Python Lab 2

Simple Coding

“Hello World” in Java

```
public class HelloWorld {
public static void main(String[] args) {
System.out.println("Hello World!");
}
}
```

“Hello World” in C++

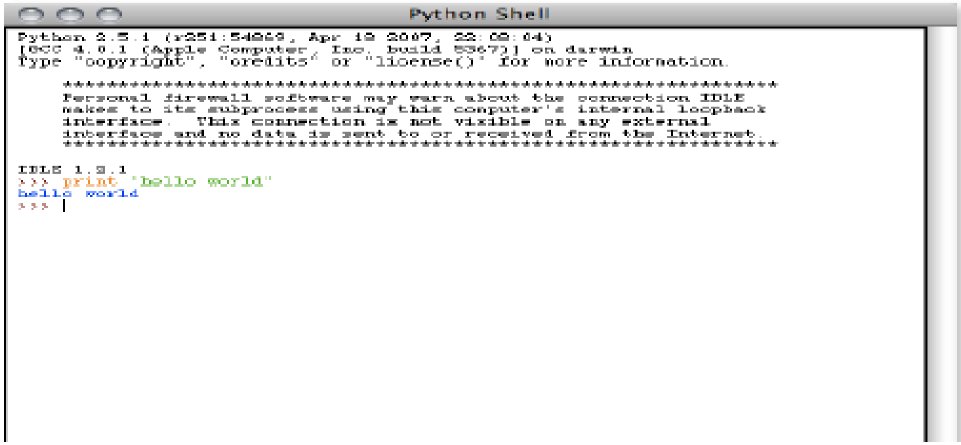
```
Double
#include <iostream>
using namespace std;
int main() {
cout << "Hello World!" << endl;
}
```

- “Hello World” in Python

  print “hello world”

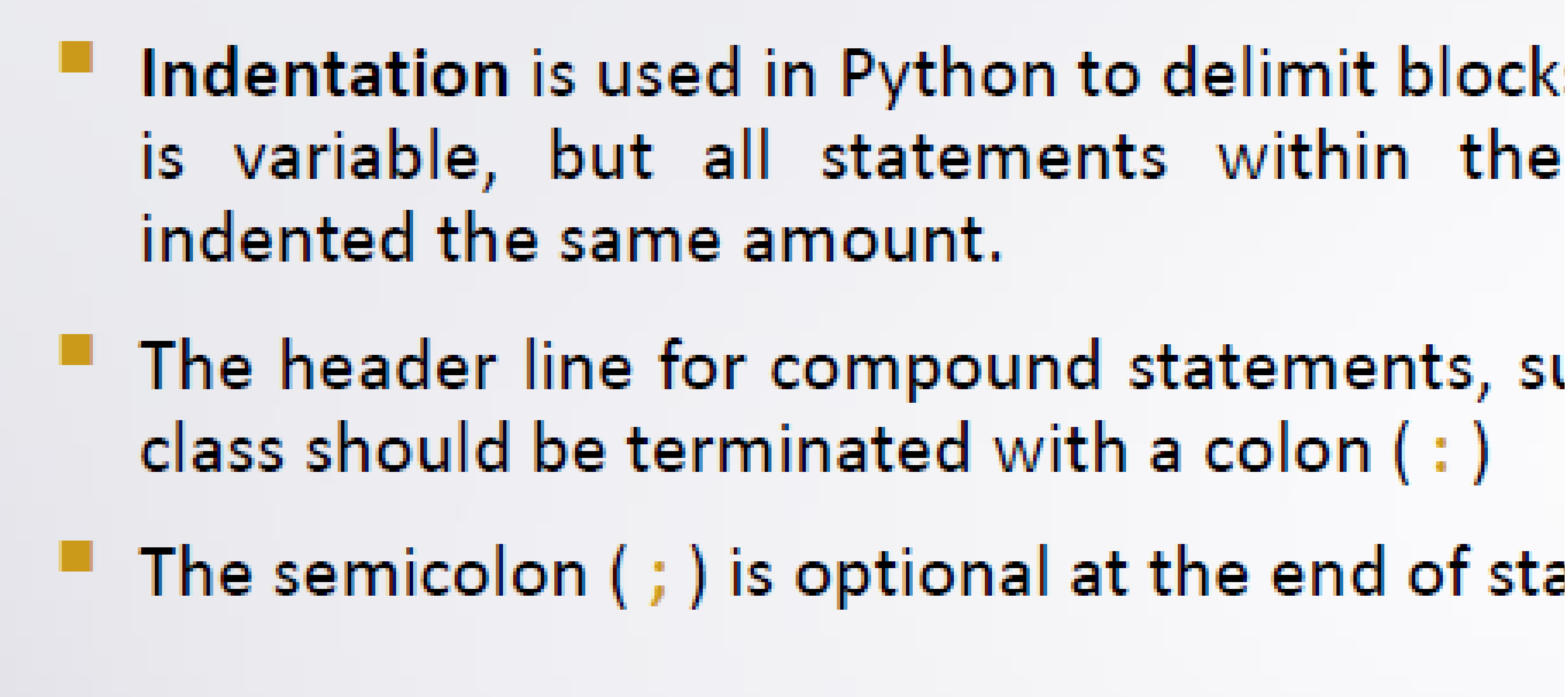
Prints hello world to standard out,

Open IDLE and try it out yourself.



```
Python 2.5.1 (r251:54863, Apr 18 2007, 22:08:04)
[GCC 4.0.1 (Apple Computer, Inc. build 5367)] on darwin
Type "copyright", "credits" or "license()" for more information.

    ****************************************************************
    Personal firewall software may warn about the connection IDLE
    makes to its subprocess using this computer's internal loopback
    interface.  This connection is not visible on any external
    interface and no data is sent to or received from the Internet.
    ****************************************************************

IDLE 1.2.1
>>> print 'hello world'
hello world
>>> 
```

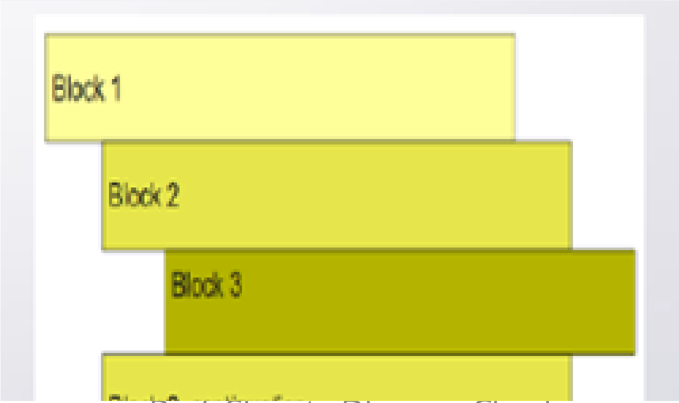
Basic Syntax



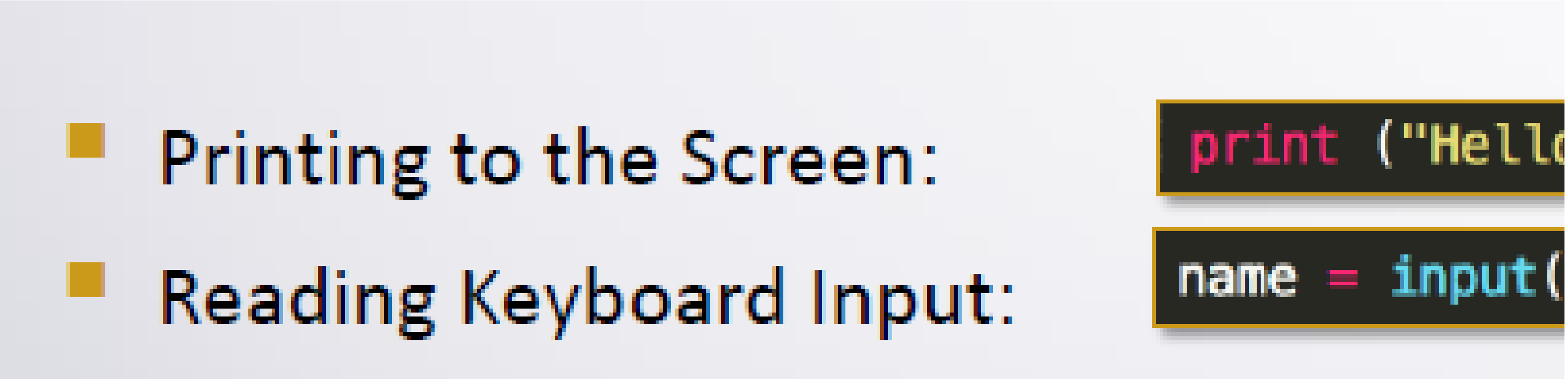
- **Indentation** is used in Python to delimit block
is variable, but all statements within the
indented the same amount.

- The header line for compound statements, su
class should be terminated with a colon (:)

- The semicolon (;) is optional at the end of sta



Basic Syntax



- Printing to the Screen:

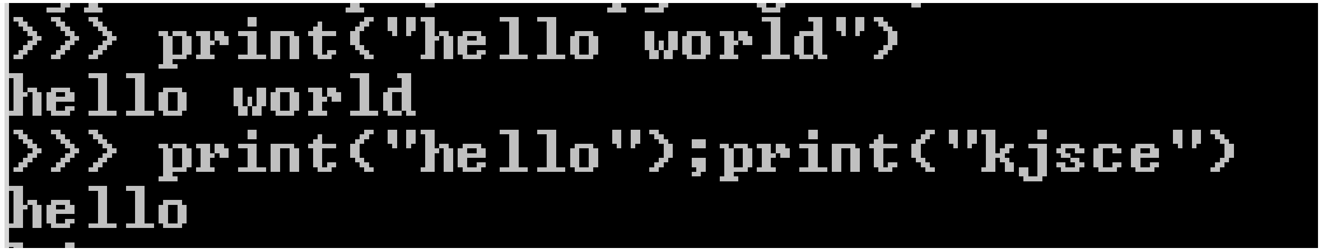
```
print ("Hell
```

- Reading Keyboard Input:

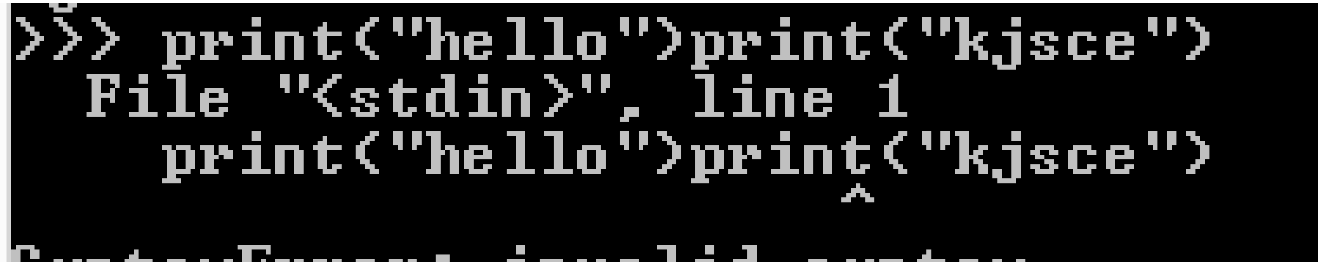
```
name = input(
```

Playing with print

- Semicolon optional at the end of the statement
- Semicolon needed for multiple statements on same line



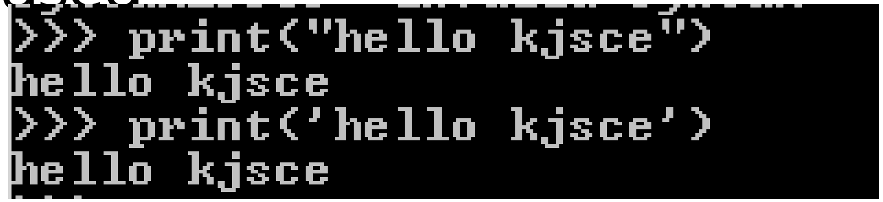
```
>>> print("hello world")
hello world
>>> print("hello");print("kjsce")
hello
```



```
>>> print("hello")print("kjsce")
  File "<stdin>", line 1
    print("hello")print("kjsce")
                      ^

SyntaxError: invalid syntax
```

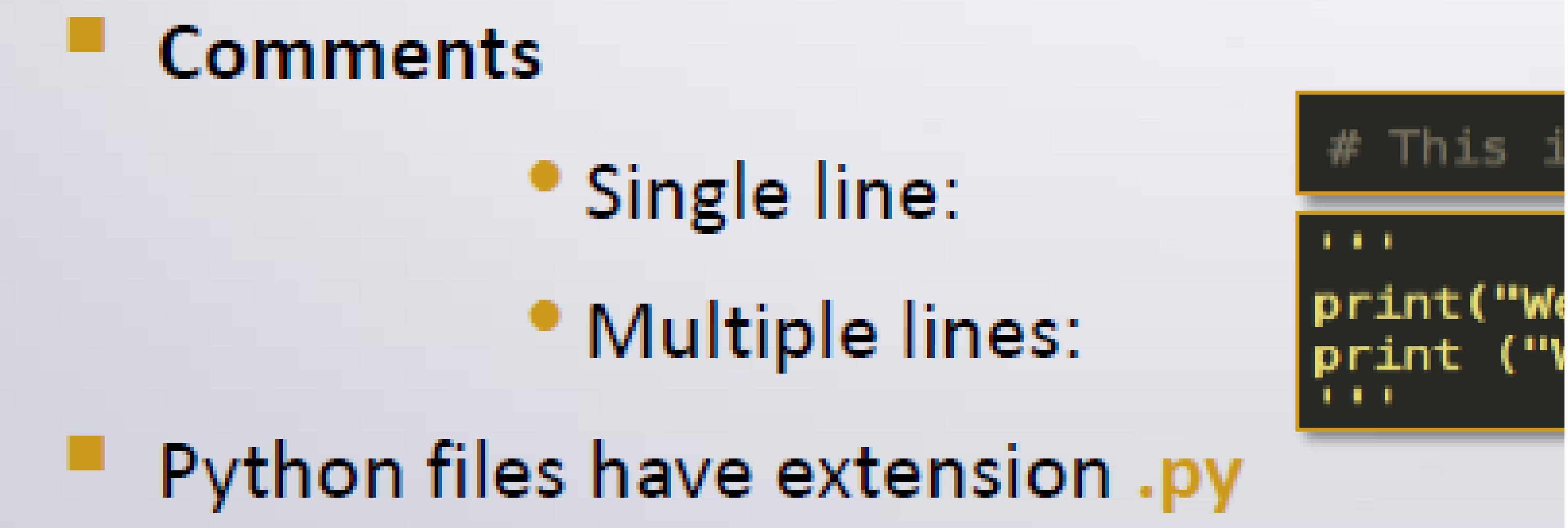
Playing with print

- For displaying text ,Both double and single quotes can be used



```
>>> print("hello kjsce")
hello kjsce
>>> print('hello kjsce')
hello kjsce
```

Basic Syntax



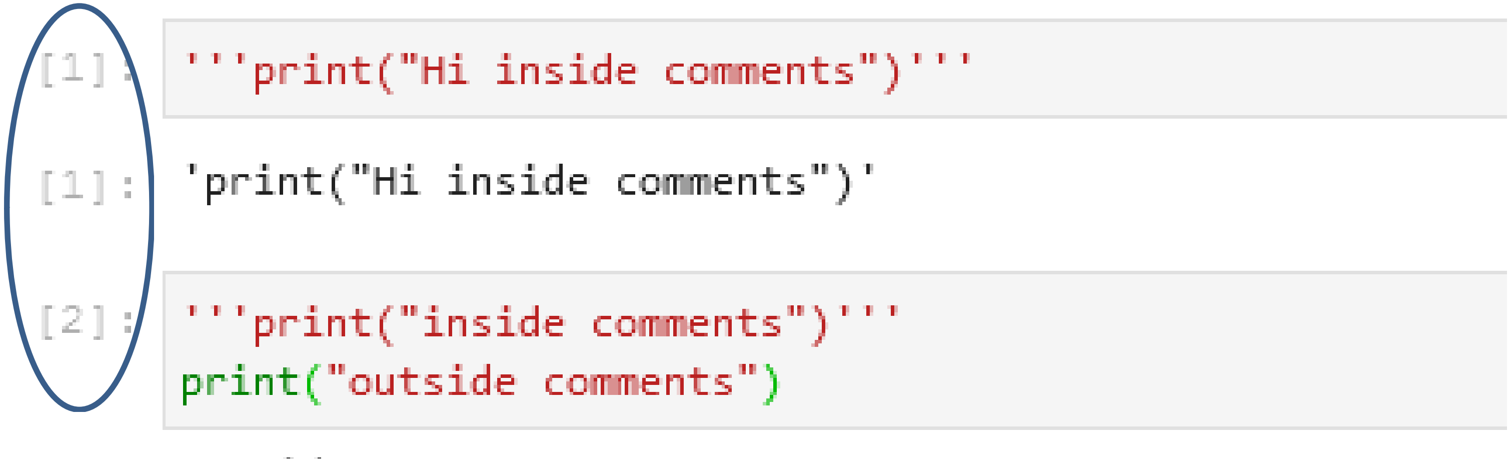
- Comments
  - Single line:
  - Multiple lines:
- Python files have extension .py

```
# This i
```

```
'''
print("We
print ("W
'''
```

Comments

- With code, Comments will not be displayed
- Without Code, comments will be displayed
- No is assigned to the LOC after execution
- |



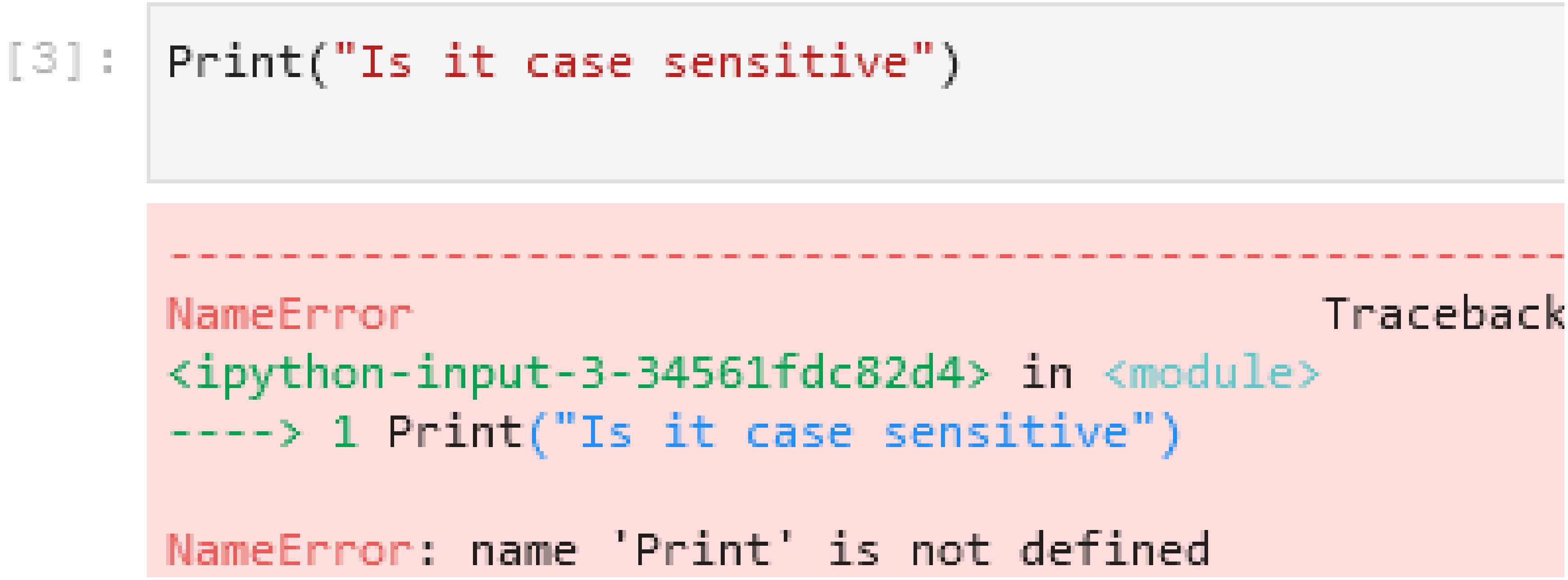
```
[1]: '''print("Hi inside comments")'''

[1]: 'print("Hi inside comments")'

[2]: '''print("inside comments")'''
     print("outside comments")
```

Case Sensitive?

- Try it?

Case Sensitive? Try it?

- Yes, Python is Case Sensitive



```
[3]: Print("Is it case sensitive")
```

```
---------------------------------------------------
NameError                                 Traceback
<ipython-input-3-34561fdc82d4> in <module>
----> 1 Print("Is it case sensitive")

NameError: name 'Print' is not defined
```

Data types

- Python has the following data types built-in by default, in these categories:

Text Type:	str
Numeric Types:	int, float, complex
Sequence Types:	list, tuple, range
Mapping Type:	dict
Set Types:	set, frozenset
Boolean Type:	bool
Binary Types:	bytes, bytearray, memoryview

Data types

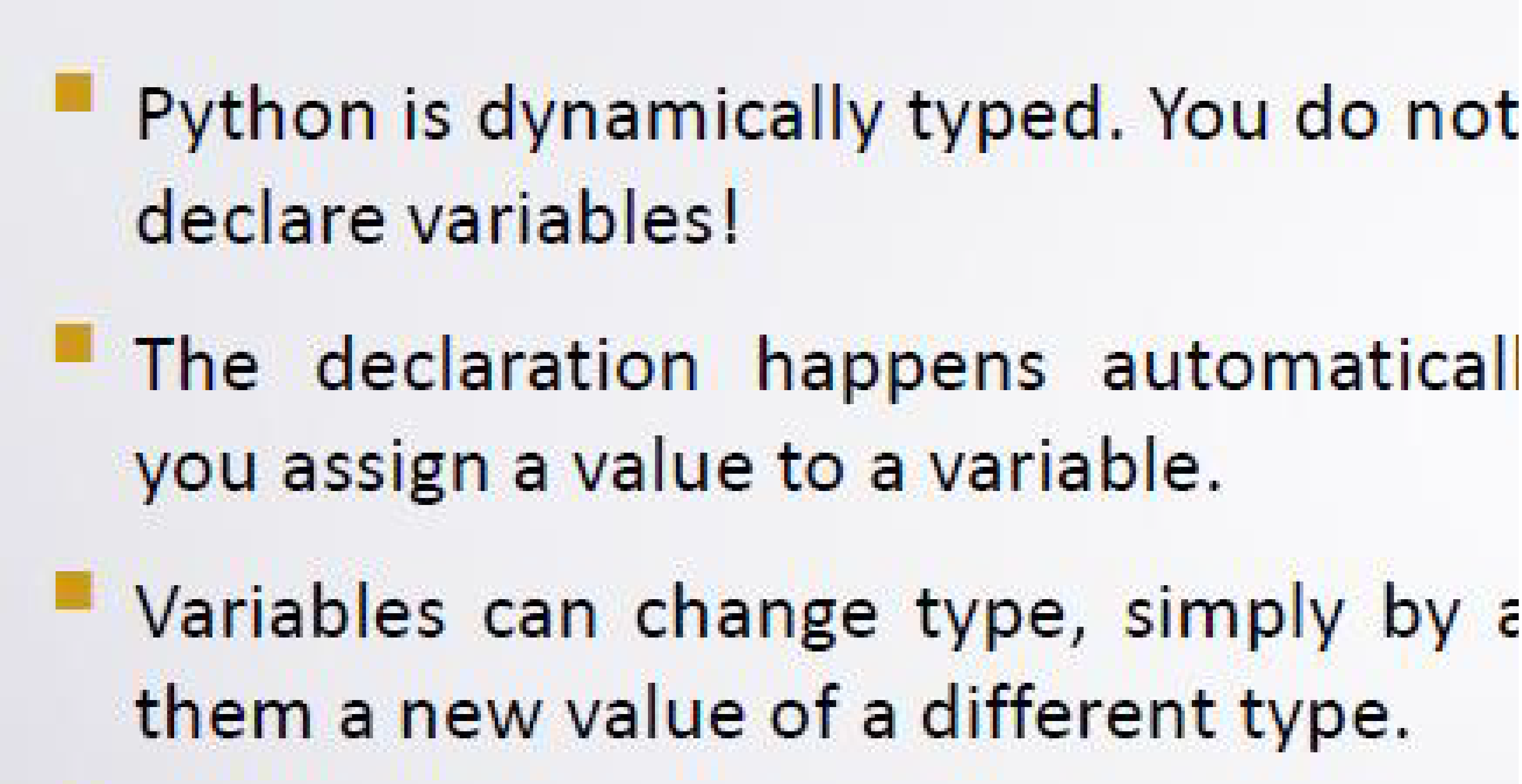
Setting the Data Type

- In Python, the data type is set when you assign a value to a variable

Variables

Assigning values to Variables and Printing it.

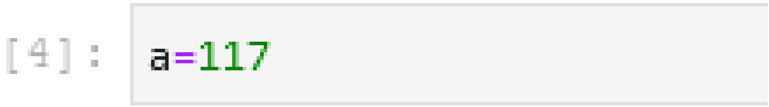
Variables



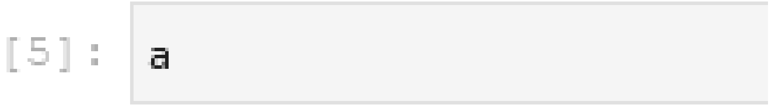
- Python is dynamically typed. You do not declare variables!
- The declaration happens automaticall you assign a value to a variable.
- Variables can change type, simply by a them a new value of a different type.

Variables

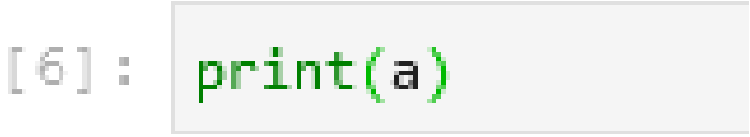
- Printing values of variables
  - Simply write the Variable name
- With print statement
- Assigning float values to same variable
- Assigning string values to the variable
- Everything is treated as object in python, without print statement also variable is displayed



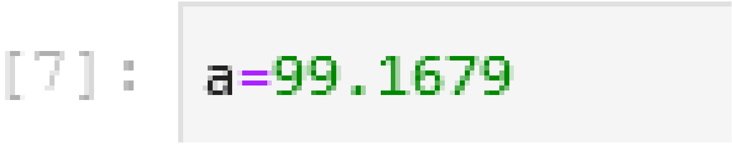
```
[4]: a=117
```



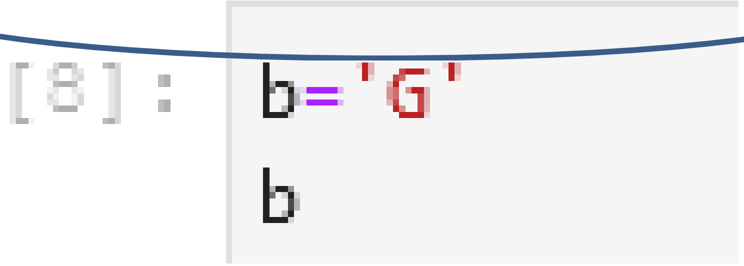
```
[5]: a
```



```
[6]: print(a)
```



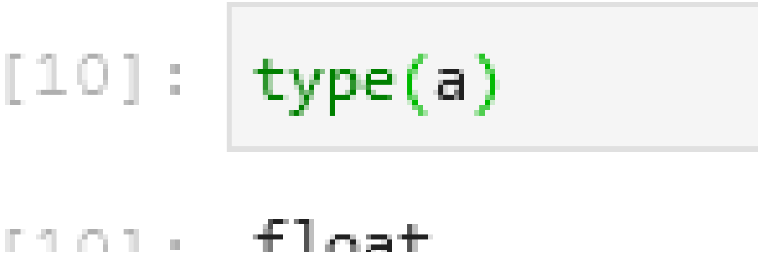
```
[7]: a=99.1679
```



```
[8]: b='G'
     b
```

Variables

- Type command



```
[10]: type(a)
```

```
[10]: float
```

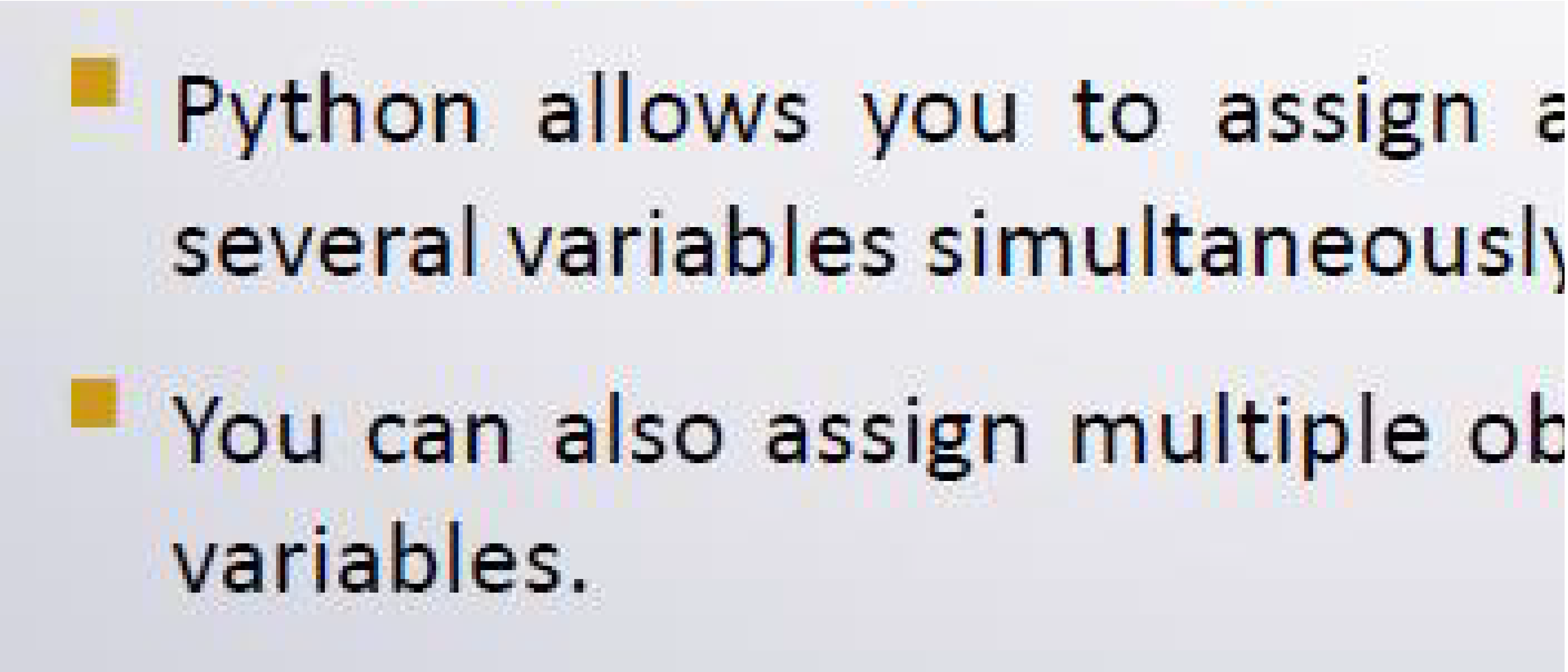
Variables

- >>> x = 3
- After you’ve assigned a value to a variable, you can use the variable in expressions:
- >>> x * 2
- 6

Variables

Multiple Initializations

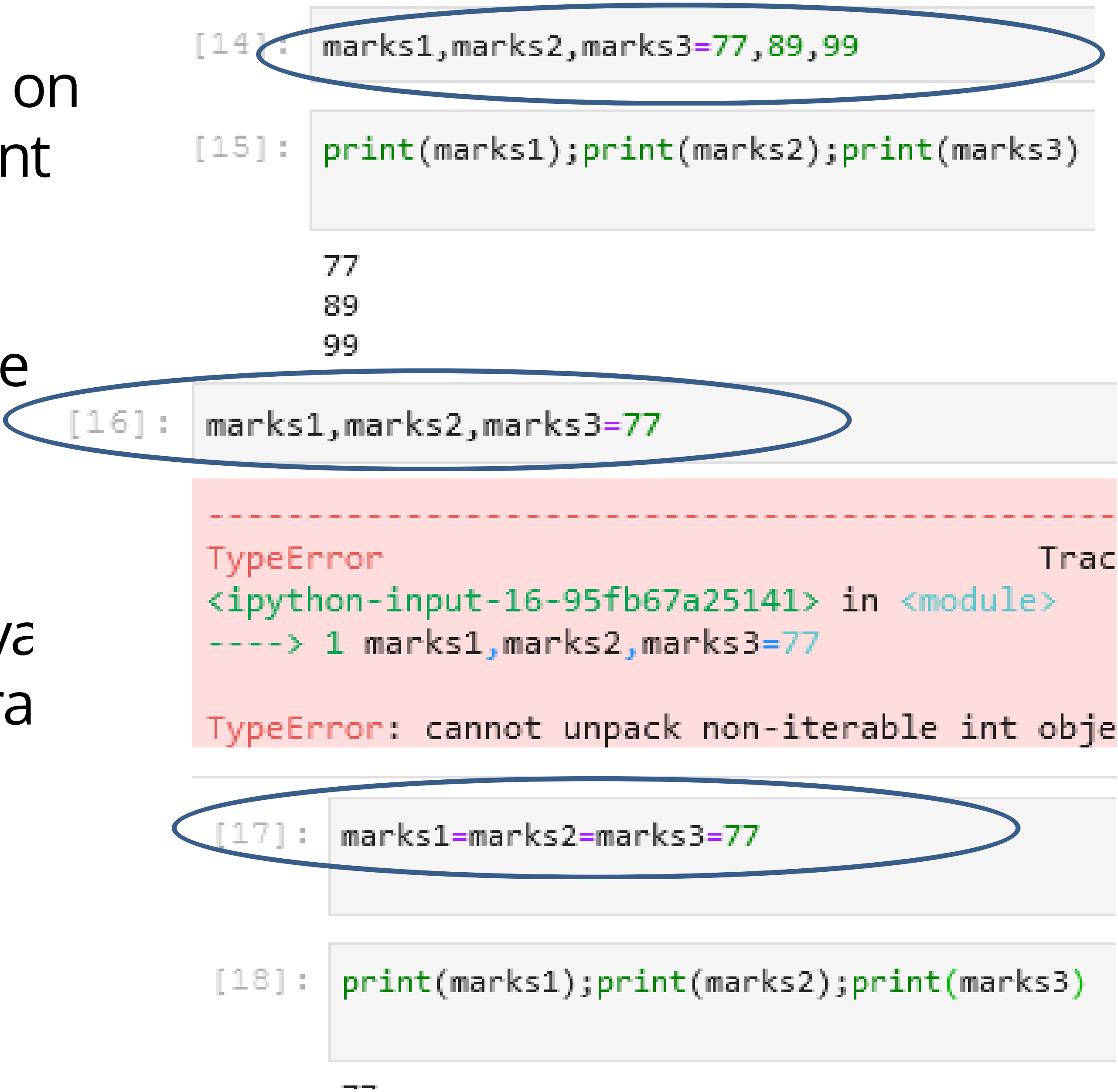
Variables



- Python allows you to assign a several variables simultaneously
- You can also assign multiple ob variables.

Variables

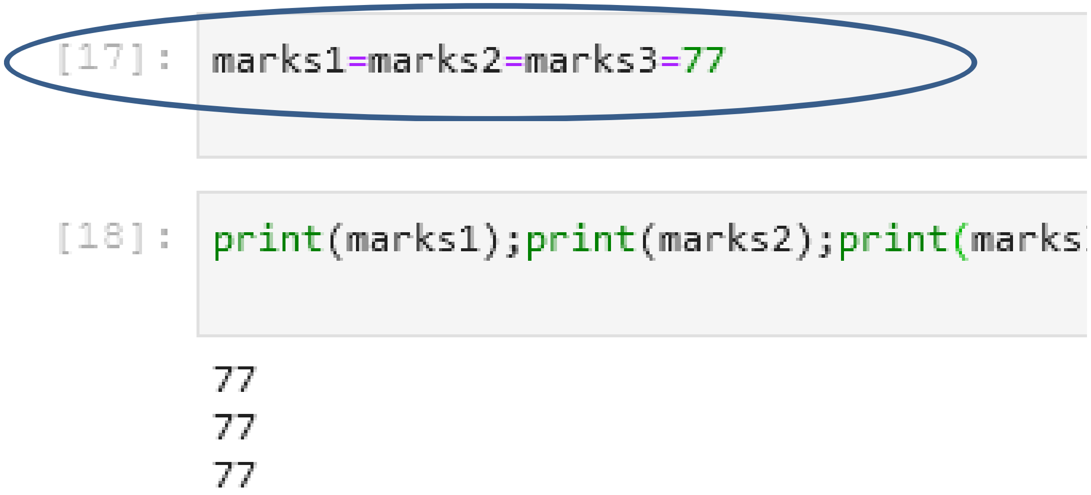
- Multiple initializations on same line with different values
- Multiple variables on initialization with same value on same line
    - Gives error
- Multiple variables initialized with single va with assignment opera



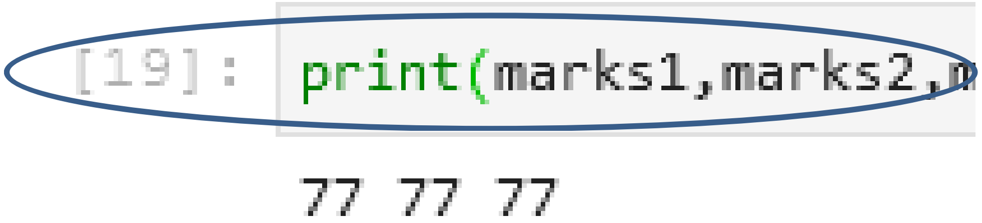
```
[14]: marks1,marks2,marks3=77,89,99

[15]: print(marks1);print(marks2);print(marks3)

77
89
99

[16]: marks1,marks2,marks3=77

---------------------------------------------------------
TypeError                                             Trac
<ipython-input-16-95fb67a25141> in <module>
----> 1 marks1,marks2,marks3=77

TypeError: cannot unpack non-iterable int obje

[17]: marks1=marks2=marks3=77

[18]: print(marks1);print(marks2);print(marks3)
```

Variables

- Multiple values printed with different print statement
- Multiple values printed with a single print statement



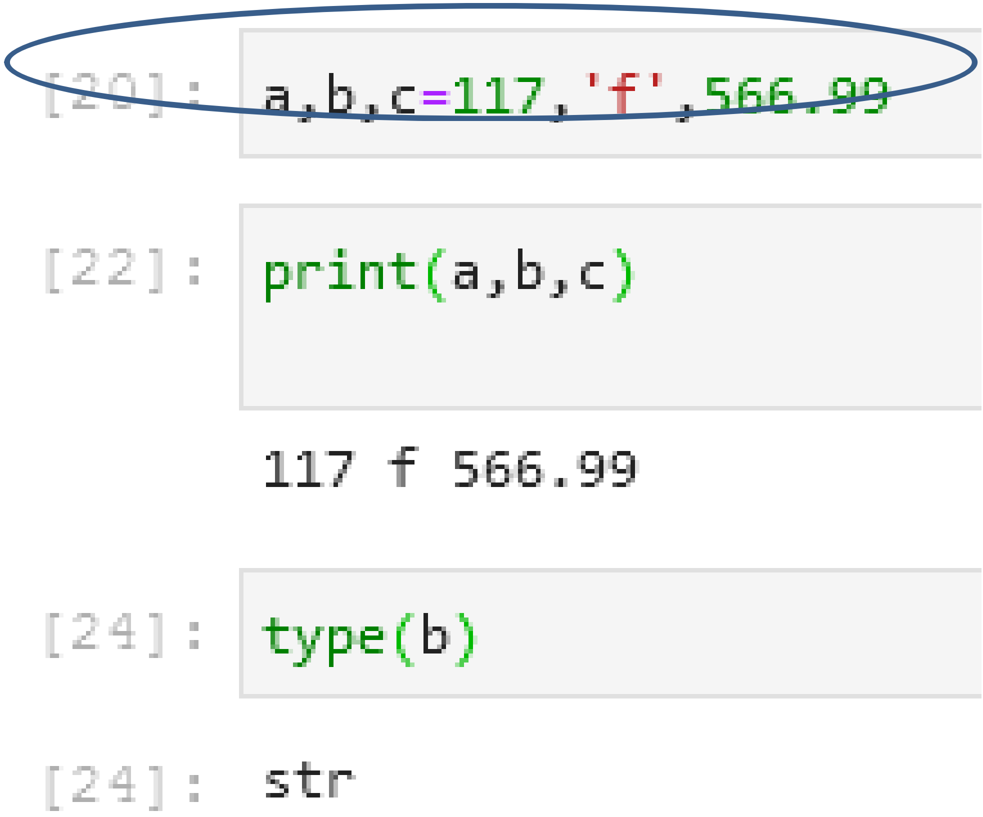
```
[17]:  marks1=marks2=marks3=77

[18]:  print(marks1);print(marks2);print(marks

       77
       77
       77
```



```
[19]:  print(marks1,marks2,m

       77 77 77
```

Variables

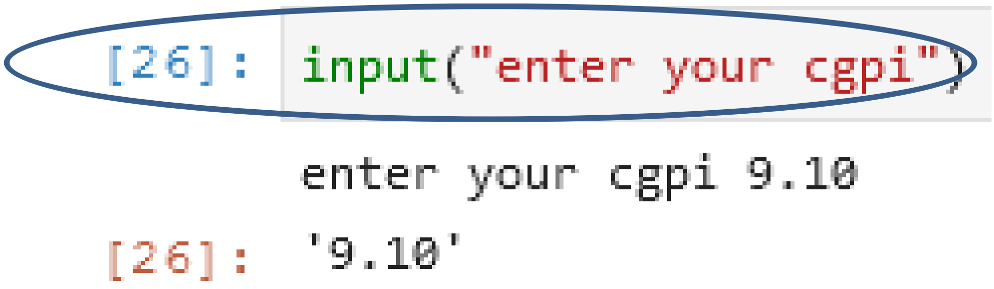
- Different data types initialization in a single statement



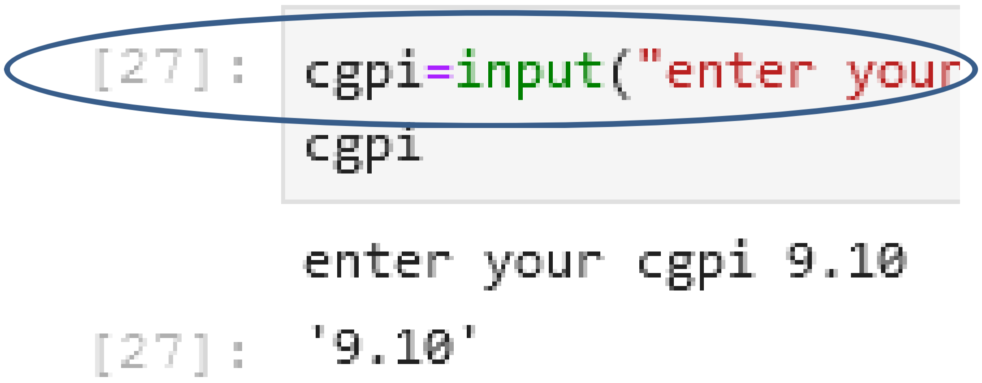
```
[20]: a,b,c=117,'f',566.99

[22]: print(a,b,c)

      117 f 566.99

[24]: type(b)

[24]: str
```

input function

- Reading data Using Input function
- Using Input function with variable



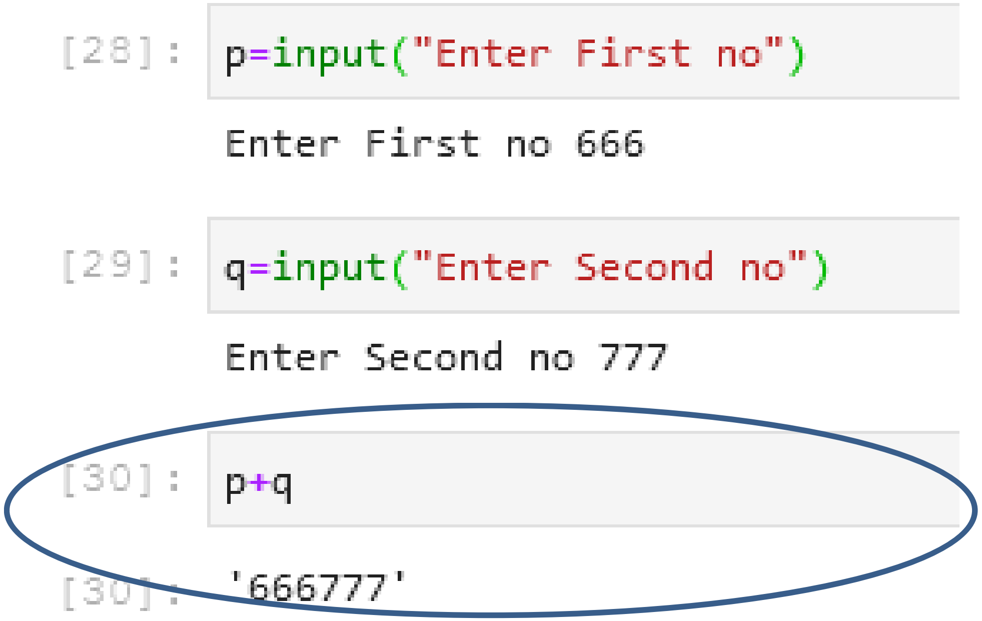
```
[26]: input("enter your cgpi")

      enter your cgpi 9.10

[26]: '9.10'
```



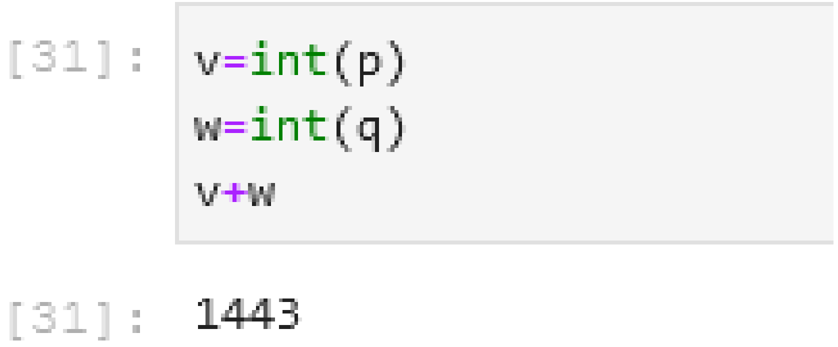
```
[27]: cgpi=input("enter your
      cgpi

      enter your cgpi 9.10

[27]: '9.10'
```

input function

- Data is read as strings
- So for applying arithmetic operations, typecasting is needed
- On using ‘+’ operator, concatenation takes place due to string datatype



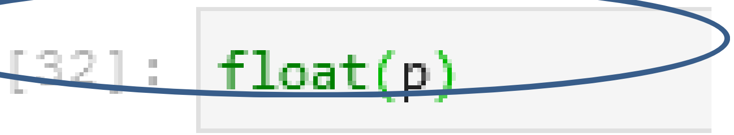
```
[28]: p=input("Enter First no")

      Enter First no 666

[29]: q=input("Enter Second no")

      Enter Second no 777

[30]: p+q

[30]: '666777'
```



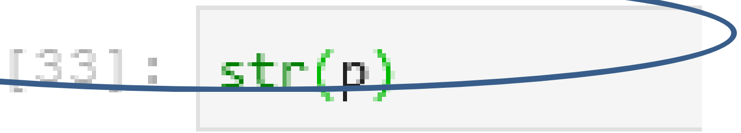
```
[31]: v=int(p)
      w=int(q)
      v+w

[31]: 1443
```

input function

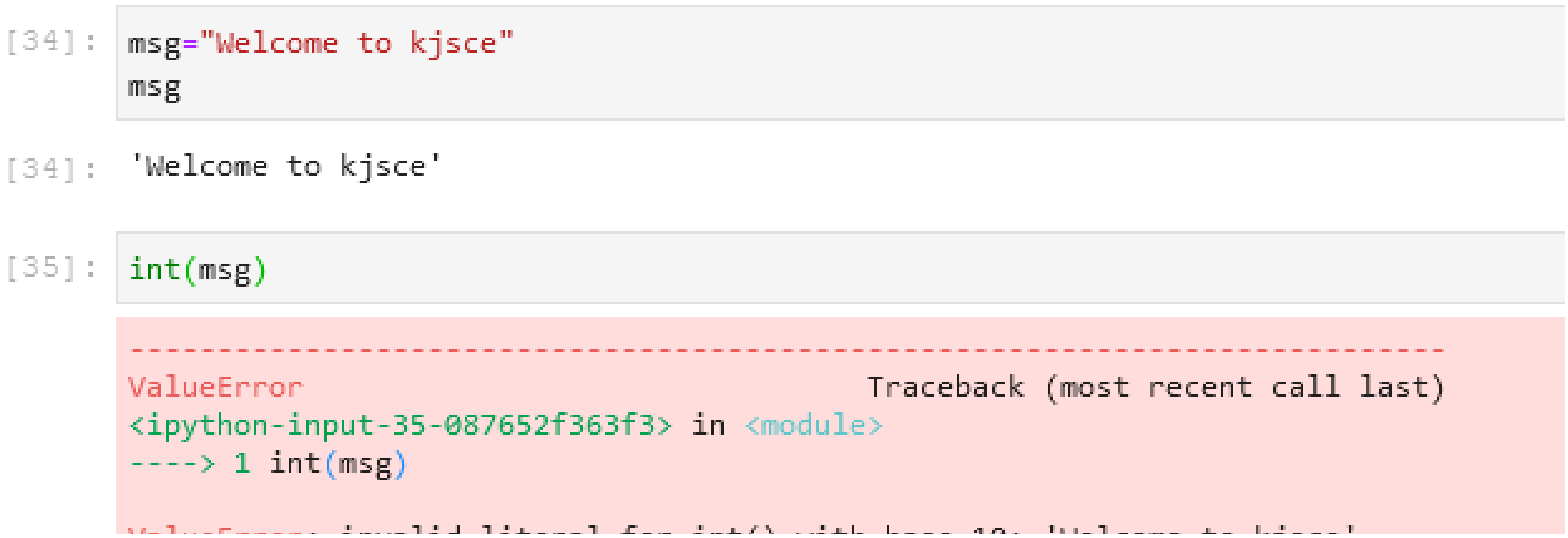
- Type conversion to float
- Type conversion to string data type
- **<u>But every string value cannot be converted to int or float =>i.e. variable initialized with</u>**



```
[32]: float(p)
```



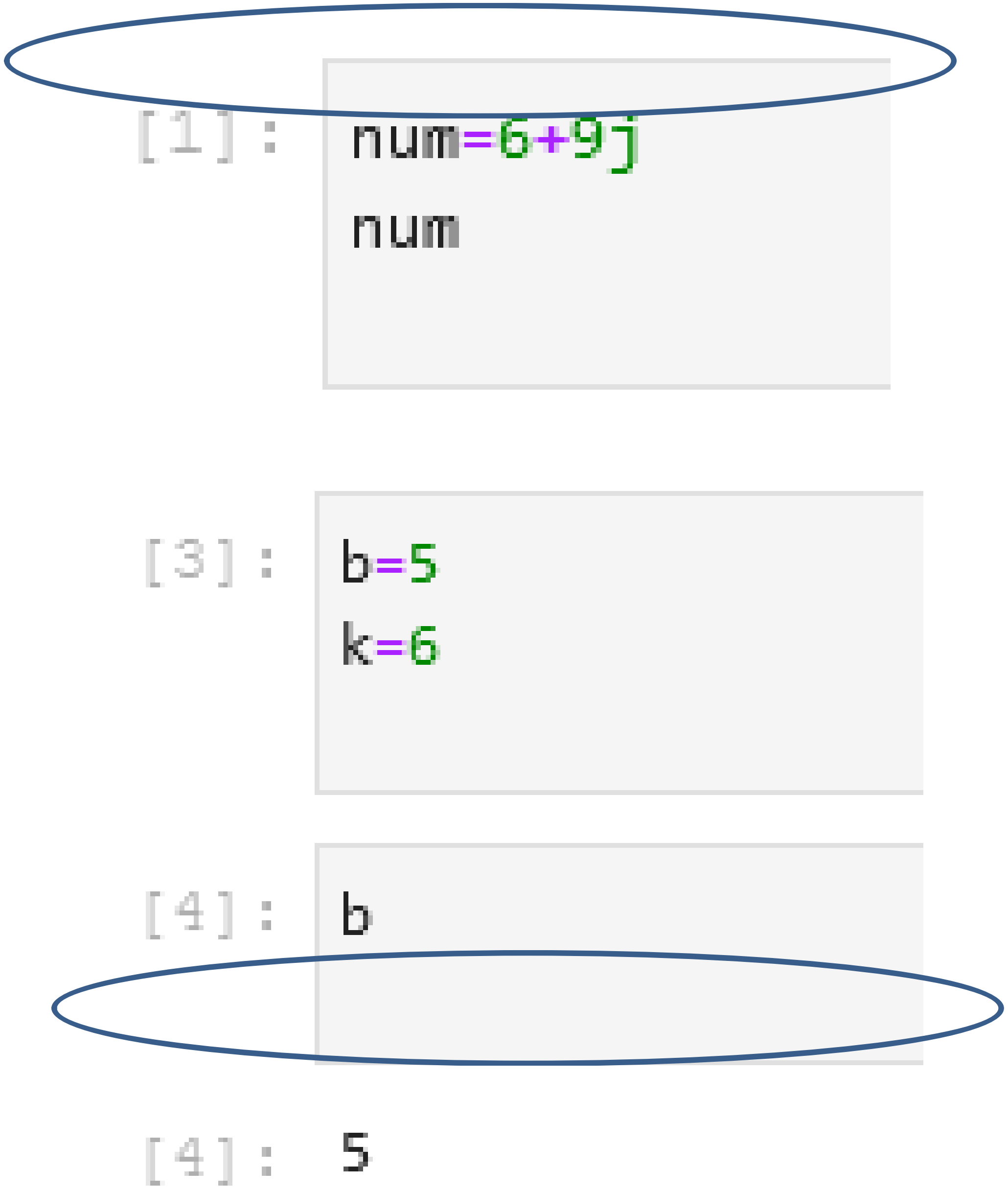
```
[33]: str(p)
```



```
[34]: msg="Welcome to kjsce"
      msg

[34]: 'Welcome to kjsce'

[35]: int(msg)

      ---------------------------------------------------------------------------
      ValueError                                Traceback (most recent call last)
      <ipython-input-35-087652f363f3> in <module>
      ----> 1 int(msg)

      ValueError: invalid literal for int() with base 10: 'Welcome to kjsce'
```

Complex numbers

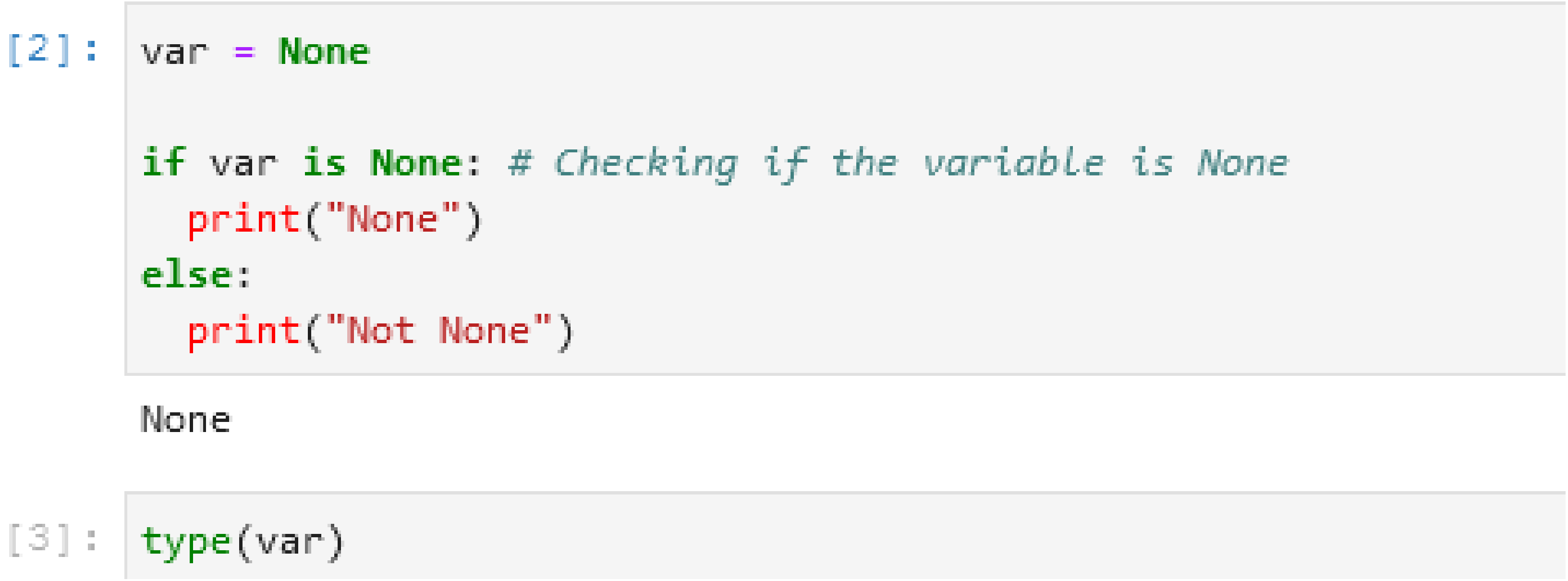
- Printing complex numbers
- Generating complex numbers from two integers



```
[1]: num=6+9j
     num

[3]: b=5
     k=6

[4]: b

[4]: 5
```

None

- **None** is used to define a null variable or an object.
- Assigning value None to Variable
- Testing the value of Variable



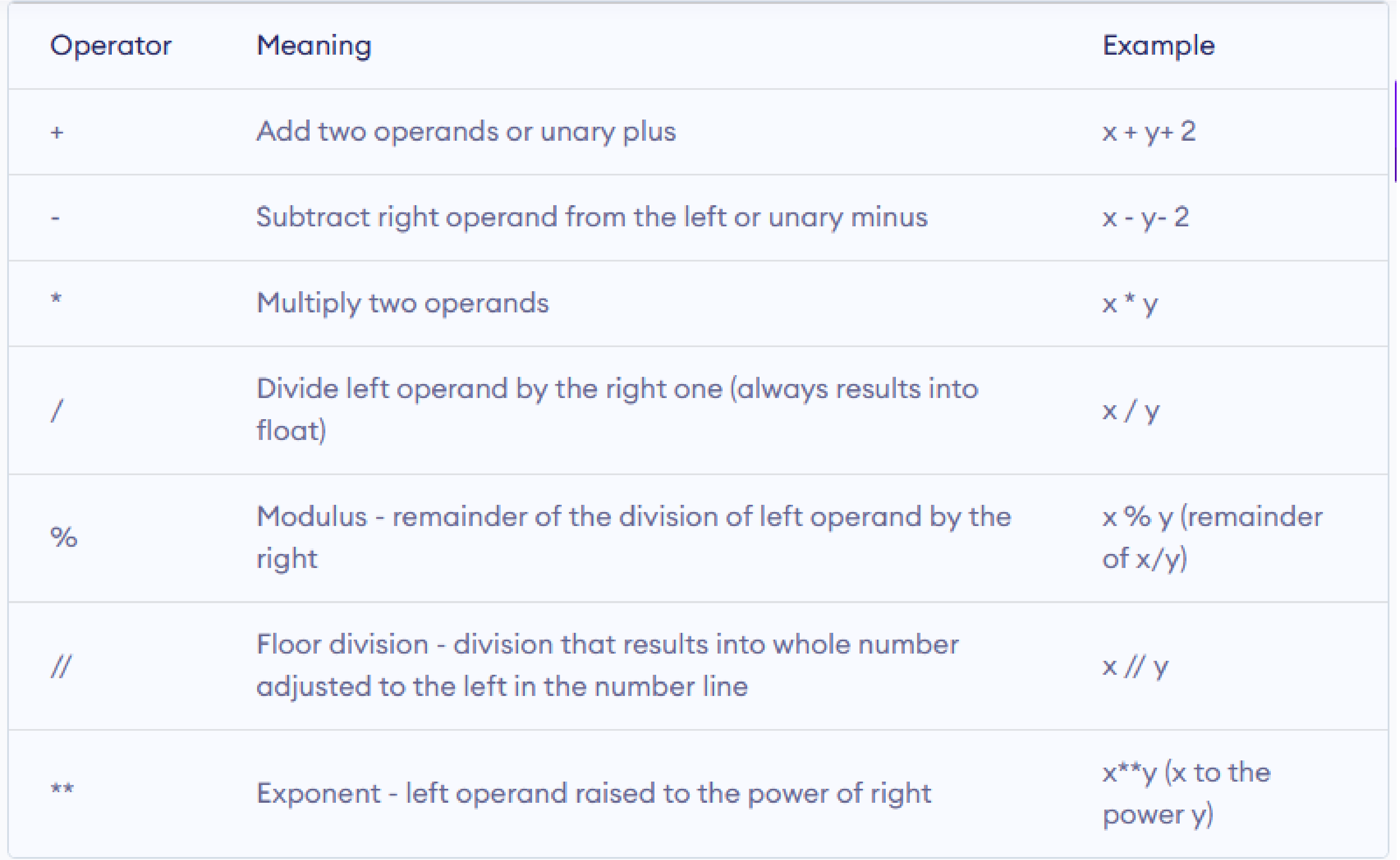
```
[2]: var = None

     if var is None: # Checking if the variable is None
       print("None")
     else:
       print("Not None")

     None

[3]: type(var)
```

Python Operators

- Python language supports the following types of operators.
  - Arithmetic Operators
  - Comparison (Relational) Operators
  - Assignment Operators
  - Logical Operators
  - Bitwise Operators
  - Membership Operators
  - Identity Operators

Arithmetic operators-

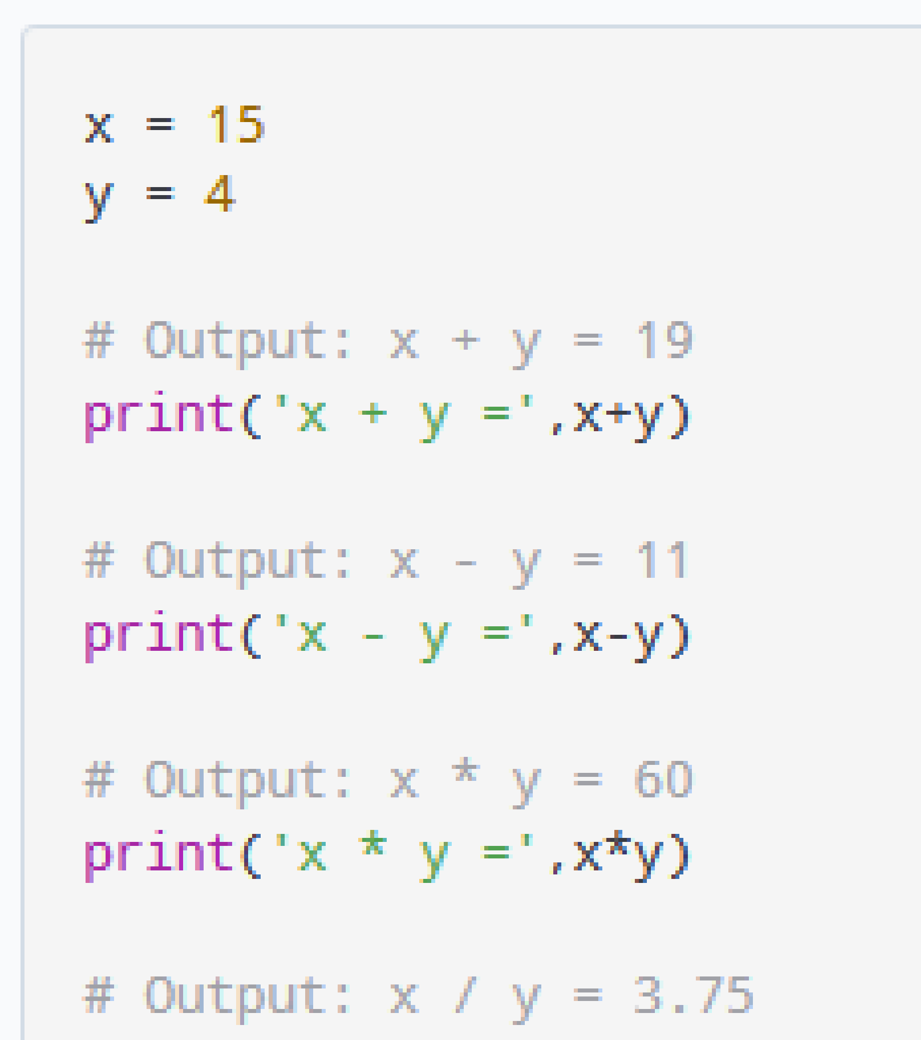
To perform mathematical operations



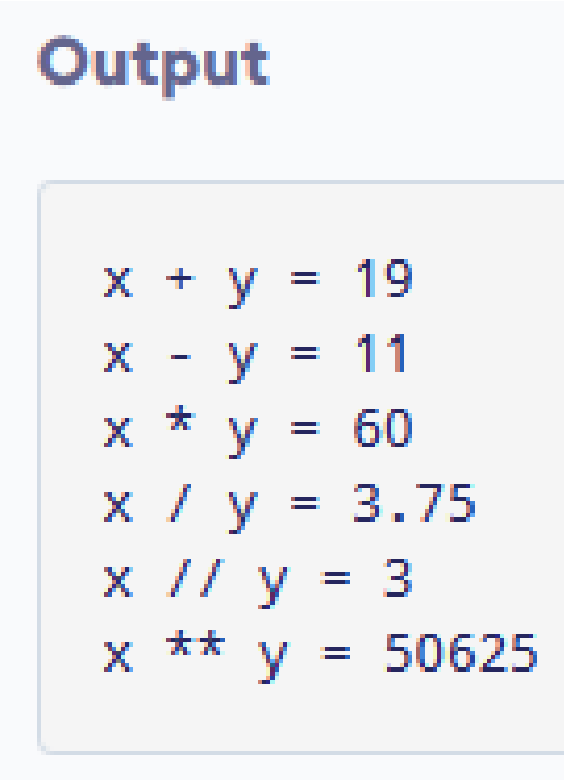
Operator	Meaning	Example
+	Add two operands or unary plus	x + y+ 2
-	Subtract right operand from the left or unary minus	x - y- 2
*	Multiply two operands	x * y
/	Divide left operand by the right one (always results into float)	x / y
%	Modulus - remainder of the division of left operand by the right	x % y (remainder of x/y)
//	Floor division - division that results into whole number adjusted to the left in the number line	x // y
**	Exponent - left operand raised to the power of right	x**y (x to the power y)

Arithmetic operators-

To perform mathematical operations



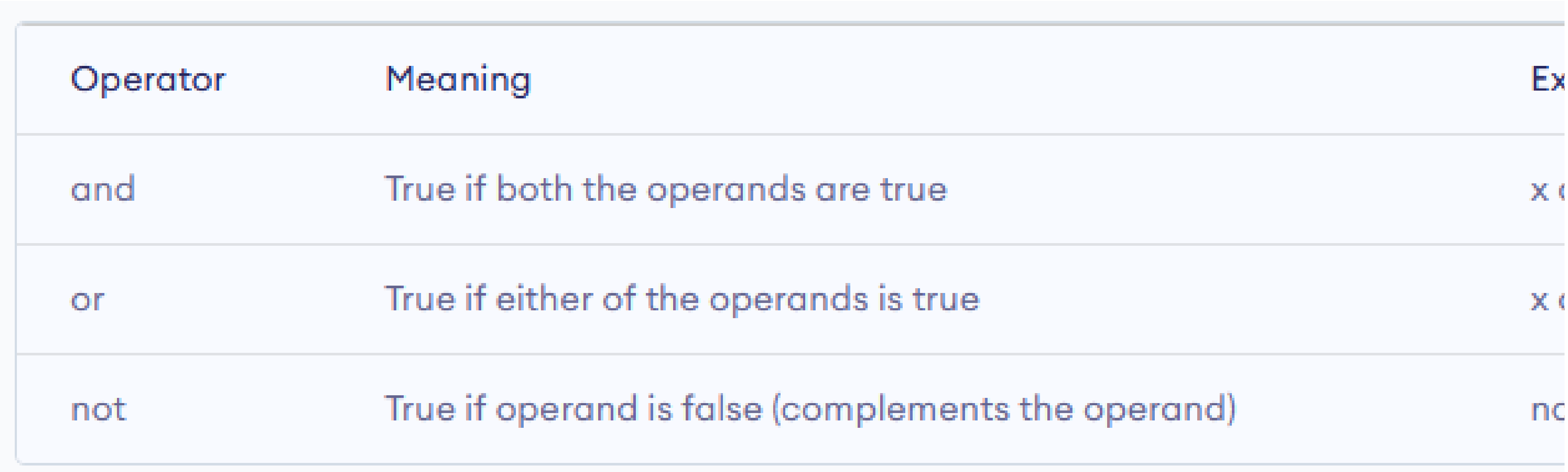
```
x = 15
y = 4

# Output: x + y = 19
print('x + y =',x+y)

# Output: x - y = 11
print('x - y =',x-y)

# Output: x * y = 60
print('x * y =',x*y)

# Output: x / y = 3.75
```



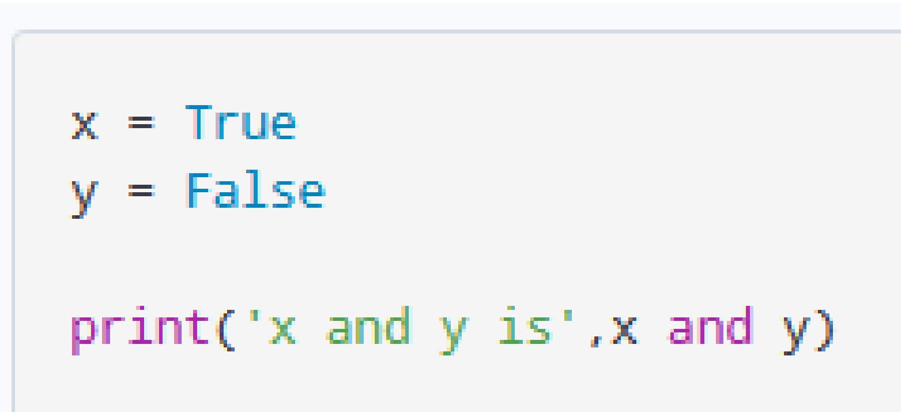
Output

```
x + y = 19
x - y = 11
x * y = 60
x / y = 3.75
x // y = 3
x ** y = 50625
```

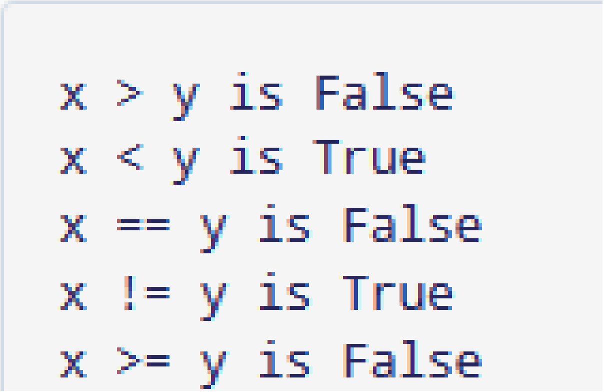
Logical Operators



Operator	Meaning	Ex
and	True if both the operands are true	x
or	True if either of the operands is true	x
not	True if operand is false (complements the operand)	no



```
x = True
y = False

print('x and y is',x and y)
```



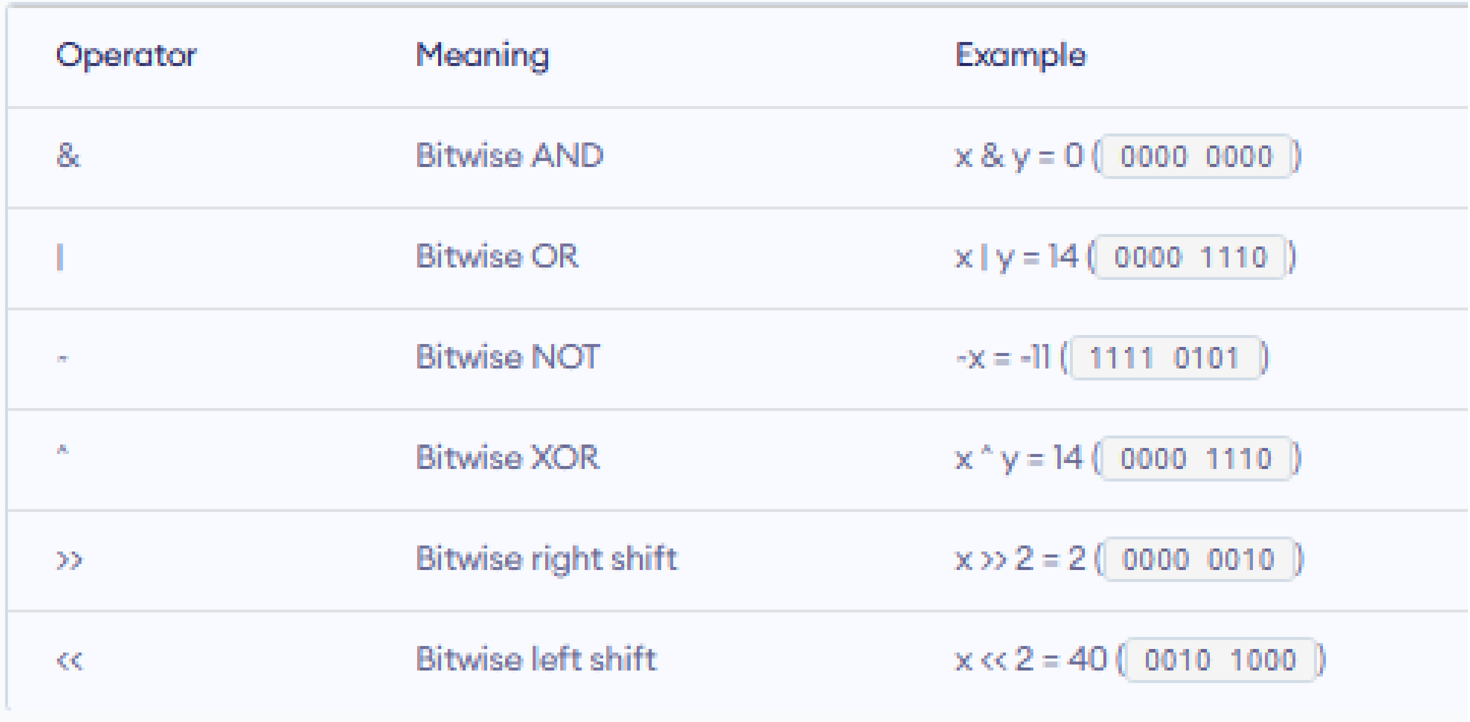
```
x > y is False
x < y is True
x == y is False
x != y is True
x >= y is False
```

Bitwise operators

Bitwise operators act on operands as if they were strings of binary digits. They operate bit by bit, hence the name.

For example, 2 is 10 in binary and 7 is 111.

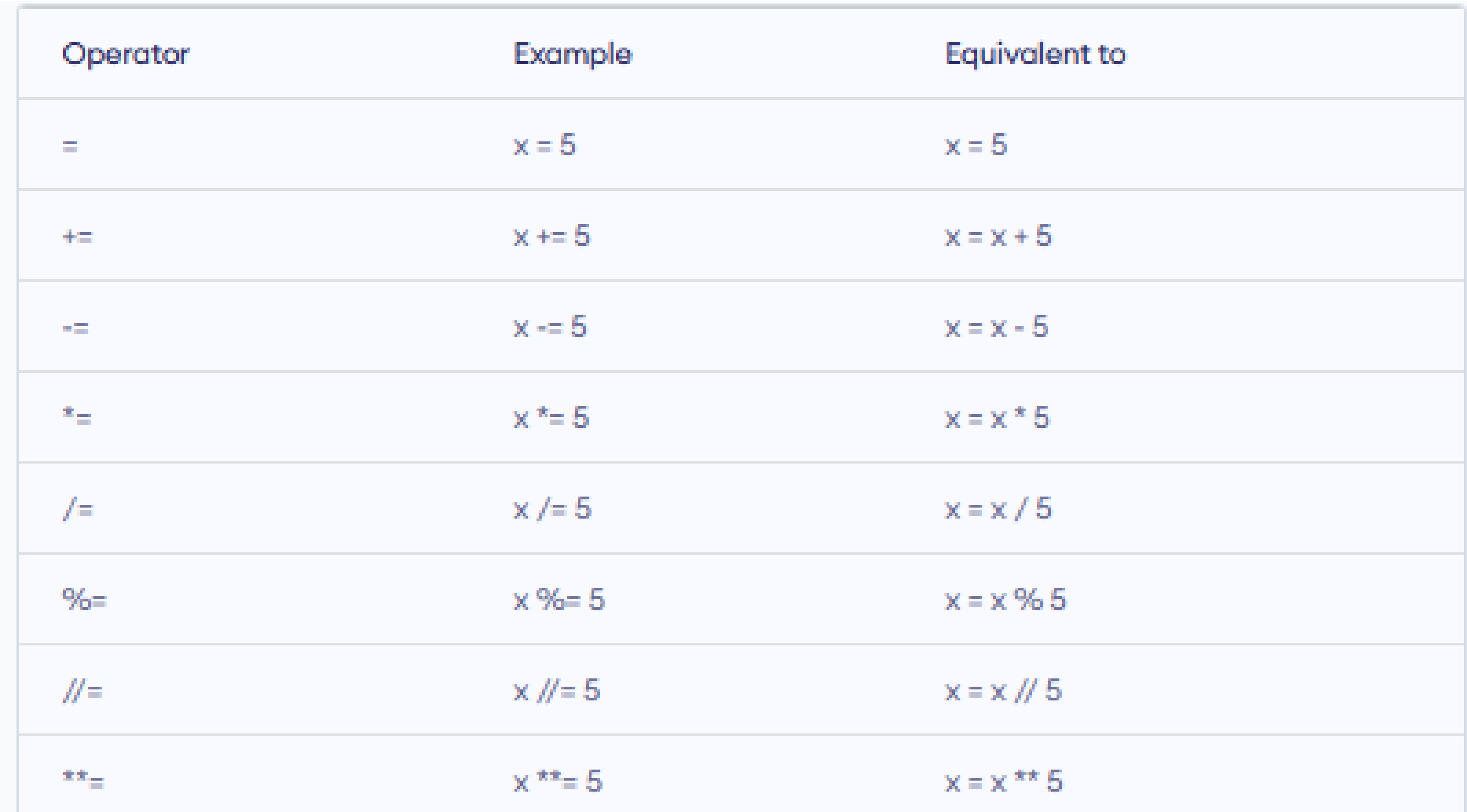
In the table below: Let *x* = 10 (0000 1010 in binary) and *y* = 4 (0000 0100 in binary)



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)

Assignment operators

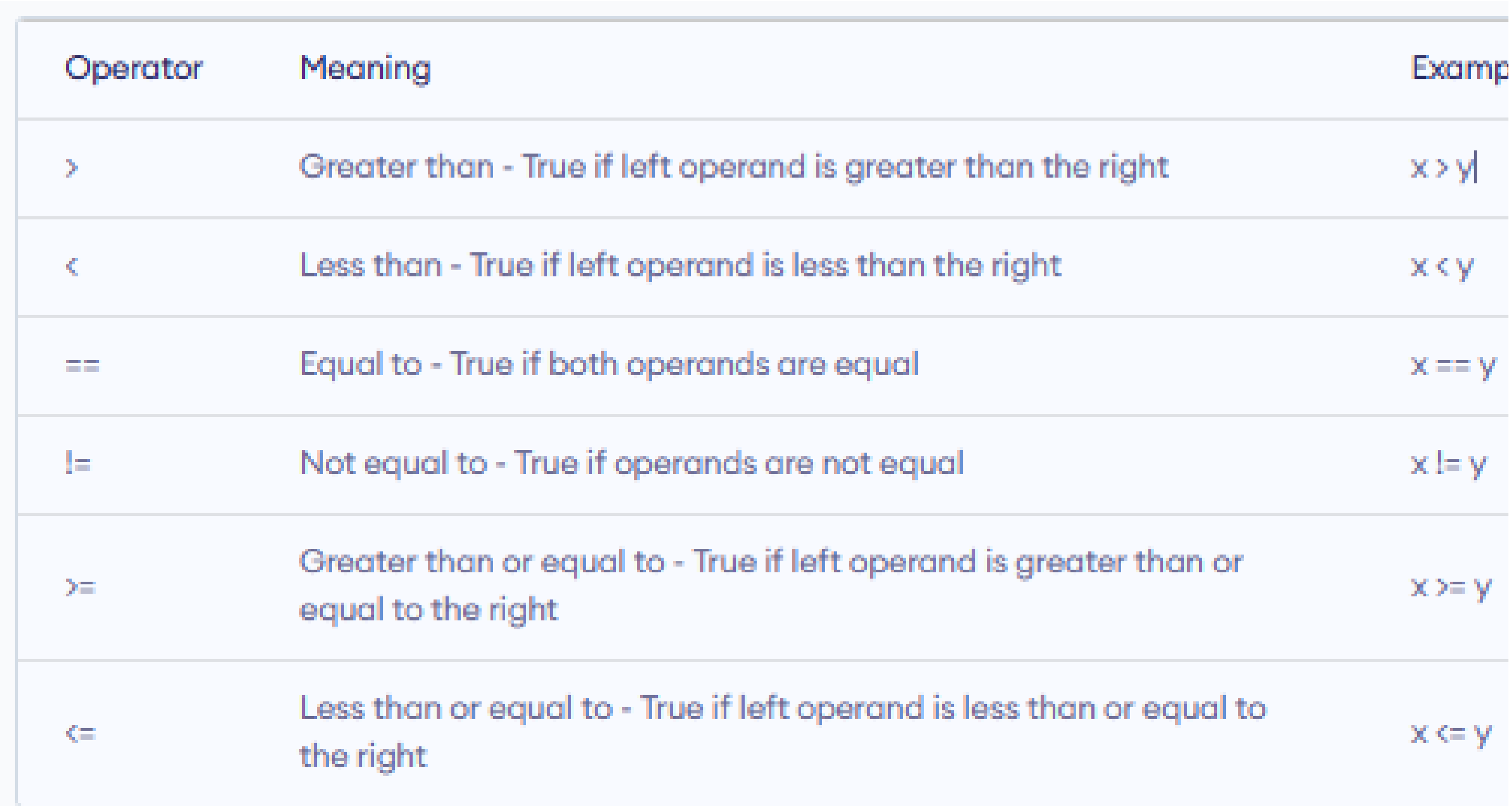
Assignment operators are used in Python to assign values to variables.



Operator	Example	Equivalent 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
**=	x **= 5	x = x ** 5

Comparison operators

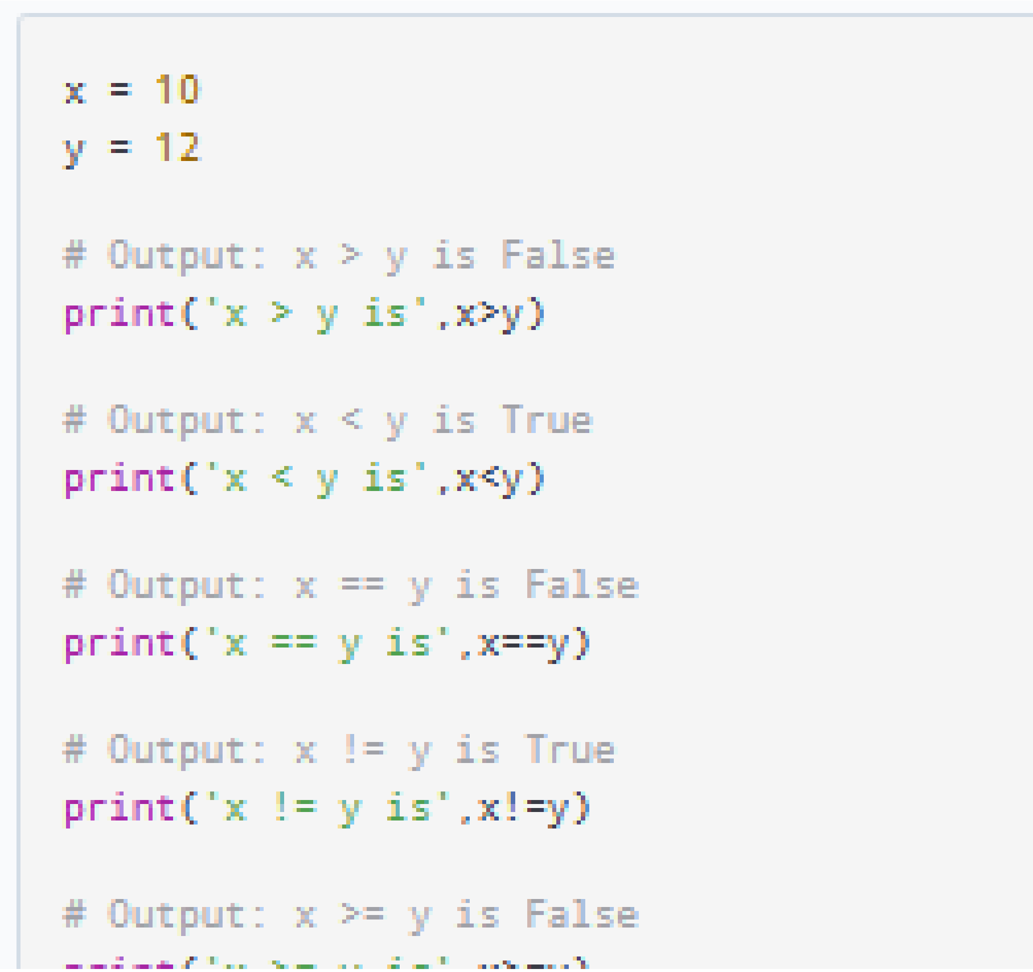
Comparison operators are used to compare values. It returns either True or False according to the condition.



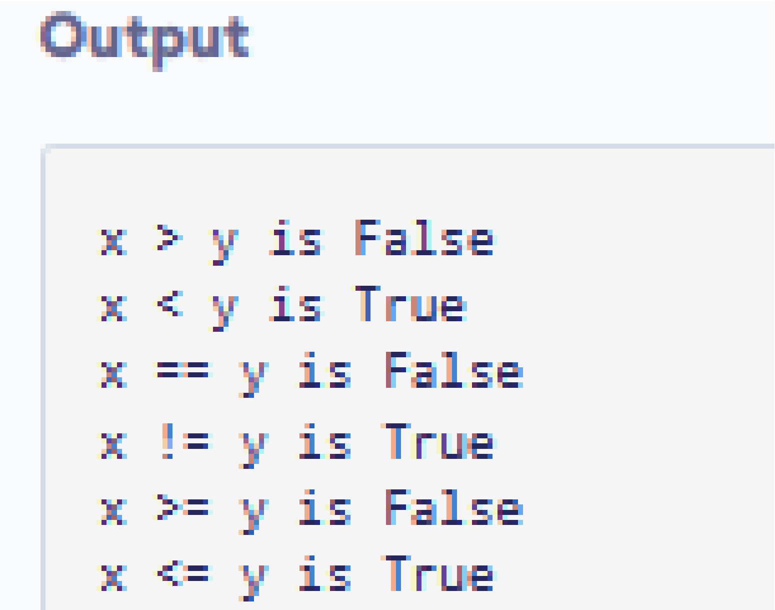
Operator	Meaning	Examp
>	Greater than - True if left operand is greater than the right	x > y
<	Less than - True if left operand is less than the right	x < y
==	Equal to - True if both operands are equal	x == y
!=	Not equal to - True if operands are not equal	x != y
>=	Greater than or equal to - True if left operand is greater than or equal to the right	x >= y
<=	Less than or equal to - True if left operand is less than or equal to the right	x <= y

Comparison operators

Comparison operators are used to compare values. It returns either True or False according to the condition.



```
x = 10
y = 12

# Output: x > y is False
print('x > y is',x>y)

# Output: x < y is True
print('x < y is',x<y)

# Output: x == y is False
print('x == y is',x==y)

# Output: x != y is True
print('x != y is',x!=y)

# Output: x >= y is False
print('x >= y is',x>=y)
```

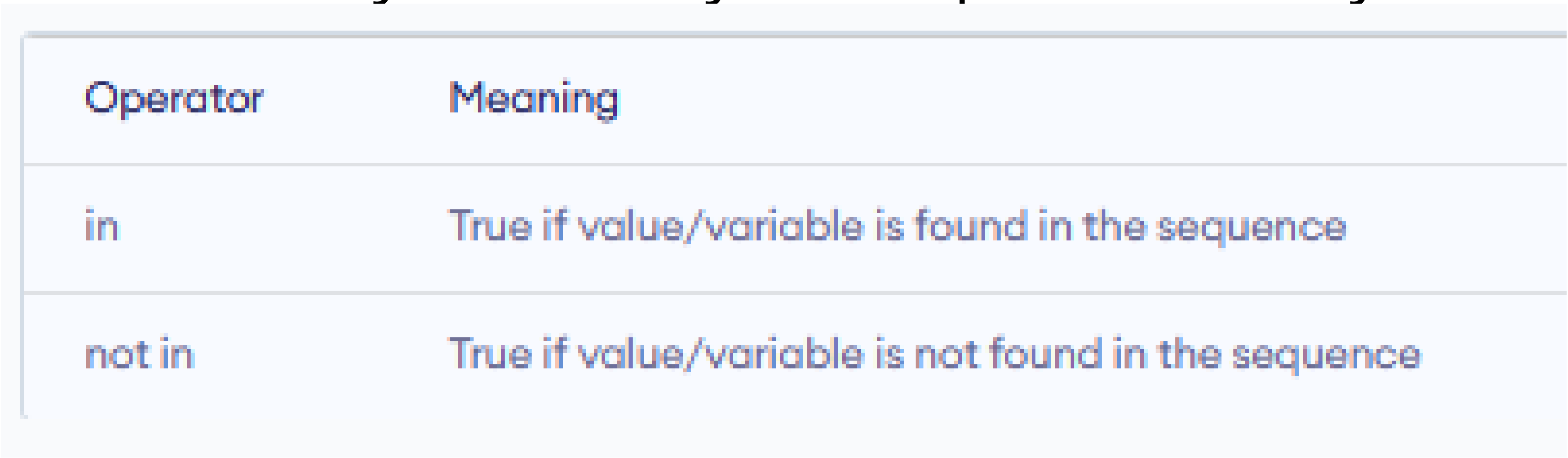


Output

```
x > y is False
x < y is True
x == y is False
x != y is True
x >= y is False
x <= y is True
```

Membership Operators

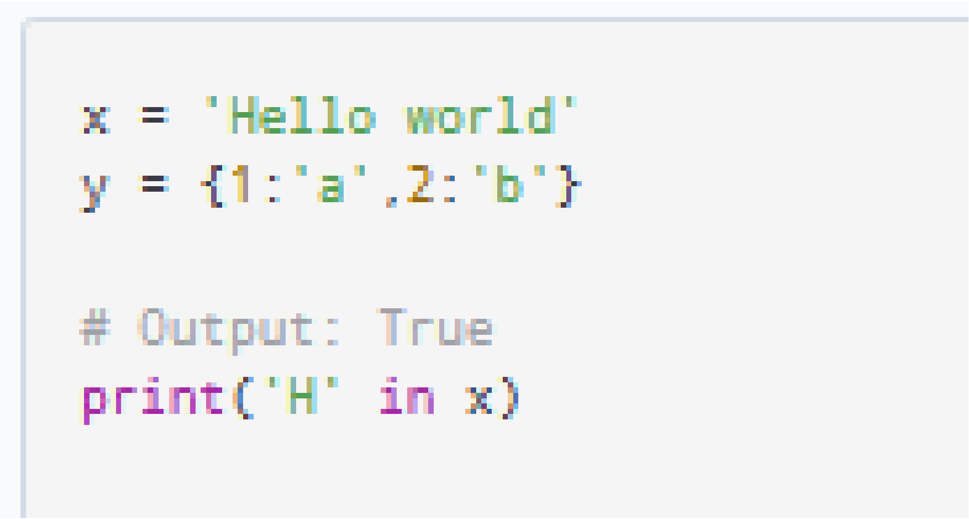
- in and not in are the membership operators in Python.
-
- To test whether a value or variable is found in a sequence (string, list, tuple, set and dictionary).
-
- In a dictionary we can only test for presence of key, not the



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

Membership Operators

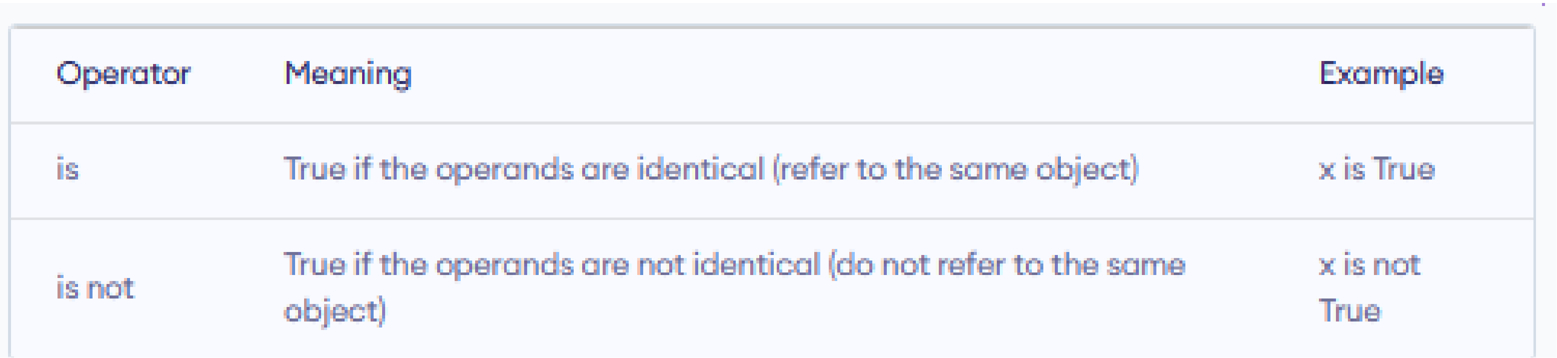
- Here, 'H' is in *x* but 'hello' is not present in *x* (remember, Python is case sensitive).



```
x = 'Hello world'
y = {1:'a',2:'b'}

# Output: True
print('H' in x)
```

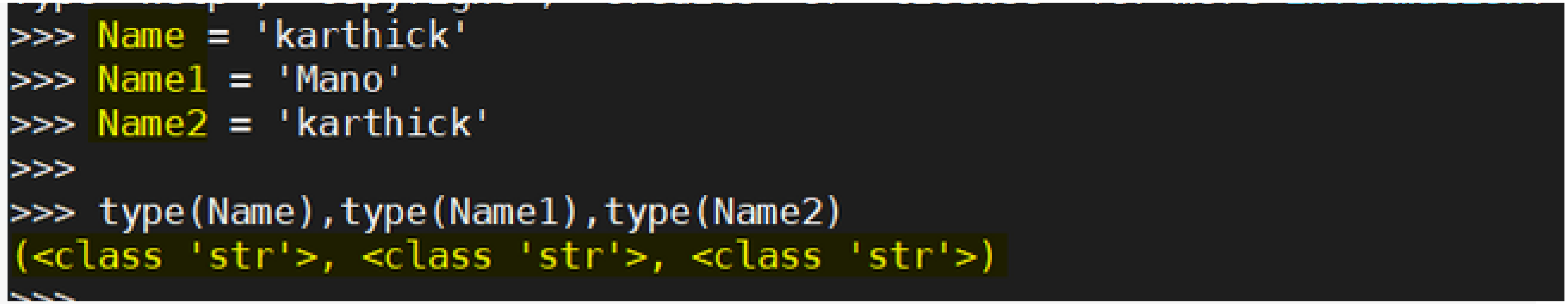
Identity operators

- Identity operators compare the memory locations of two objects.
- Identity operators are used to compare the objects,
  - not if they are equal, but
  - if they are actually the same object, with the same memory location.
- Two variables that are equal does not imply that they are identical.
- They are used to check if two values (or variables) are located on the same part of the memory.

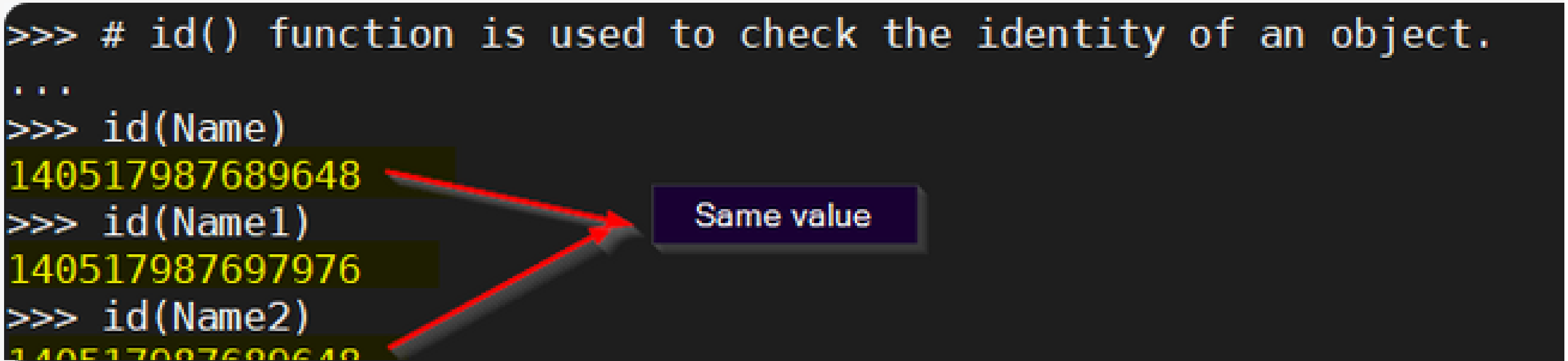


Operator	Meaning	Example
is	True if the operands are identical (refer to the same object)	x is True
is not	True if the operands are not identical (do not refer to the same object)	x is not True

Identity operators

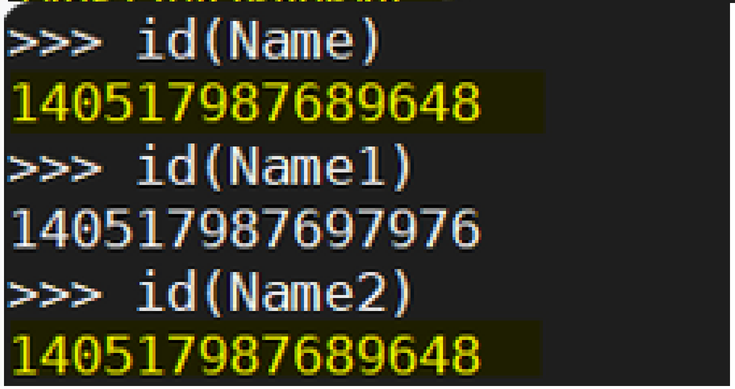


```
>>> Name = 'karthick'
>>> Name1 = 'Mano'
>>> Name2 = 'karthick'
>>>
>>> type(Name),type(Name1),type(Name2)
(<class 'str'>, <class 'str'>, <class 'str'>)
```



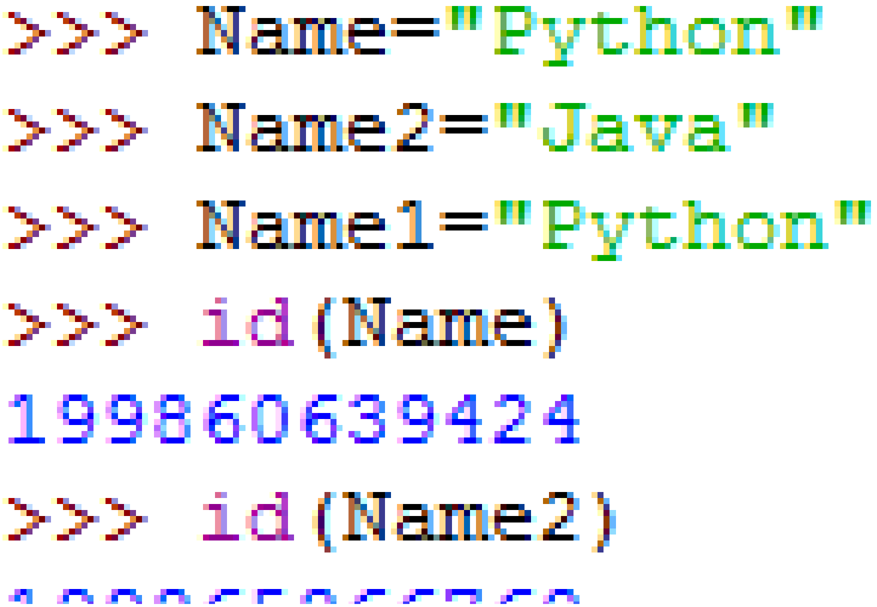
```
>>> # id() function is used to check the identity of an object.
...
>>> id(Name)
140517987689648
>>> id(Name1)
140517987697976
>>> id(Name2)
```

Same value



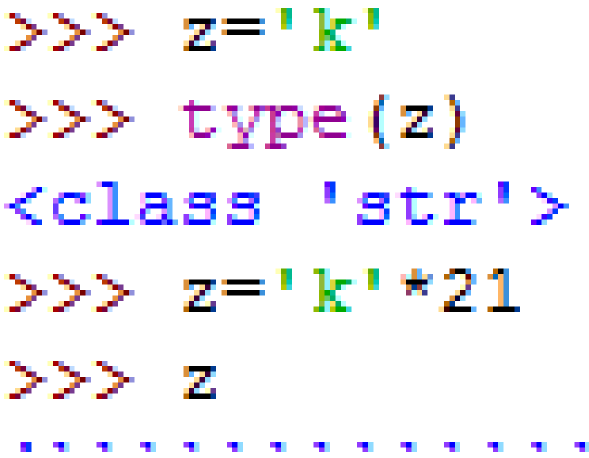
```
>>> id(Name)
140517987689648
>>> id(Name1)
140517987697976
>>> id(Name2)
140517987689648
```

Identity operators



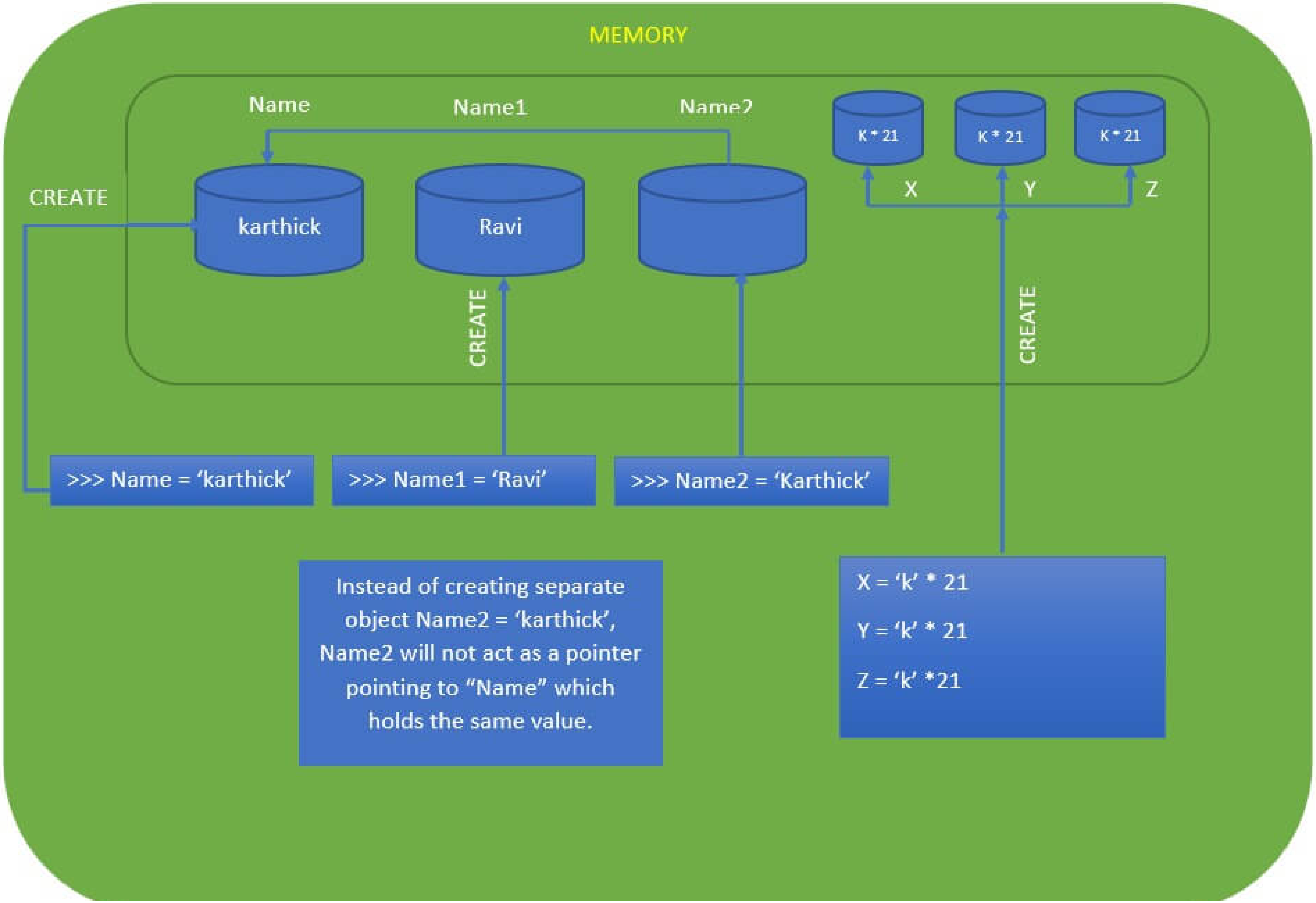
```
>>> Name="Python"
>>> Name2="Java"
>>> Name1="Python"
>>> id(Name)
199860639424
>>> id(Name2)
```

Testing



```
>>> z='k'
>>> type(z)
<class 'str'>
>>> z='k'*21
>>> z
'kkkkkkkkkkkkkkkkkkkkk'
```

- This is because of the python design implementation.
- When you create
- an integer object in range (**-5,256**) and
- string objects greater than or equal to **20** chars,
- instead of creating different objects at memory for the same value these objects act as a pointer to already created objects.



```
>>> Name = 'karthick'
>>> Name1 = 'Ravi'
>>> Name2 = 'Karthick'
```

Instead of creating separate object Name2 = 'karthick', Name2 will not act as a pointer pointing to "Name" which holds the same value.

```
X = 'k' * 21
Y = 'k' * 21
Z = 'k' *21
```

https://www.tecmint.com/learn-python-identity-operator/

Jupyter Lab

- Open https://jupyter.org/