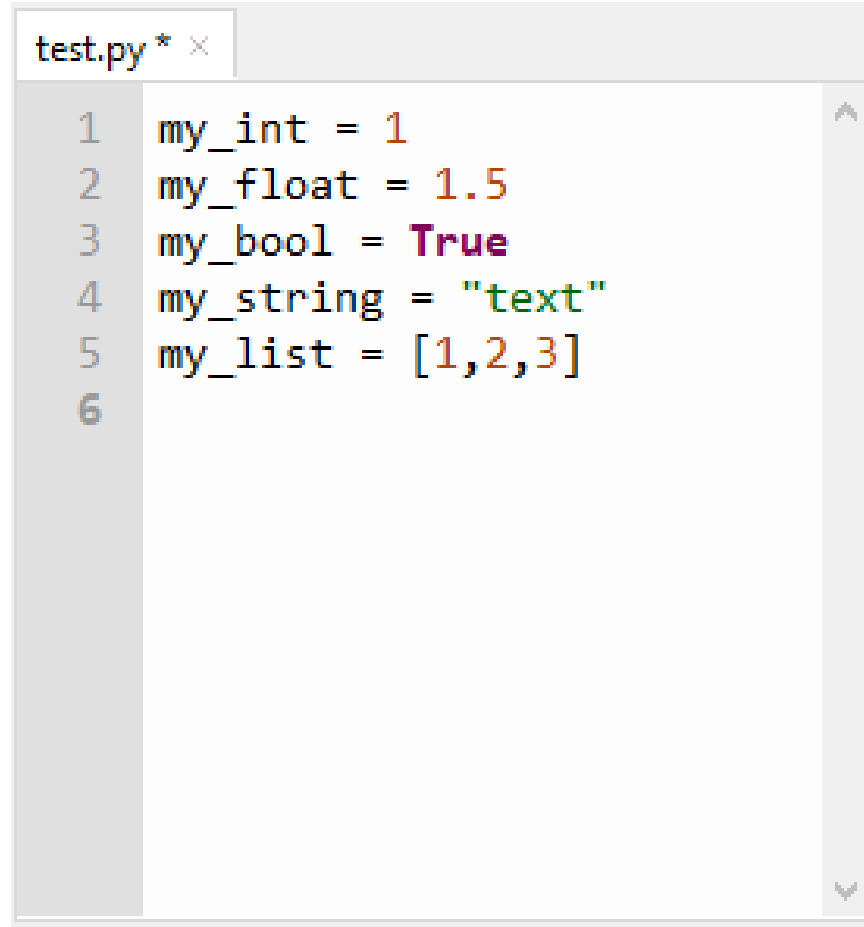
Data Types

Data comes in **types**

Integers	int	1
Floats	float	1.5
Booleans	bool	True
Strings	str	"text"
Lists	list	[a,b,c]
Others	varies	varies

Declaring Variables

Syntax
Variable = value



```
test.py * x
1 my_int = 1
2 my_float = 1.5
3 my_bool = True
4 my_string = "text"
5 my_list = [1,2,3]
6
```

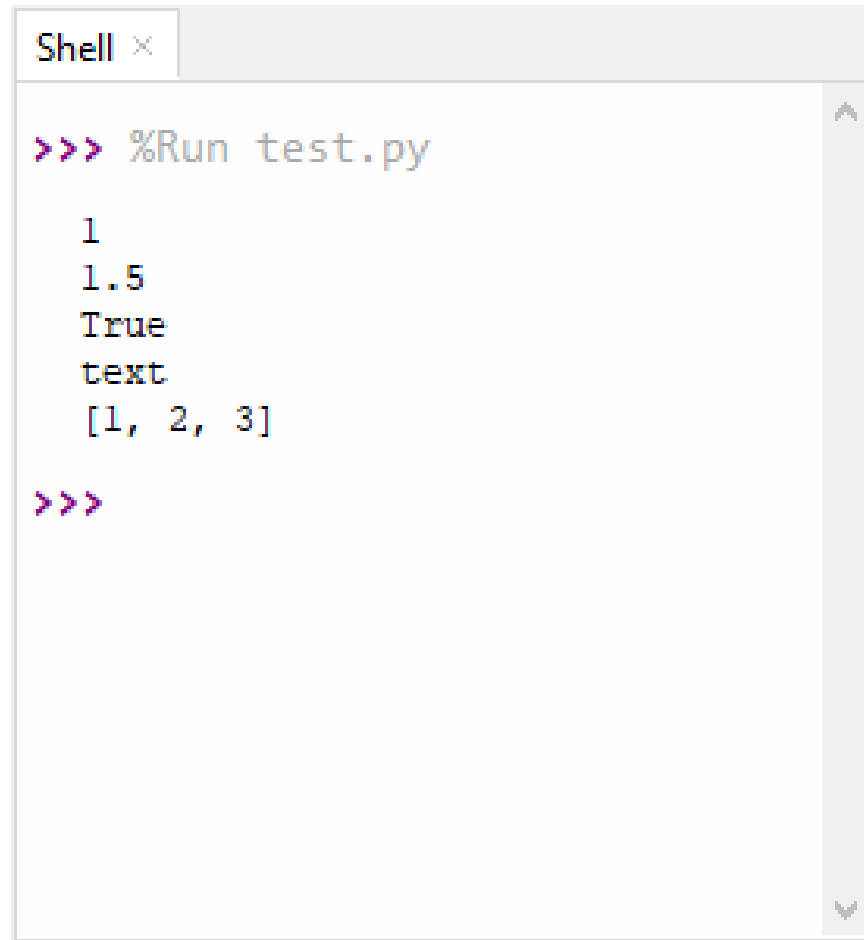
Calling/Printing Variables

Function
print(variable)

```
test.py ×  
1 my_int = 1  
2 my_float = 1.5  
3 my_bool = True  
4 my_string = "text"  
5 my_list = [1,2,3]  
6  
7 print(my_int)  
8 print(my_float)  
9 print(my_bool)  
10 print(my_string)  
11 print(my_list)
```


Calling/Printing Variables

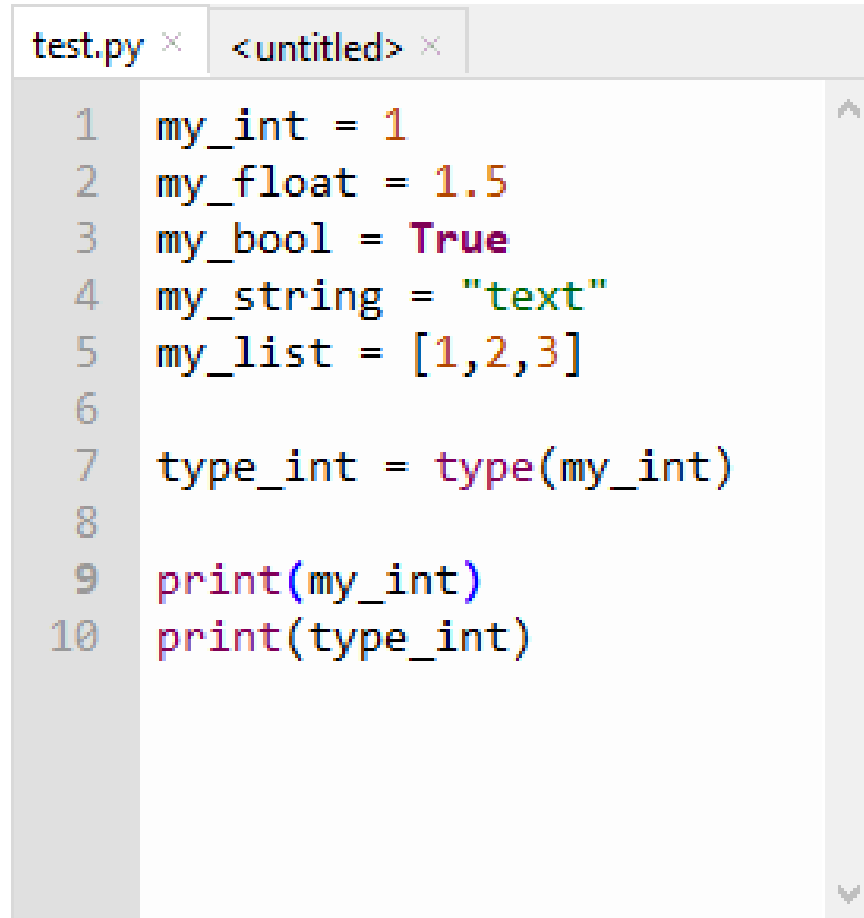
Result
print(variable)

A screenshot of a Jupyter Notebook shell window. The window has a title bar with a tab labeled "Shell" and a close button. The main area contains the output of a Python script. The first line is a prompt ">>>" followed by the command "%Run test.py". The subsequent lines show the output of the script: "1", "1.5", "True", "text", and "[1, 2, 3]". The prompt ">>>" appears again at the bottom. A vertical scrollbar is visible on the right side of the output area.

```
Shell ×  
  
>>> %Run test.py  
  
1  
1.5  
True  
text  
[1, 2, 3]  
  
>>>
```

Checking Data Types

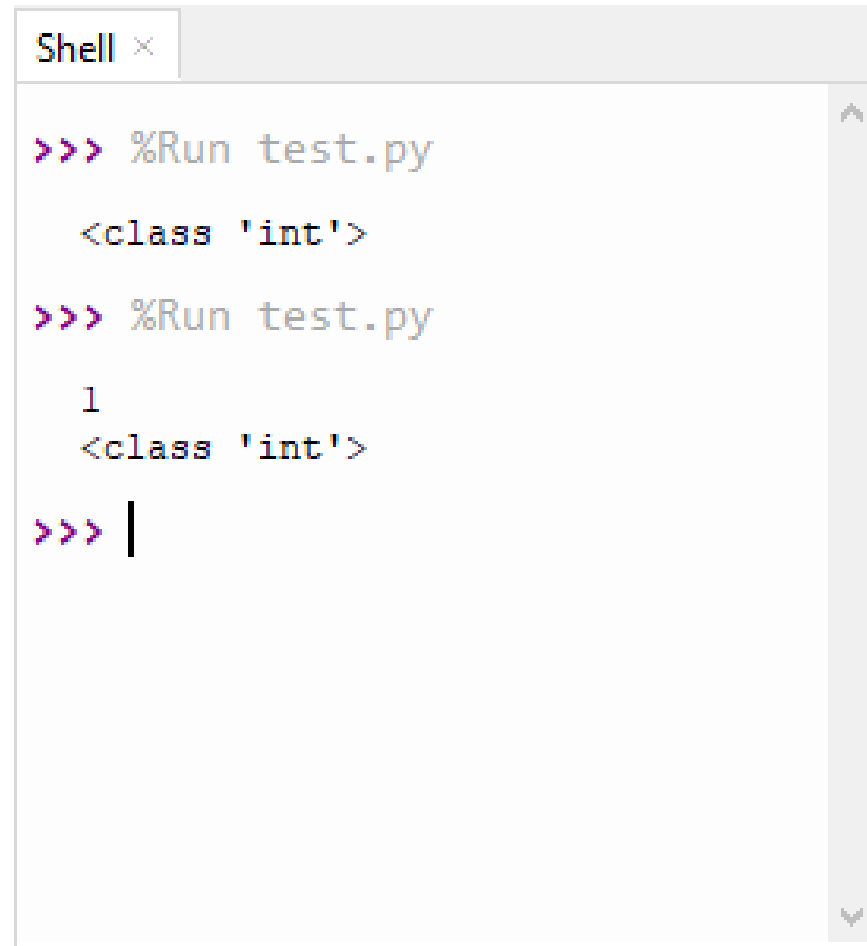
Function Type(variable)



```
test.py × <untitled> ×  
1 my_int = 1  
2 my_float = 1.5  
3 my_bool = True  
4 my_string = "text"  
5 my_list = [1,2,3]  
6  
7 type_int = type(my_int)  
8  
9 print(my_int)  
10 print(type_int)
```

Checking Data Types

Function
Type(variable)



```
Shell x
```

```
>>> %Run test.py
<class 'int'>

>>> %Run test.py
1
<class 'int'>

>>> |
```

Converting Data Types



In Python, **always** deal
with the right data **type**

Converting Data Types

Function(s)
class(variable)

```
test.py × <untitled> ×  
1 my_int = 1  
2  
3 my_float = float(my_int)  
4 my_bool = bool(my_int)  
5 my_string = str(my_int)  
6  
7 print(my_int)  
8 print(my_float)  
9 print(my_bool)  
10 print(my_string)
```

Converting Data Types

Function(s)
class(variable)

Shell ×

```
>>> %Run test.py
```

```
1
```

```
1.0
```

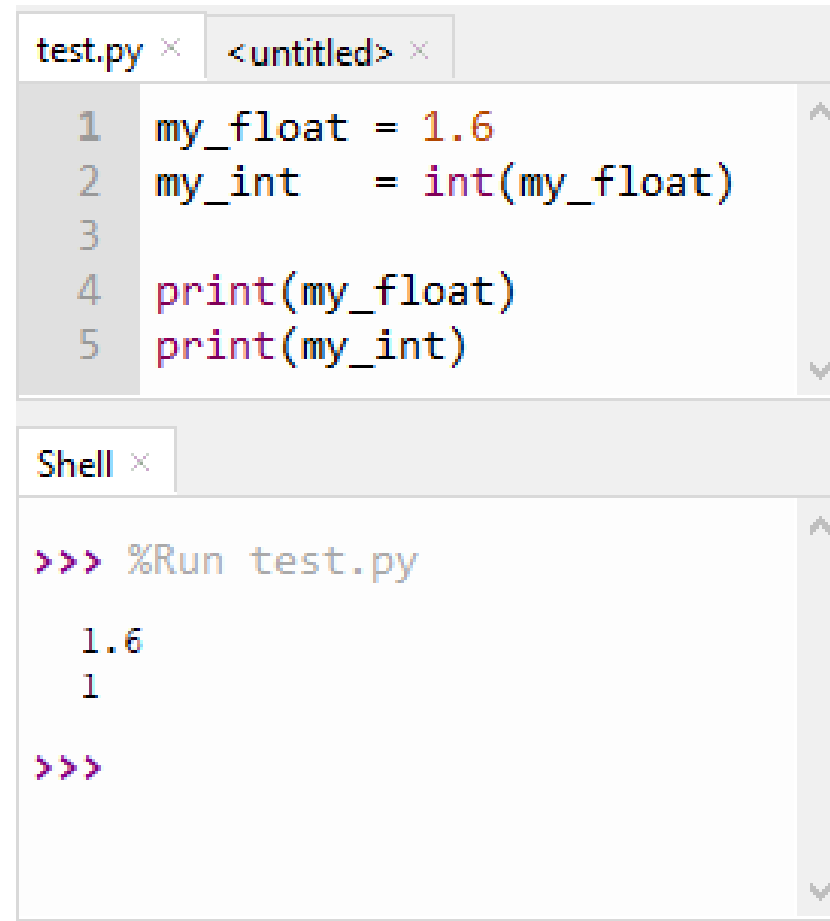
```
True
```

```
1
```

```
>>> |
```

Converting Data Types

Not all data types are
cleanly converted...



The screenshot shows a Python IDE with two panels. The top panel, titled 'test.py' and '<untitled>', contains the following code:

```
1 my_float = 1.6
2 my_int   = int(my_float)
3
4 print(my_float)
5 print(my_int)
```

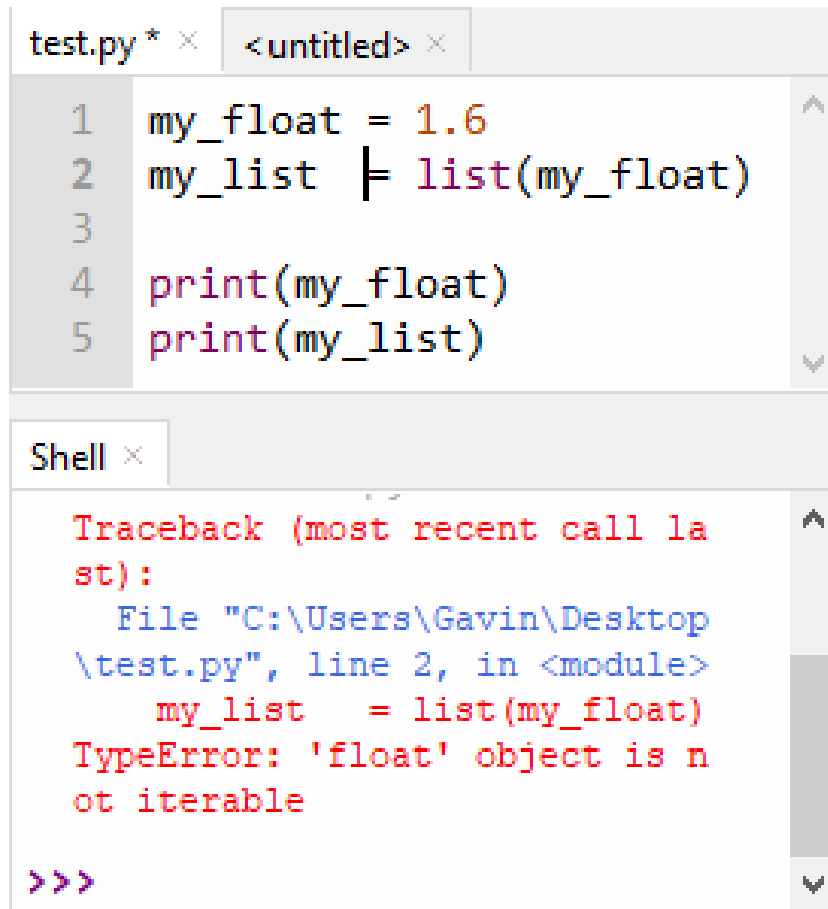
The bottom panel, titled 'Shell', shows the execution of the script:

```
>>> %Run test.py
1.6
1
>>>
```

The output demonstrates that the float value 1.6 is printed as is, while the integer value 1 is printed as an integer, illustrating that not all data types are converted cleanly.

Converting Data Types

Not all data types are
able to be converted...



The screenshot shows a Python IDE with two tabs: 'test.py *' and '<untitled>'. The 'test.py' tab is active and contains the following code:

```
1 my_float = 1.6
2 my_list = list(my_float)
3
4 print(my_float)
5 print(my_list)
```

Below the code editor is a 'Shell' tab. It displays a traceback error message:

```
Traceback (most recent call last):
  File "C:\Users\Gavin\Desktop\test.py", line 2, in <module>
    my_list = list(my_float)
TypeError: 'float' object is not iterable

>>>
```

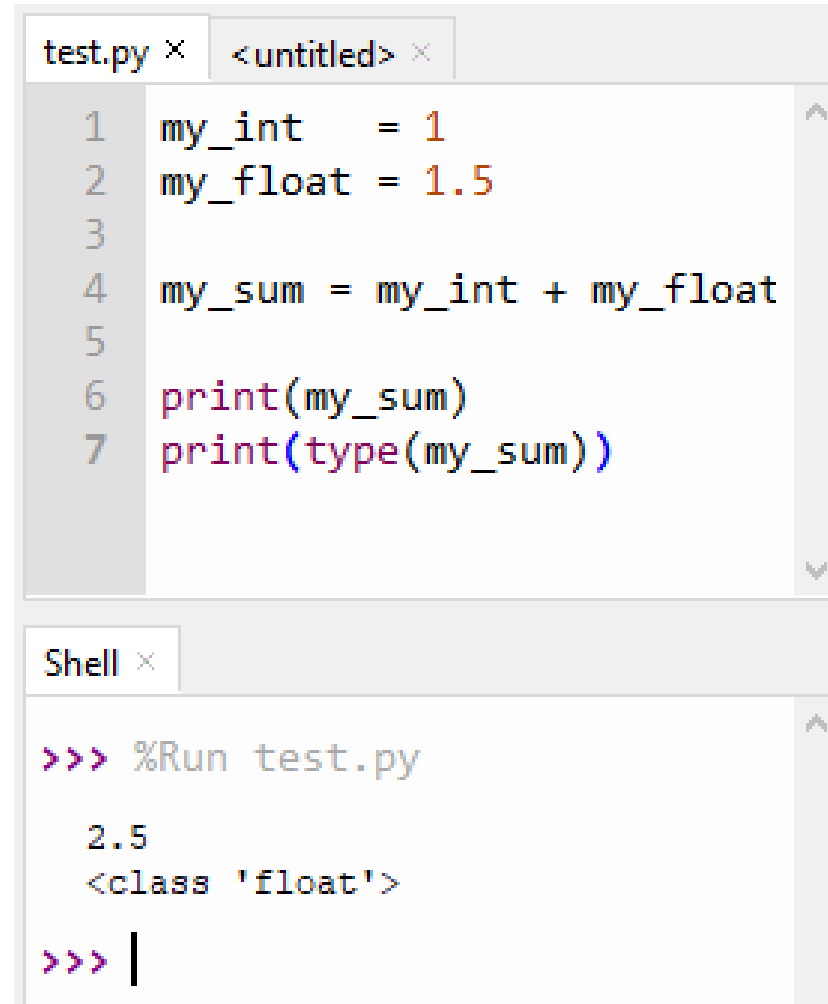

Arithmetic

Operators

Add	+
Subtract	-
Multiply	*
Divide	/
Exponential	**
Remainder	&

Arithmetic

Combining variables



The screenshot shows a Python IDE with two panels. The top panel, titled 'test.py ×' and '<untitled> ×', contains a Python script with seven lines of code. The bottom panel, titled 'Shell ×', shows the output of running the script. The script defines two variables, 'my_int' and 'my_float', adds them to 'my_sum', and prints the result and its type. The shell output shows the result '2.5' and its type '<class 'float'>'. The prompt '>>>' is visible at the end of the output line.

```
test.py × <untitled> ×  
1 my_int = 1  
2 my_float = 1.5  
3  
4 my_sum = my_int + my_float  
5  
6 print(my_sum)  
7 print(type(my_sum))  
  
Shell ×  
  
>>> %Run test.py  
2.5  
<class 'float'>  
>>> |
```

Arithmetic

Some operators do special things



The image shows a code editor window with two tabs: 'test.py' and '<untitled>'. The 'test.py' tab is active and contains the following Python code:

```
1 my_str1 = "hello"  
2 my_str2 = "world!"  
3  
4 message = my_str1 + " " + my_str2  
5  
6 print(message)
```

Below the code editor is a 'Shell' window. It shows the command prompt running the script:

```
>>> %Run test.py  
hello world!  
>>> |
```

Logic

Operators

Equal	==
Not equal	!=
Greater than	>
Less than	<
Greater/equal	>=
Lesser/equal	<=

Logic

Conditions

And

X and Y

Or

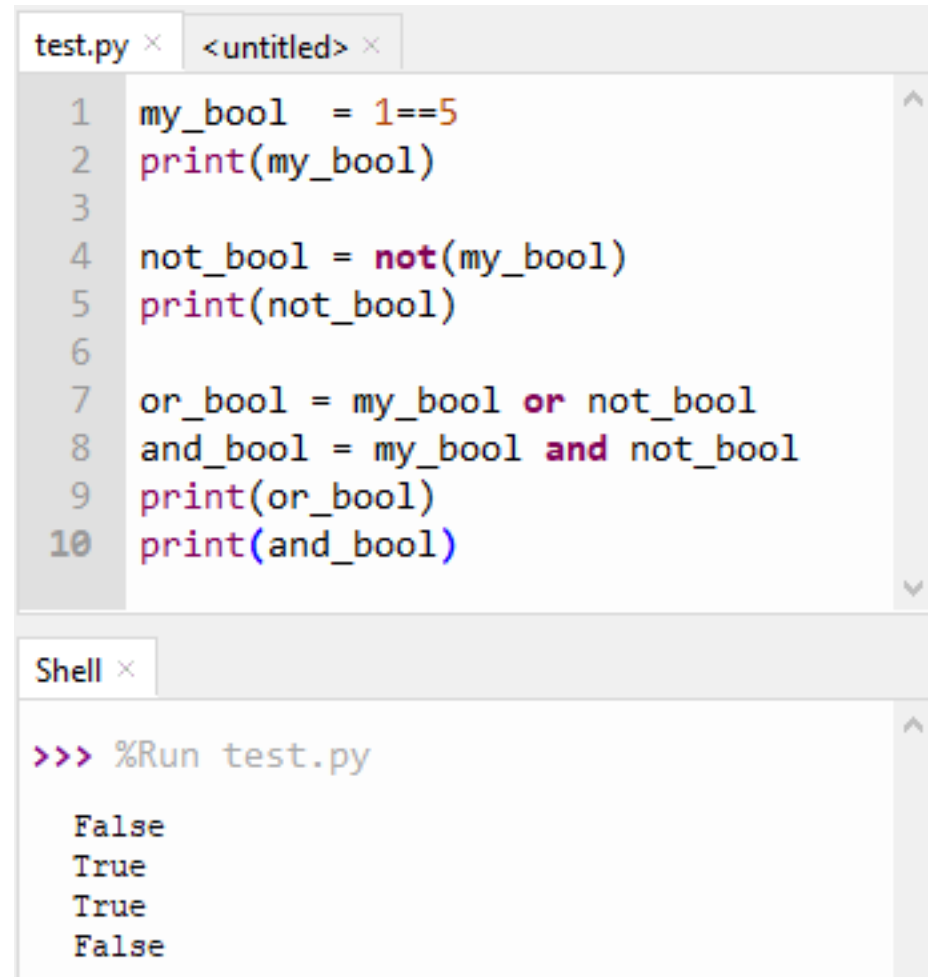
X or Y

Not

not(X)

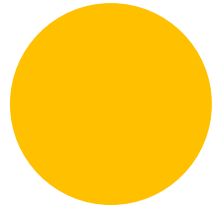
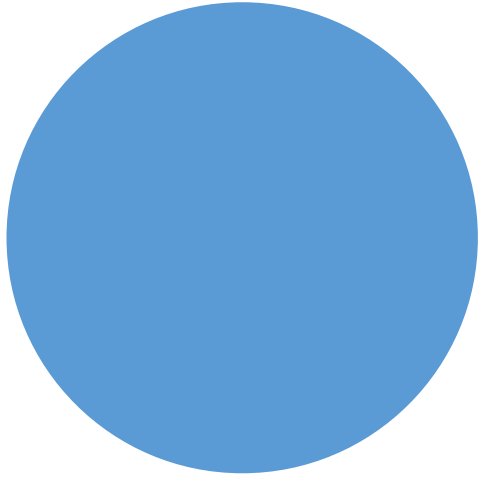
Logic

Example



The image shows a screenshot of a code editor with two tabs: 'test.py' and '<untitled>'. The 'test.py' tab is active, displaying a Python script with 10 lines of code. The code defines a boolean variable 'my_bool' as '1==5', prints its value, then defines 'not_bool' as the negation of 'my_bool', prints it, then defines 'or_bool' and 'and_bool' using logical operators, and prints both. Below the code editor is a 'Shell' tab showing the output of running the script: 'False', 'True', 'True', and 'False' on separate lines.

```
test.py × <untitled> ×  
1 my_bool = 1==5  
2 print(my_bool)  
3  
4 not_bool = not(my_bool)  
5 print(not_bool)  
6  
7 or_bool = my_bool or not_bool  
8 and_bool = my_bool and not_bool  
9 print(or_bool)  
10 print(and_bool)  
  
Shell ×  
  
>>> %Run test.py  
  
False  
True  
True  
False
```



Next on #2

Functions, Methods and Packages



1



Python Quick Tips

Data Types