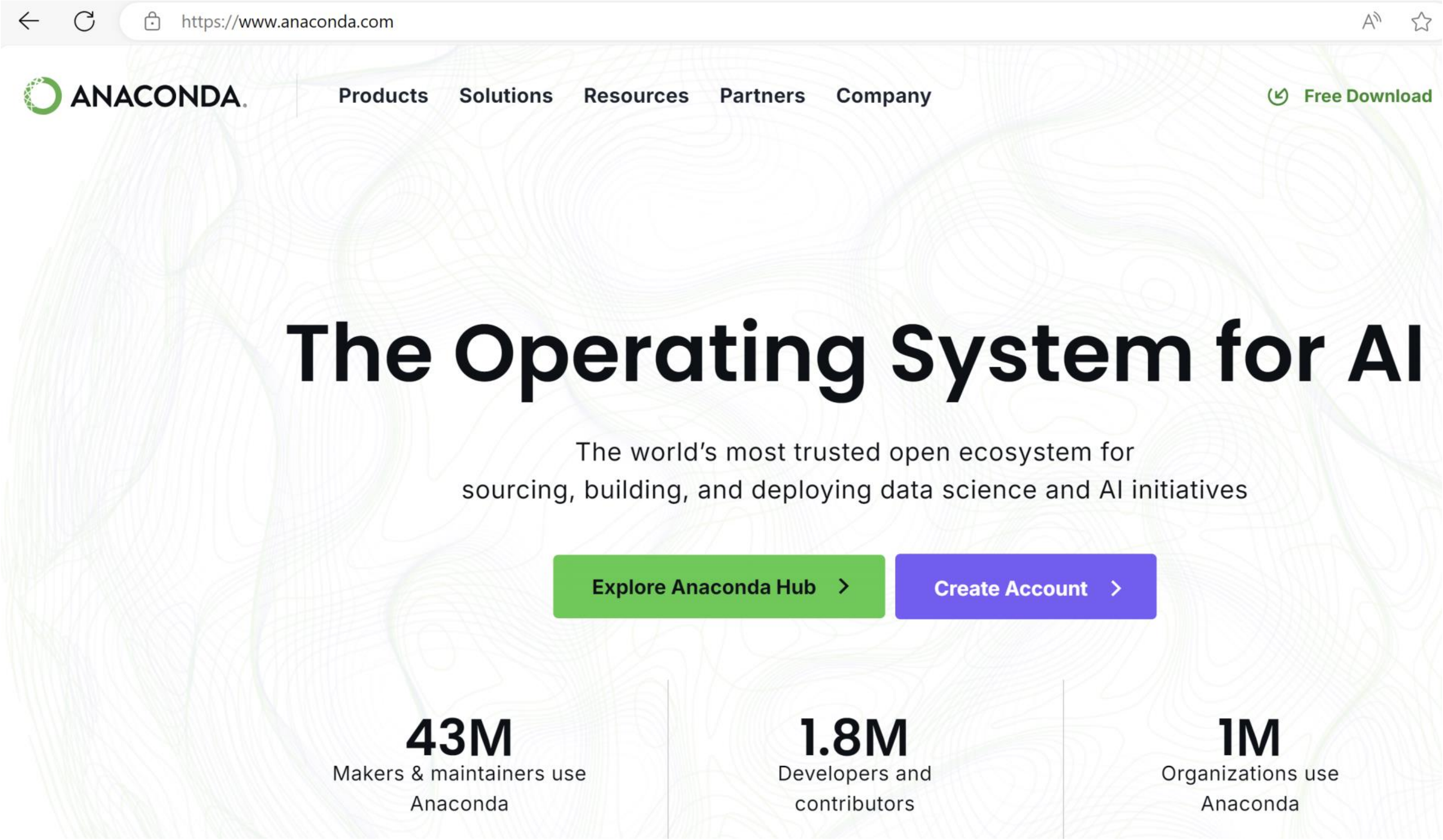
Introduction to Python Programming

Lecture: basic data types

Install Python via Anaconda

- https://www.anaconda.com/download/



The Operating System for AI

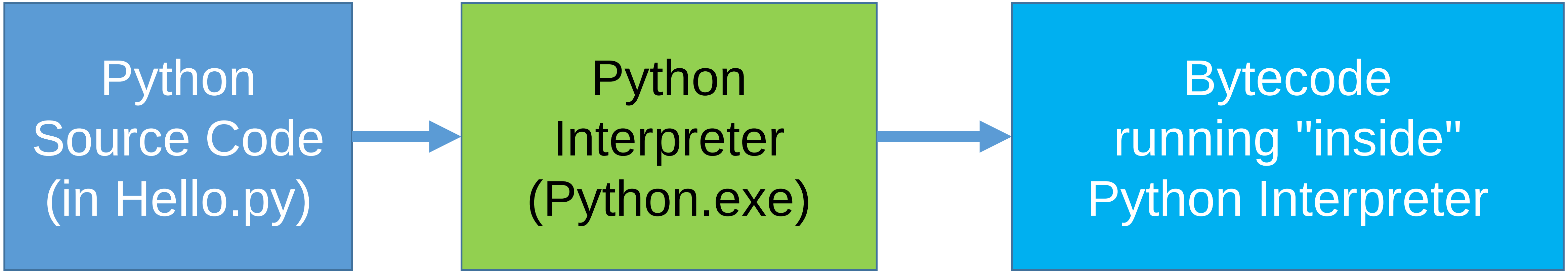
The world's most trusted open ecosystem for sourcing, building, and deploying data science and AI initiatives

43M	1.8M	1M
Makers & maintainers use Anaconda	Developers and contributors	Organizations use Anaconda

Write and Run a Python program

a Python program usually contains one or more lines of source code.

a computer program
to run Python code



Python
Source Code
(in Hello.py)

Python
Interpreter
(Python.exe)

Bytecode
running "inside"
Python Interpreter

```
print('Hello')
```

human-readable

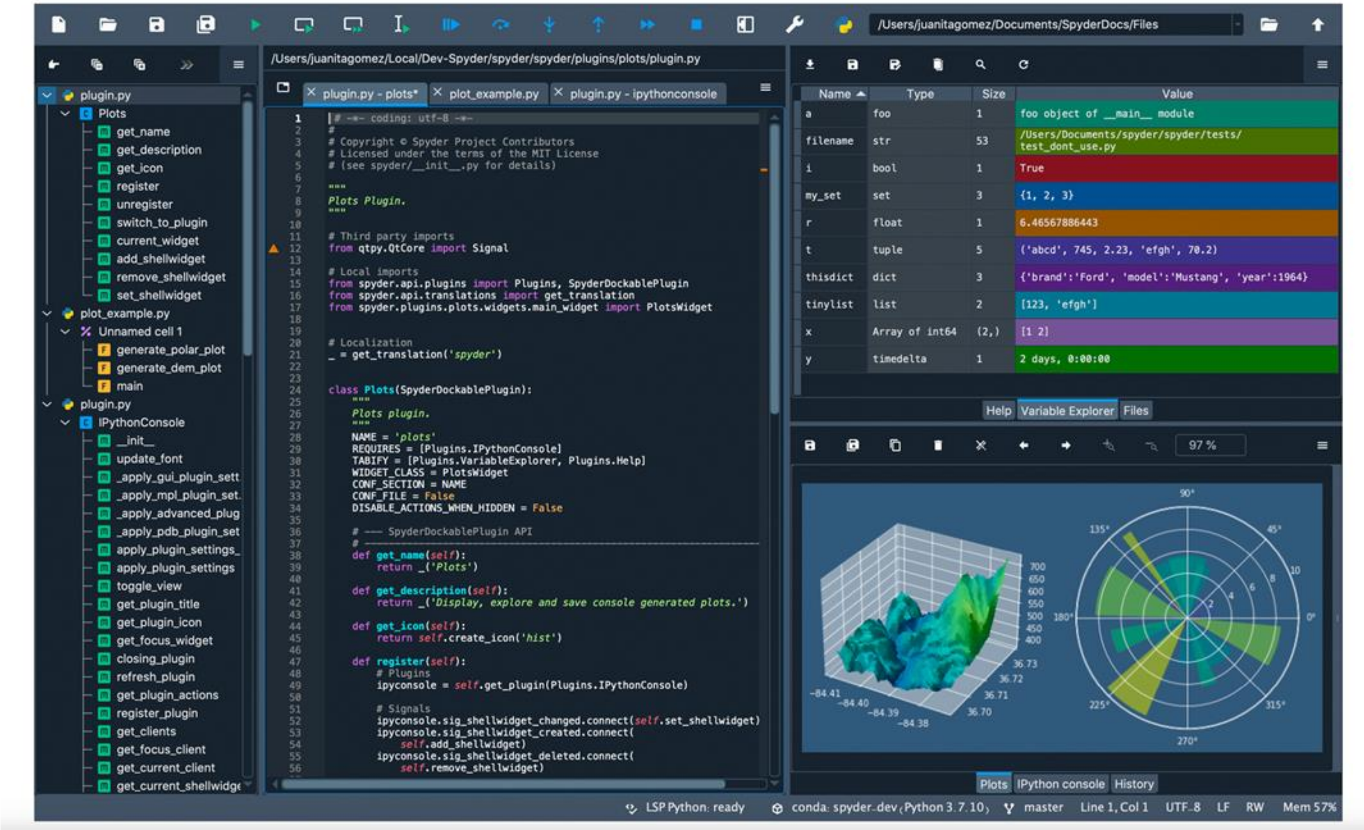
```
b'e\x00d\x00\x83\x01\x01\x00d\x01S\x00'
```

Not human-readable

Spyder IDE (Scientific Python Development Environment) with Python 3

“Spyder is a powerful scientific environment written in Python, for Python, and designed by and for scientists, engineers and data analysts.”

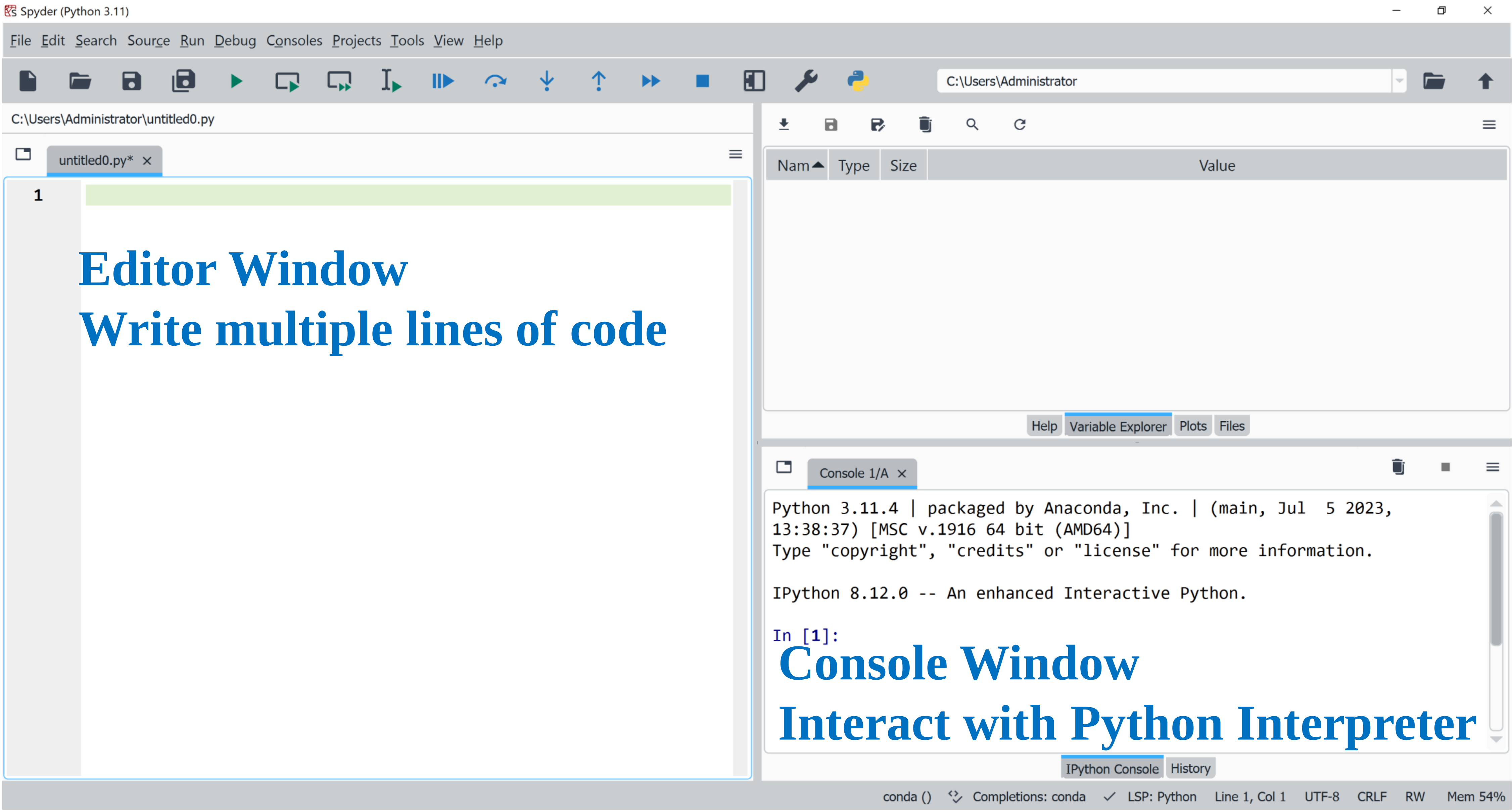
https://www.spyder-ide.org/
https://github.com/spyder-ide



```python
# -*- coding: utf-8 -*-
#
# Copyright © Spyder Project Contributors
# Licensed under the terms of the MIT License
# (see spyder/__init__.py for details)

"""
Plots Plugin.
"""

# Third party imports
from qtpy.QtCore import Signal

# Local imports
from spyder.api.plugins import Plugins, SpyderDockablePlugin
from spyder.api.translations import get_translation
from spyder.plugins.plots.widgets.main_widget import PlotsWidget


# Localization
_ = get_translation('spyder')


class Plots(SpyderDockablePlugin):
    """
    Plots plugin.
    """
    NAME = 'plots'
    REQUIRES = [Plugins.IPythonConsole]
    TABIFY = [Plugins.VariableExplorer, Plugins.Help]
    WIDGET_CLASS = PlotsWidget
    CONF_SECTION = NAME
    CONF_FILE = False
    DISABLE_ACTIONS_WHEN_HIDDEN = False

    # --- SpyderDockablePlugin API
    # ------------------------------------------------------------------------
    def get_name(self):
        return _('Plots')

    def get_description(self):
        return _('Display, explore and save console generated plots.')

    def get_icon(self):
        return self.create_icon('hist')

    def register(self):
        # Plugins
        ipyconsole = self.get_plugin(Plugins.IPythonConsole)

        # Signals
        ipyconsole.sig_shellwidget_changed.connect(self.set_shellwidget)
        ipyconsole.sig_shellwidget_created.connect(
            self.add_shellwidget)
        ipyconsole.sig_shellwidget_deleted.connect(
            self.remove_shellwidget)
```

Name ▲	Type	Size	Value
a	foo	1	foo object of __main__ module
filename	str	53	/Users/Documents/spyder/spyder/tests/test_dont_use.py
i	bool	1	True
my_set	set	3	{1, 2, 3}
r	float	1	6.46567886443
t	tuple	5	('abcd', 745, 2.23, 'efgh', 70.2)
thisdict	dict	3	{'brand':'Ford', 'model':'Mustang', 'year':1964}
tinylist	list	2	[123, 'efgh']
x	Array of int64	(2,)	[1 2]
y	timedelta	1	2 days, 0:00:00

Spyder IDE (Scientific Python Development Environment) with Python 3


Spyder (Python 3.11)
File Edit Search Source Run Debug Consoles Projects Tools View Help
C:\Users\Administrator
C:\Users\Administrator\untitled0.py
untitled0.py*
1
Nam Type Size Value
Help Variable Explorer Plots Files
Console 1/A
IPython Console History
conda () Completions: conda LSP: Python Line 1, Col 1 UTF-8 CRLF RW Mem 54%

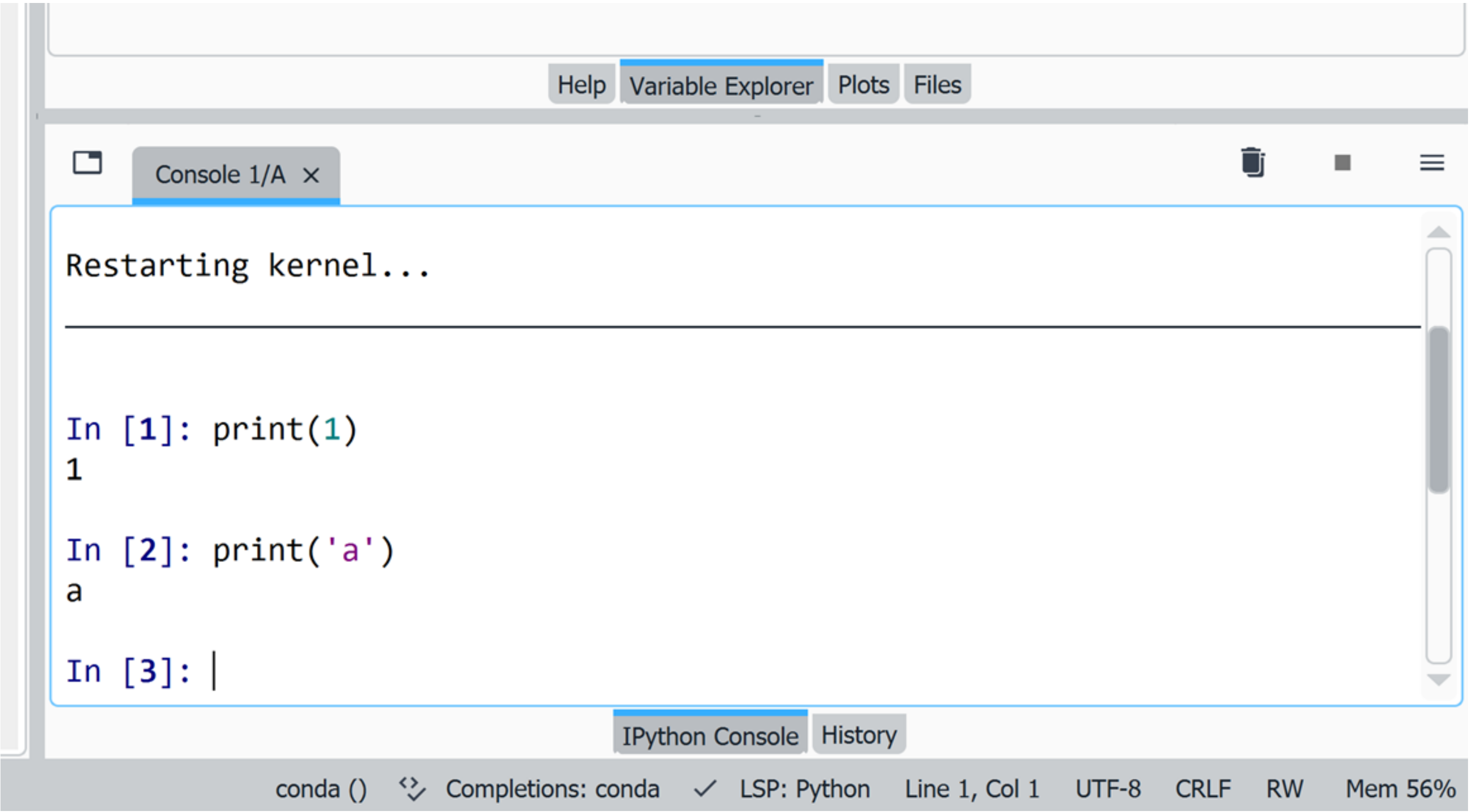
```
Python 3.11.4 | packaged by Anaconda, Inc. | (main, Jul  5 2023, 13:38:37) [MSC v.1916 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 8.12.0 -- An enhanced Interactive Python.

In [1]:
```

Editor Window
Write multiple lines of code

Console Window
Interact with Python Interpreter

The Function: print

- print(object): to print an object to the console window



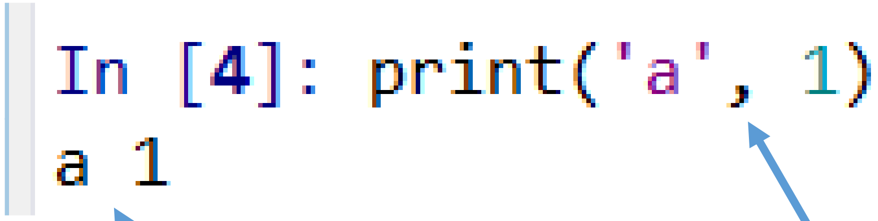
```
Restarting kernel...


In [1]: print(1)
1

In [2]: print('a')
a

In [3]:
```

print

- print(object1, object2): to print two objects to the console window

Code

Output



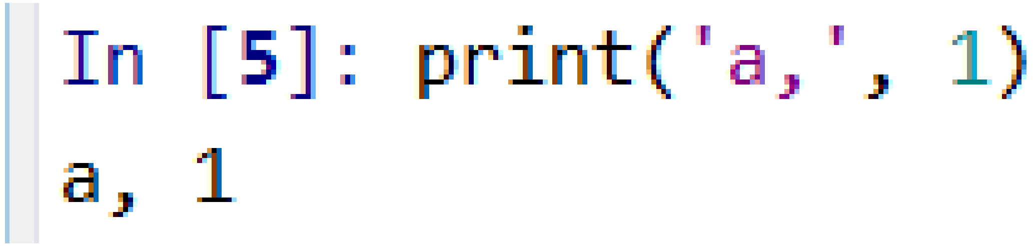
```
In [4]: print('a', 1)
a 1
```

comma is NOT printed

an empty space

Code

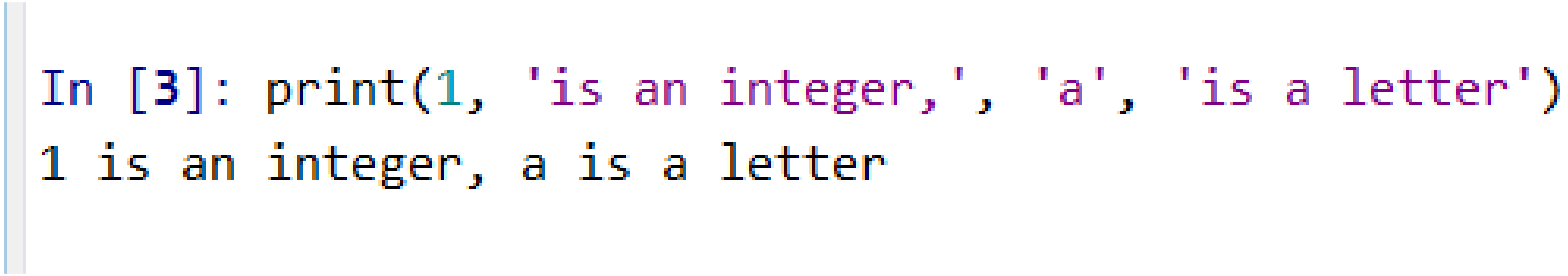
Output



```
In [5]: print('a,', 1)
a, 1
```

print

- print(object1, object2, object3, object4): to print four objects to the console

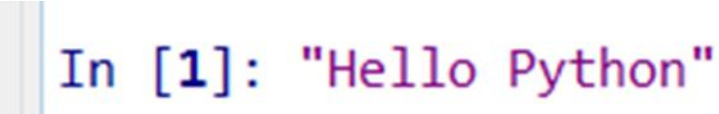


```
In [3]: print(1, 'is an integer,', 'a', 'is a letter')
1 is an integer, a is a letter
```

String in Python

- a string is a sequence of characters
- a character is anything we can type on the keyboard in one keystroke, e.g. a letter, a number, an empty space, or a backslash.

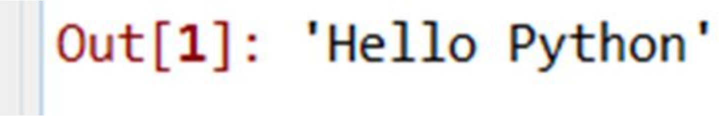
enter this line of code from the console



```
In [1]: "Hello Python"
```

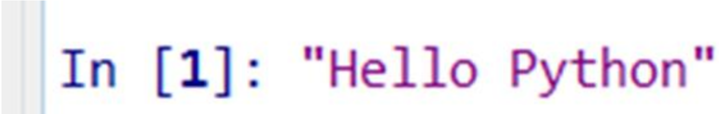
"Hello Python" is a string

see this output on the console



```
Out[1]: 'Hello Python'
```

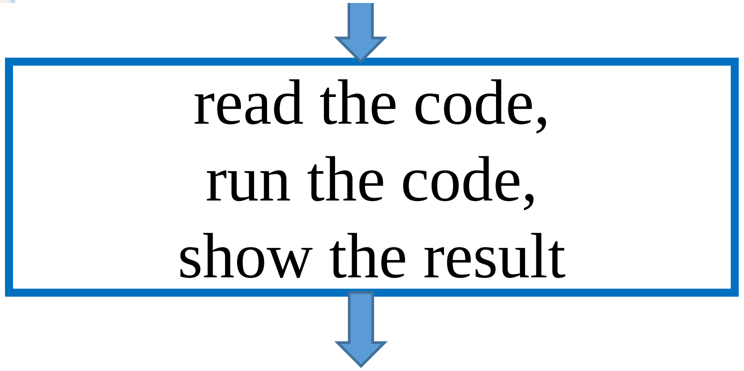
enter this line of code from the console



```
In [1]: "Hello Python"
```

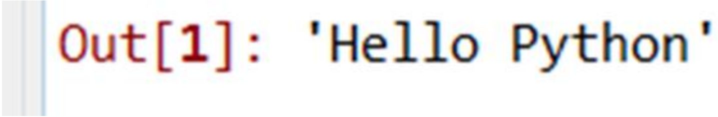
"Enhanced"
interactive interpreter for Python

IPython



read the code,
run the code,
show the result

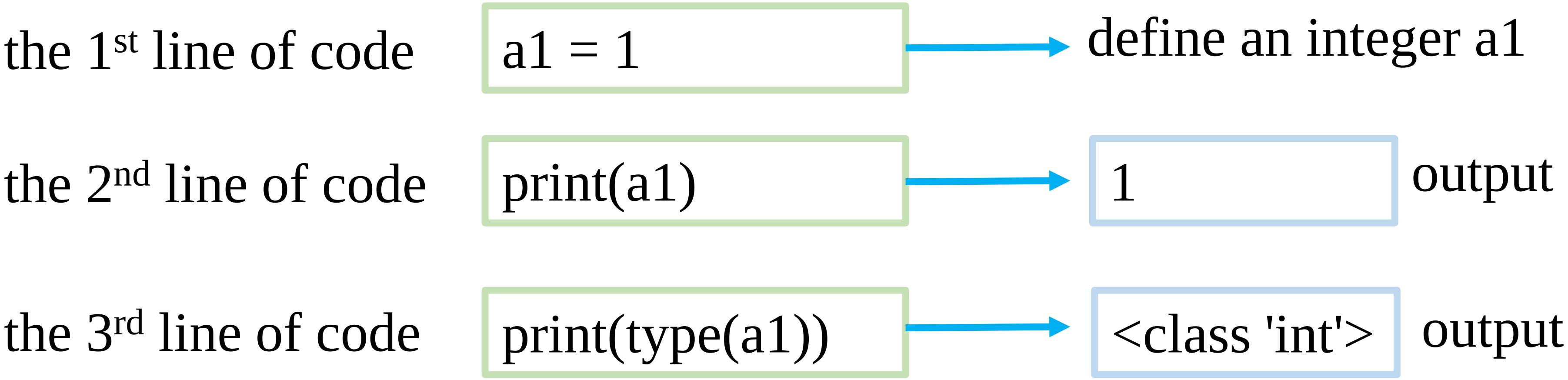
IPython shows the string



```
Out[1]: 'Hello Python'
```

Note: IPython is included in the Spyder IDE

Python program runs line by line



the 1st line of code

```
a1 = 1
```

define an integer a1

the 2nd line of code

```
print(a1)
```

```
1
```

output

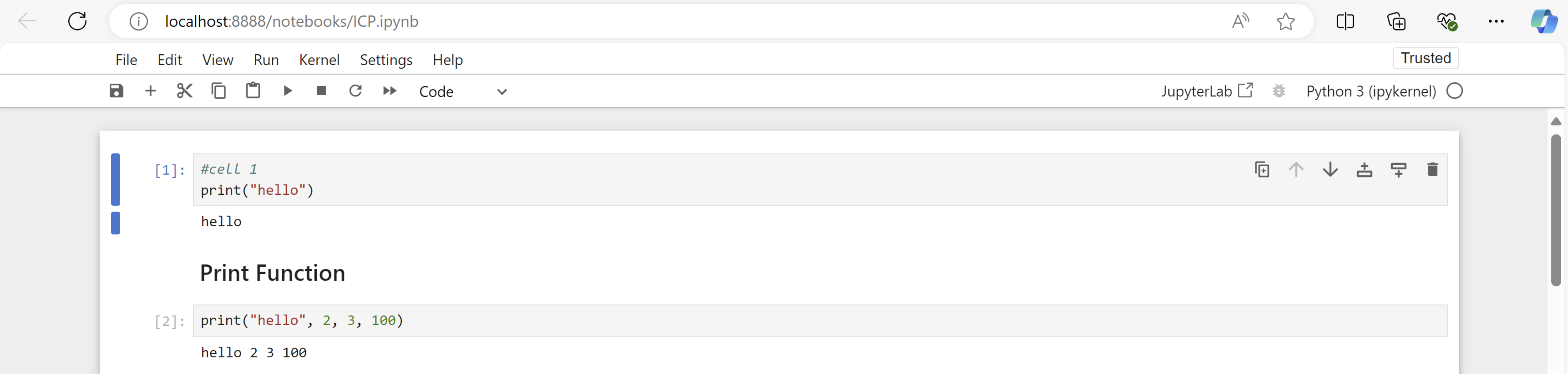
the 3rd line of code

```
print(type(a1))
```

```
<class 'int'>
```

output

Write and Run a Python program in Jupyter Notebook



[1]:
```
#cell 1
print("hello")
```

```
hello
```

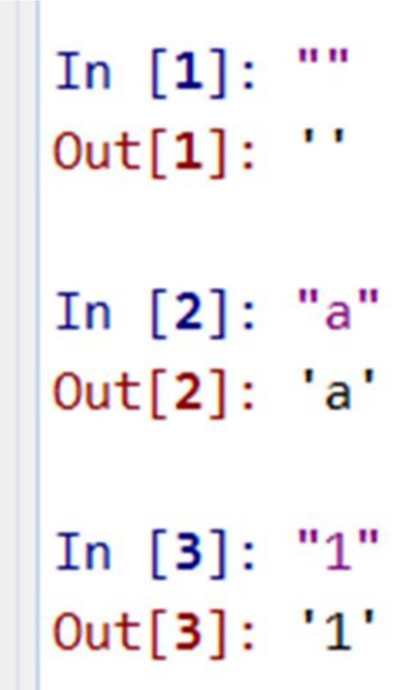
Print Function

[2]:
```
print("hello", 2, 3, 100)
```

```
hello 2 3 100
```

String in Python

- a string is a sequence of characters
- a character is anything we can type on the keyboard in one keystroke,
  e.g. a letter, a number, an empty space, or a backslash.



```
In [1]: ""
Out[1]: ''
```

an empty string

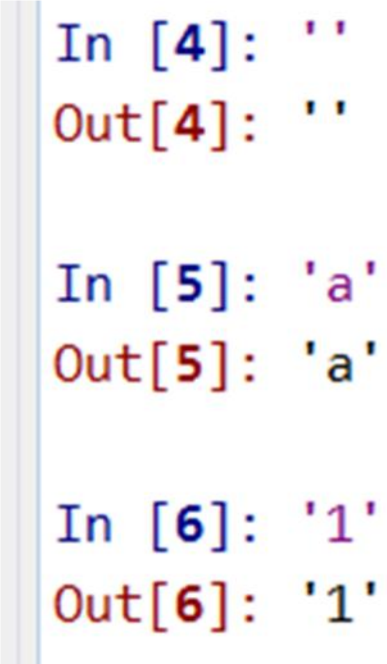
```
In [2]: "a"
Out[2]: 'a'
```

a string with a single letter

```
In [3]: "1"
Out[3]: '1'
```

a string with a single number

double quotation marks (quotes) " "



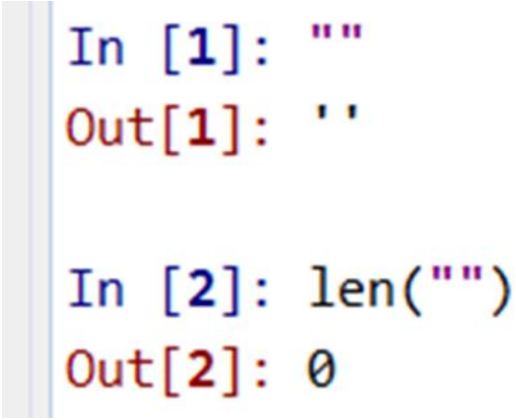
```
In [4]: ''
Out[4]: ''

In [5]: 'a'
Out[5]: 'a'

In [6]: '1'
Out[6]: '1'
```

single quotation marks (quotes) ' '

String in Python

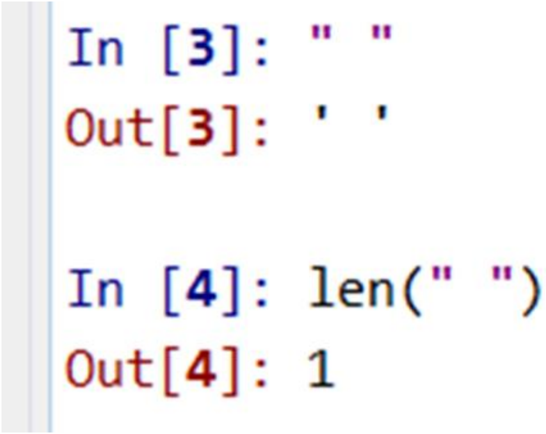
- a string is a sequence of characters
- a character is anything we can type on the keyboard in one keystroke, e.g. a letter, a number, an empty space, or a backslash.
- the length of a string is the number of characters in the string

an empty string



```
In [1]: ""
Out[1]: ''


In [2]: len("")
Out[2]: 0
```

a string with a blank space is not empty



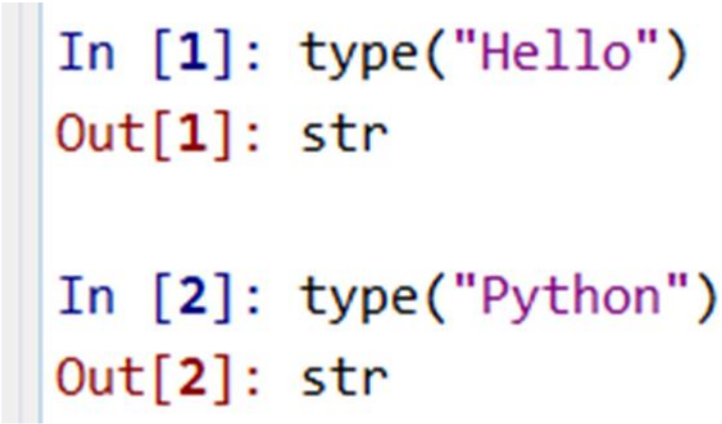
```
In [3]: " "
Out[3]: ' '


In [4]: len(" ")
Out[4]: 1
```

len(string) will show the length of the string, i.e., the number of characters

String in Python

- a string is a sequence of characters
- a character is anything we can type on the keyboard in one keystroke,
  e.g. a letter, a number, an empty space, or a backslash.
- the length of a string is the number of characters in the string
- string is a data type in Python

type(object) will show the type of the object



```
In [1]: type("Hello")
Out[1]: str

In [2]: type("Python")
Out[2]: str
```

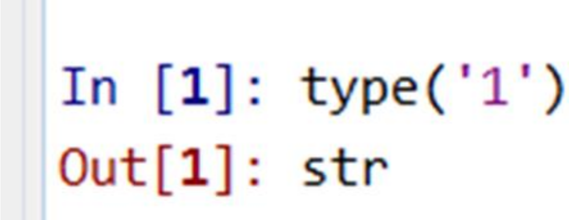
The type of "Hello" is str (i.e. string)

The type of "Python" is str (i.e. string)

String in Python

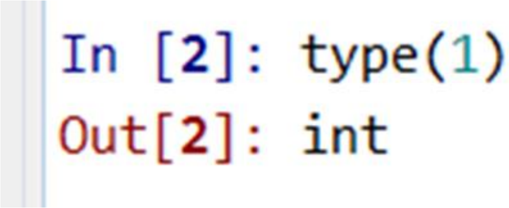
- a string is a sequence of characters
- a character is anything we can type on the keyboard in one keystroke,
  e.g. a letter, a number, an empty space, or a backslash.
- the length of a string is the number of characters in the string
- string is a data type in Python

type(object) will show the type of the object



```
In [1]: type('1')
Out[1]: str
```

'1' is a string, NOT a number in Python



```
In [2]: type(1)
Out[2]: int
```

1 is a number in Python
int is integer

Numbers in Python

real numbers in Mathematics

integer:
1, 0, -1, 123

rational number
10
1/5 = 0.5
1/3 = 0.33333333… (infinite digits)

irrational number
pi = 3.141592653… (infinite digits)

a subset of real numbers in Python

int
1, 0, -1, 123

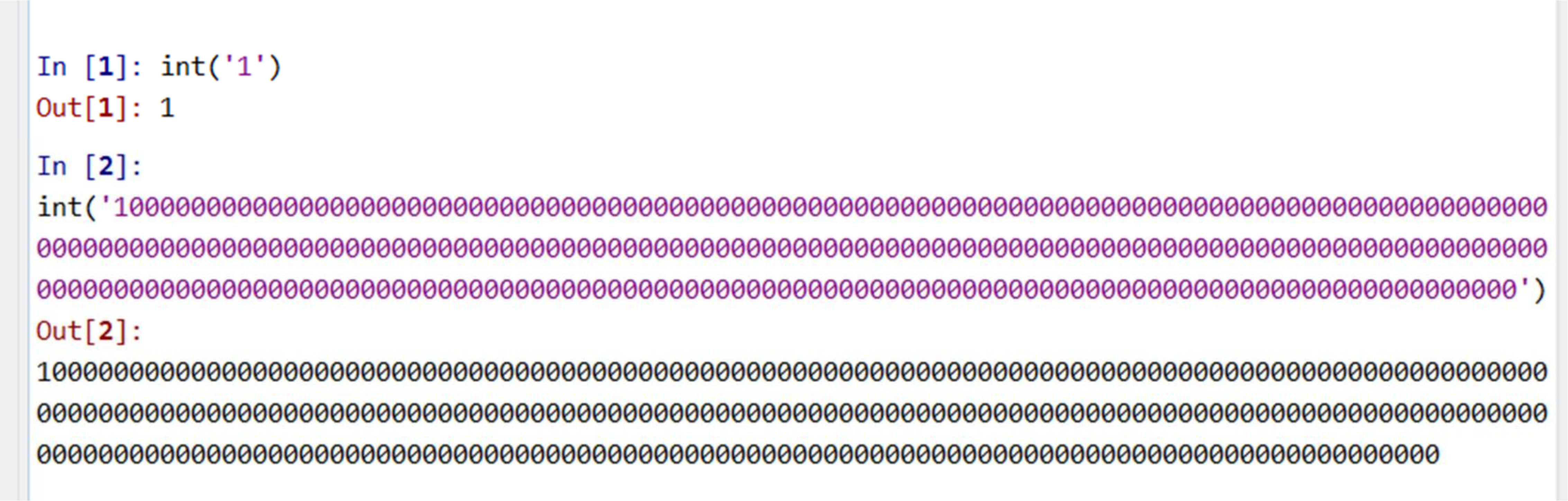
float
0.5, 0.25, 0.75, 0.125
0.3333333333333333
3.141592653589793

- float is stored as binary number (010101…) in computer
- Not every real number can be precisely represented as a float number. (e.g. 1/3, pi)
- Roughly speaking, a float can represent a real number with 15~18 significant digits

Numbers in Python

Python integer (int) can represent any integer of any length

int(string) is a function that can convert a string to an integer (int)

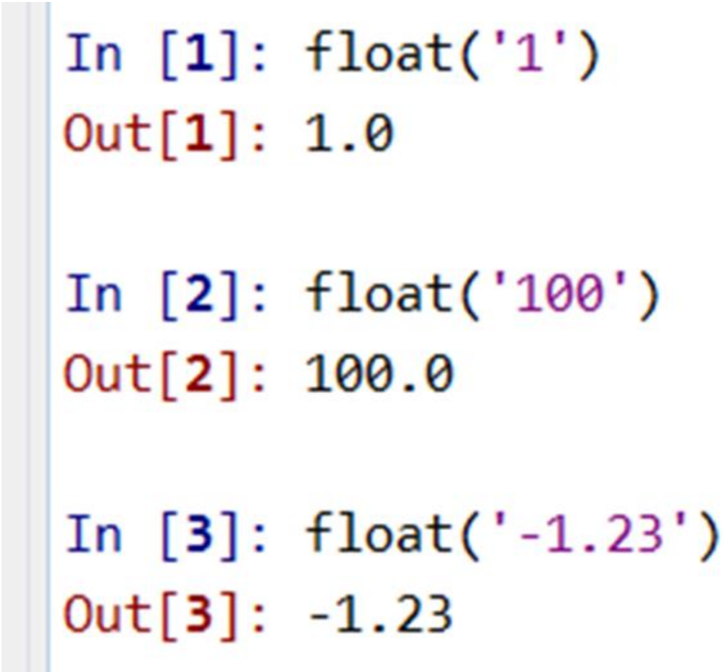


```
In [1]: int('1')
Out[1]: 1

In [2]:
int('1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000')
Out[2]:
100000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
```

Numbers in Python

Python float numbers can only represent a subset of real numbers

float(string) is a function that can convert a string to a float number

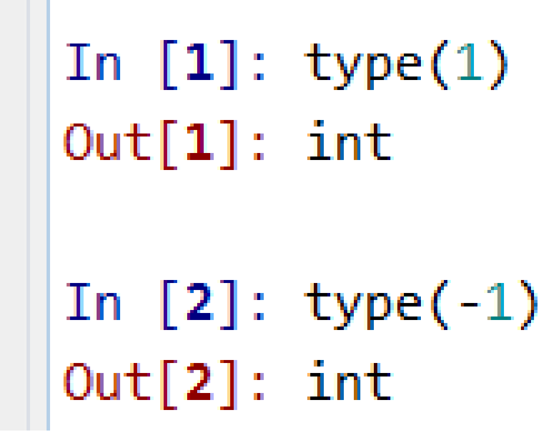


```
In [1]: float('1')
Out[1]: 1.0

In [2]: float('100')
Out[2]: 100.0

In [3]: float('-1.23')
Out[3]: -1.23
```

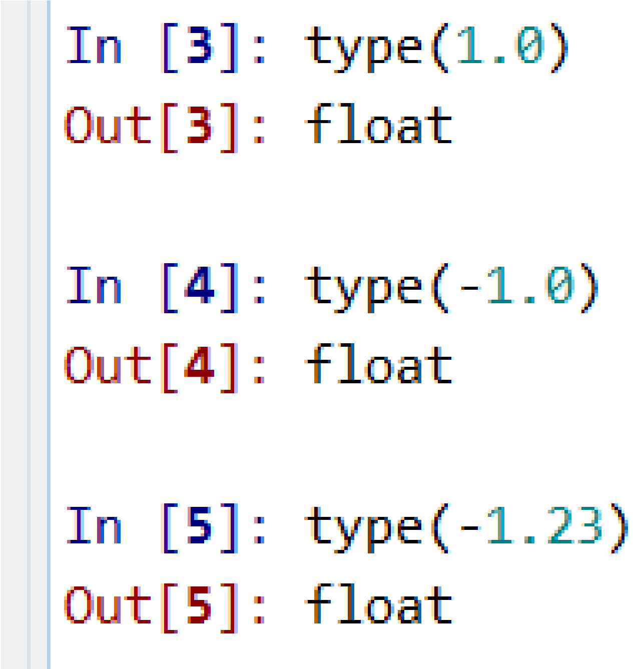
Numbers in Python

int and float are data types in Python

type(object) will show the type of the object



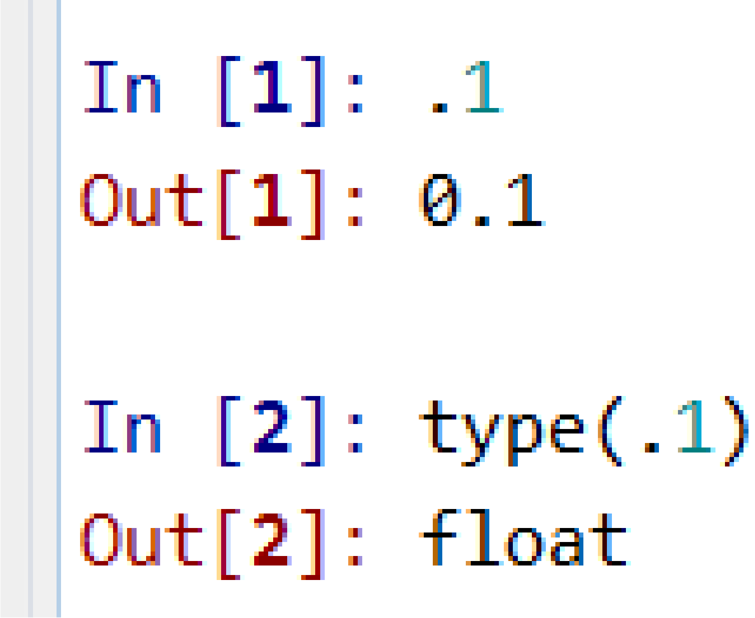
```
In [1]: type(1)
Out[1]: int

In [2]: type(-1)
Out[2]: int
```



```
In [3]: type(1.0)
Out[3]: float

In [4]: type(-1.0)
Out[4]: float

In [5]: type(-1.23)
Out[5]: float
```

Numbers in Python

0.1 can also be written as .1

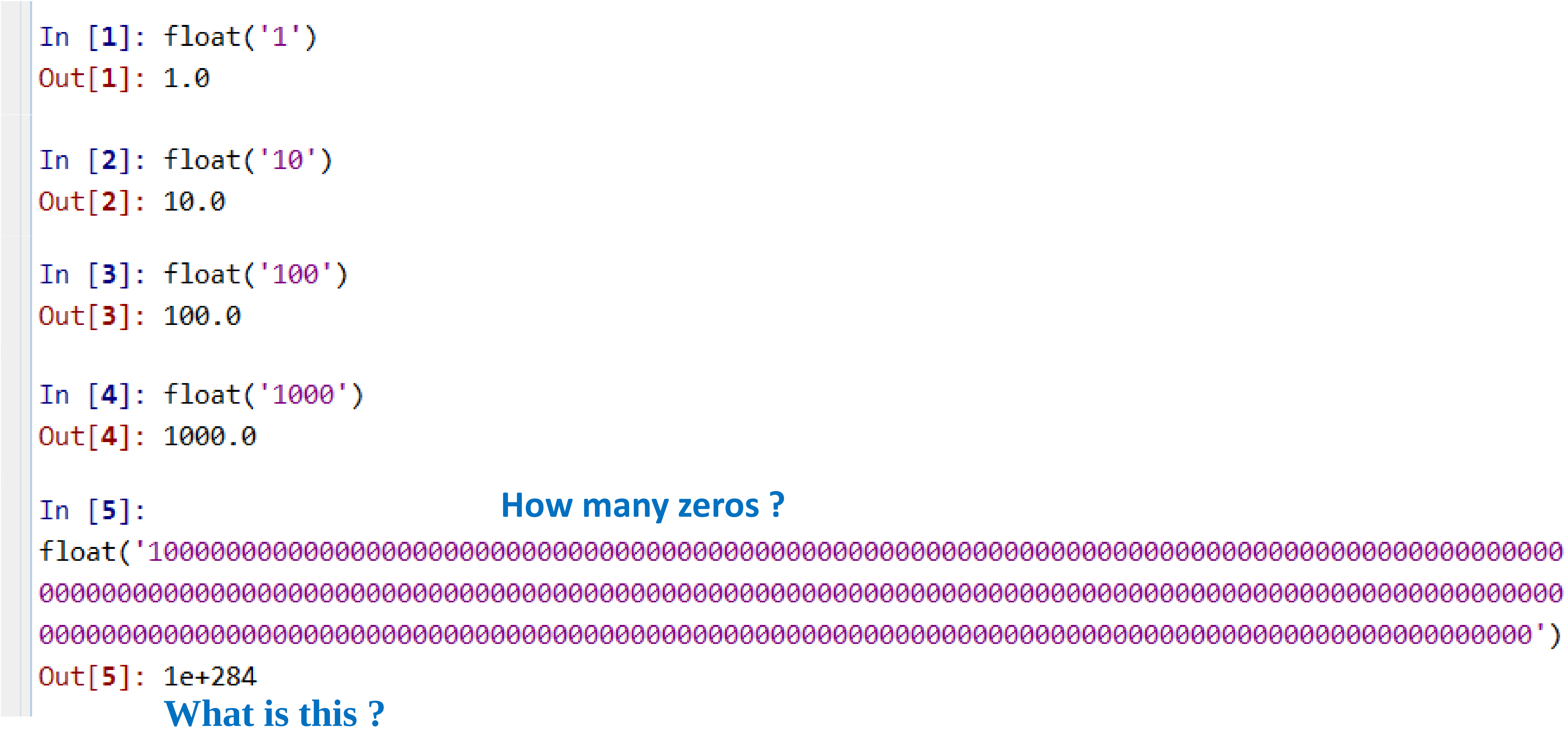


```
In [1]: .1
Out[1]: 0.1


In [2]: type(.1)
Out[2]: float
```

This may confuse you or other people,
do not use this notation

Numbers in Python

Can float represent extremely large numbers ?



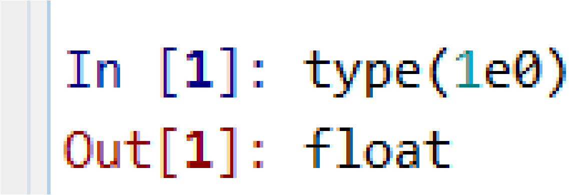
```
In [1]: float('1')
Out[1]: 1.0

In [2]: float('10')
Out[2]: 10.0

In [3]: float('100')
Out[3]: 100.0

In [4]: float('1000')
Out[4]: 1000.0

In [5]:
float('1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000')
Out[5]: 1e+284
```

How many zeros ?

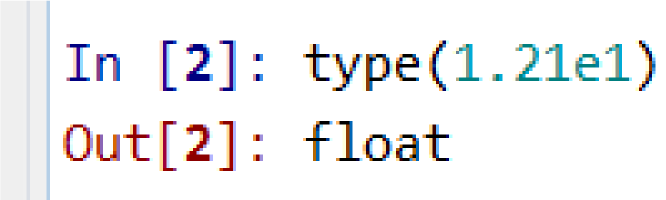
What is this ?

scientific notation of a number

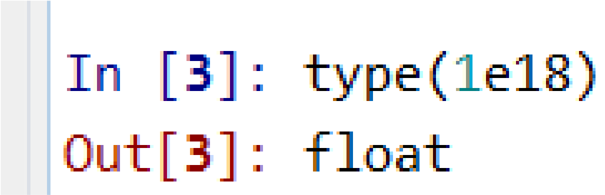
number	scientific notation	notation in Python	type in Python
1	1×10^{0}	1e0	float
12	1.2×10^{1}	1.2e1	float
123	1.23×10^{2}	1.23e2	float
1,000,000,000,000,000,000	1×10^{18}	1e18	float
0.1	1×10^{-1}	1e-1	float
0.000000000000000001	1×10^{-18}	1e-18	float



```
In [1]: type(1e0)
Out[1]: float
```



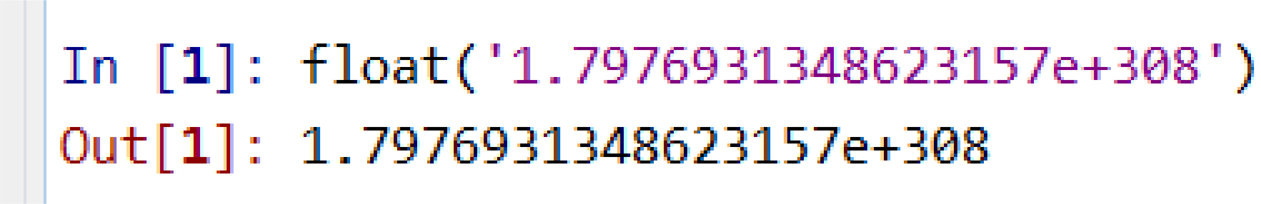
```
In [2]: type(1.21e1)
Out[2]: float
```



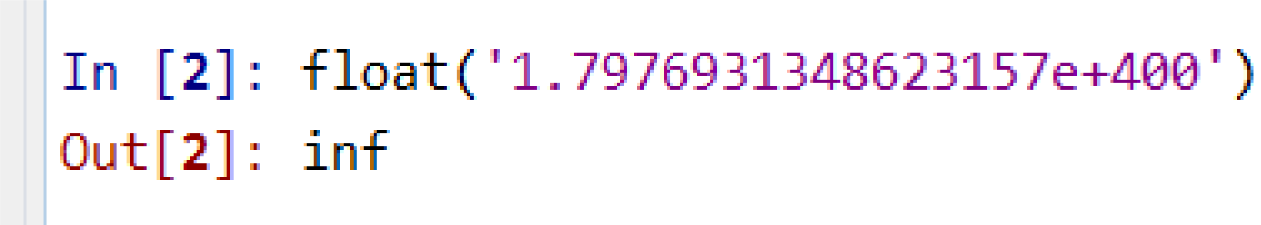
```
In [3]: type(1e18)
Out[3]: float
```

Numbers in Python

Can float represent extremely large numbers ?



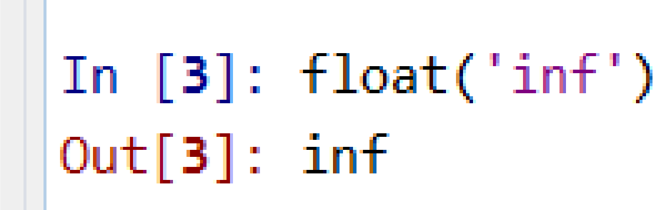
```
In [1]: float('1.7976931348623157e+308')
Out[1]: 1.7976931348623157e+308
```



```
In [2]: float('1.7976931348623157e+400')
Out[2]: inf
```

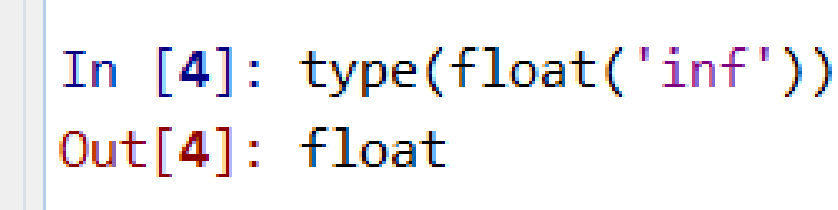
Any positive number bigger than 1.7976931348623157e+308 will be represented as **inf**
(from python documentation)

get an inf



```
In [3]: float('inf')
Out[3]: inf
```

type of inf is float

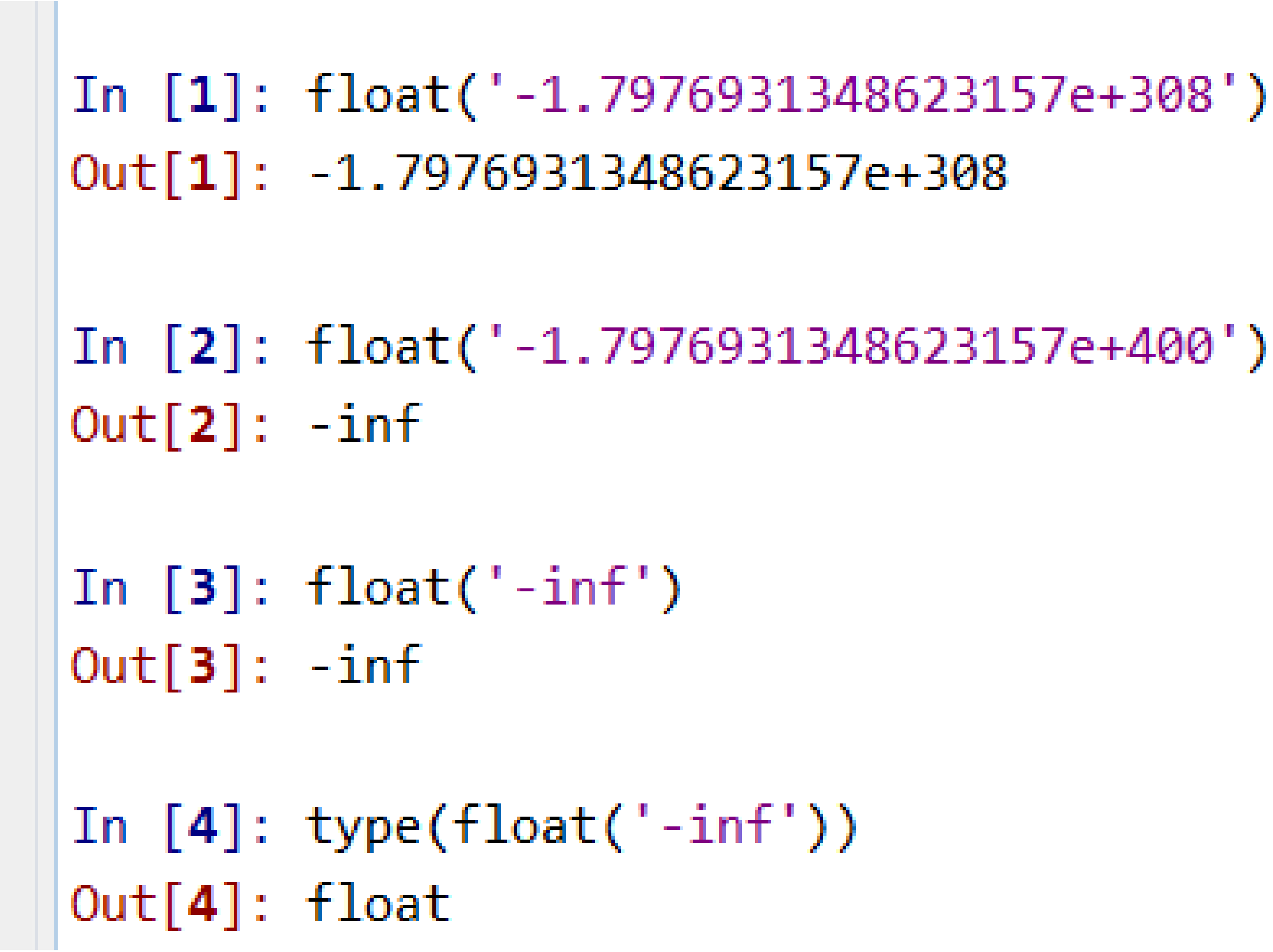


```
In [4]: type(float('inf'))
Out[4]: float
```

inf is a float to represent extremely large positive numbers

Numbers in Python

Any negative number smaller than -1.7976931348623157e+308 will be represented as **-inf**



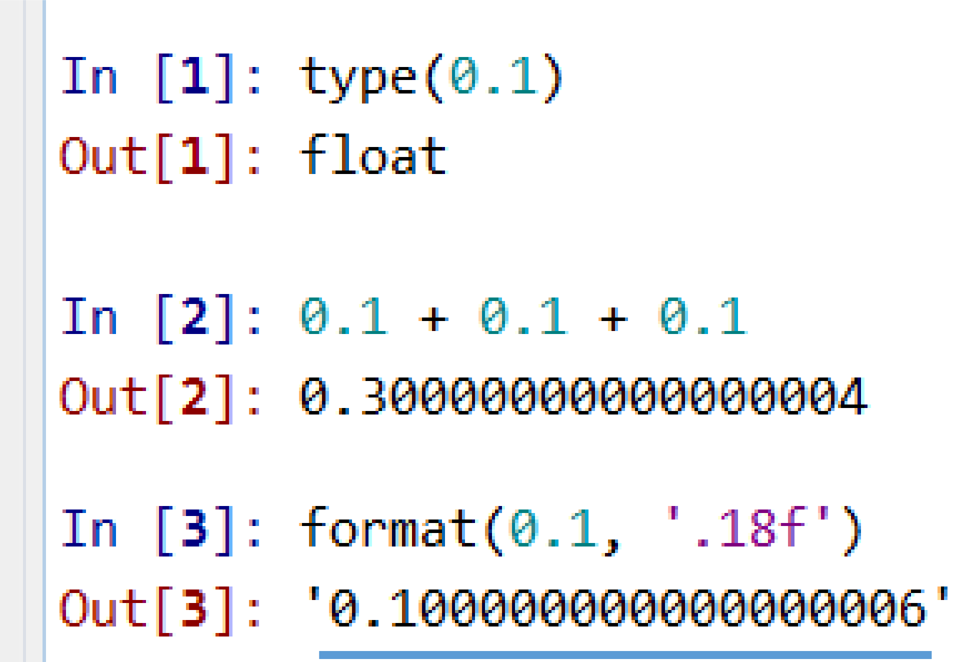
```
In [1]: float('-1.7976931348623157e+308')
Out[1]: -1.7976931348623157e+308


In [2]: float('-1.7976931348623157e+400')
Out[2]: -inf


In [3]: float('-inf')
Out[3]: -inf


In [4]: type(float('-inf'))
Out[4]: float
```

-inf is a float to represent extremely small negative numbers

Numbers in Python

- Within the range from -1.7976931348623157e+308 to 1.7976931348623157e+308, not every number can be precisely represented by a float number

- Example: 0.1 + 0.1 + 0.1 is not equal to 0.3



```
In [1]: type(0.1)
Out[1]: float

In [2]: 0.1 + 0.1 + 0.1
Out[2]: 0.30000000000000004

In [3]: format(0.1, '.18f')
Out[3]: '0.100000000000000006'
```

this is the number "0.1"
represented by a float

Float numbers are good enough for many applications that need numerical computations:

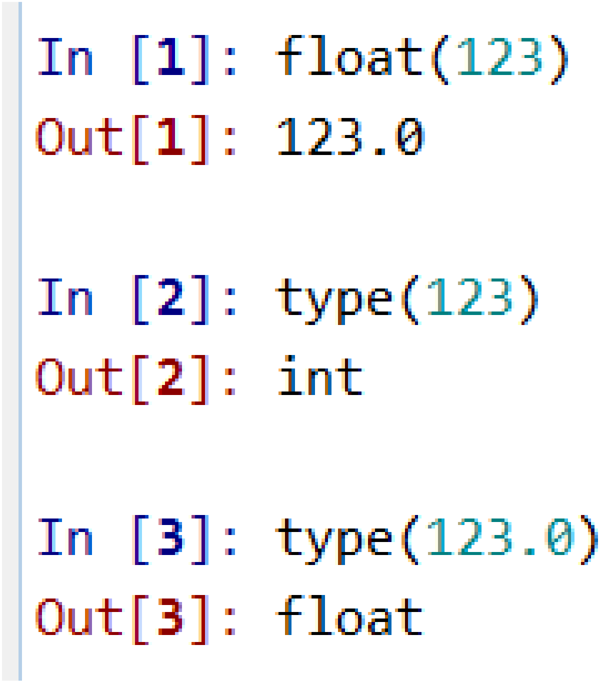
1) engineering analysis
2) machine learning

If you need high precision numbers in Python, try the decimal module.

Numbers in Python

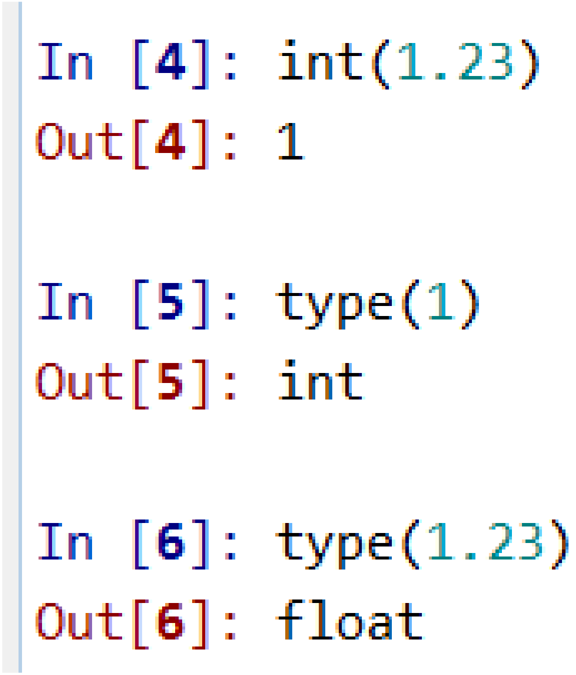
convert an **int** to a **float** or convert a **float** to an **int**

use the function float()
to convert an integer to a float



```
In [1]: float(123)
Out[1]: 123.0

In [2]: type(123)
Out[2]: int

In [3]: type(123.0)
Out[3]: float
```

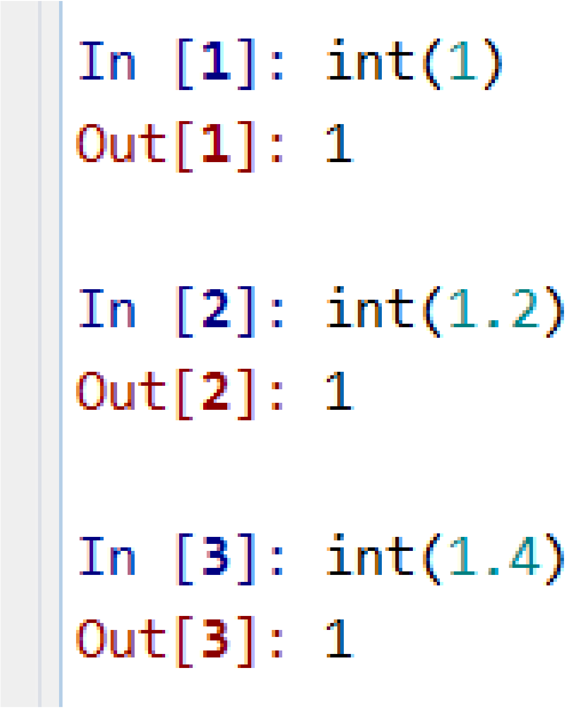
use the function int()
to convert a float to an integer



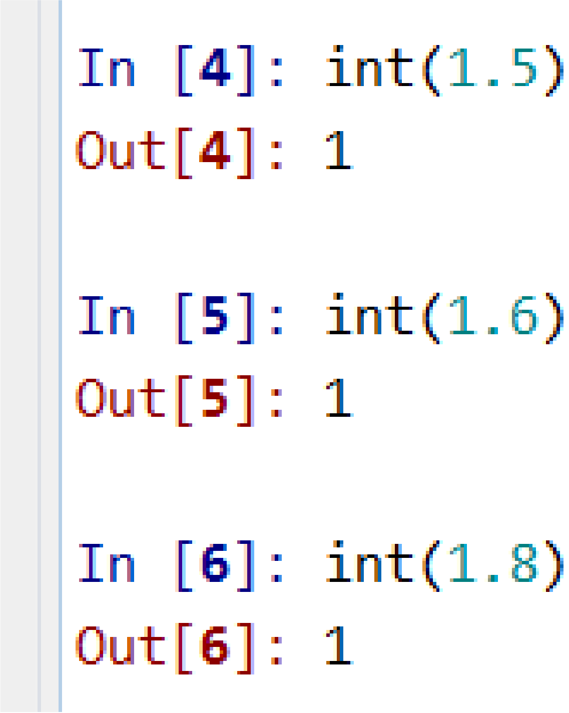
```
In [4]: int(1.23)
Out[4]: 1

In [5]: type(1)
Out[5]: int

In [6]: type(1.23)
Out[6]: float
```

Numbers in Python

the function int(x) will NOT round x to the nearest integer



```
In [1]: int(1)
Out[1]: 1

In [2]: int(1.2)
Out[2]: 1

In [3]: int(1.4)
Out[3]: 1
```

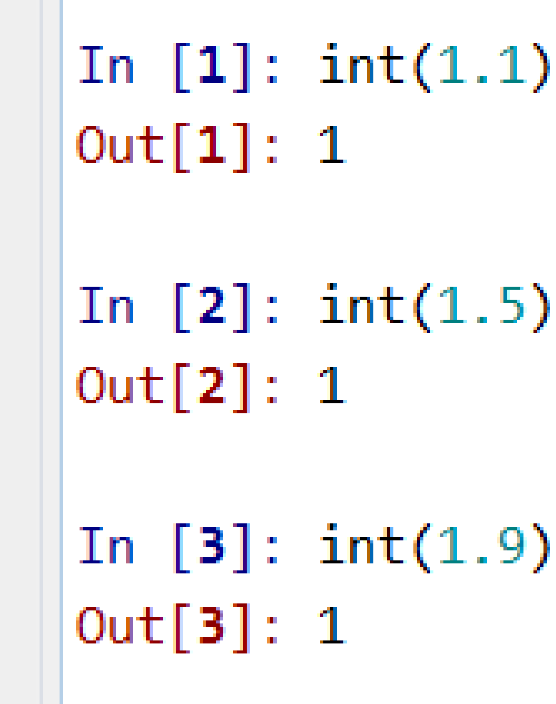


```
In [4]: int(1.5)
Out[4]: 1

In [5]: int(1.6)
Out[5]: 1

In [6]: int(1.8)
Out[6]: 1
```

Numbers in Python

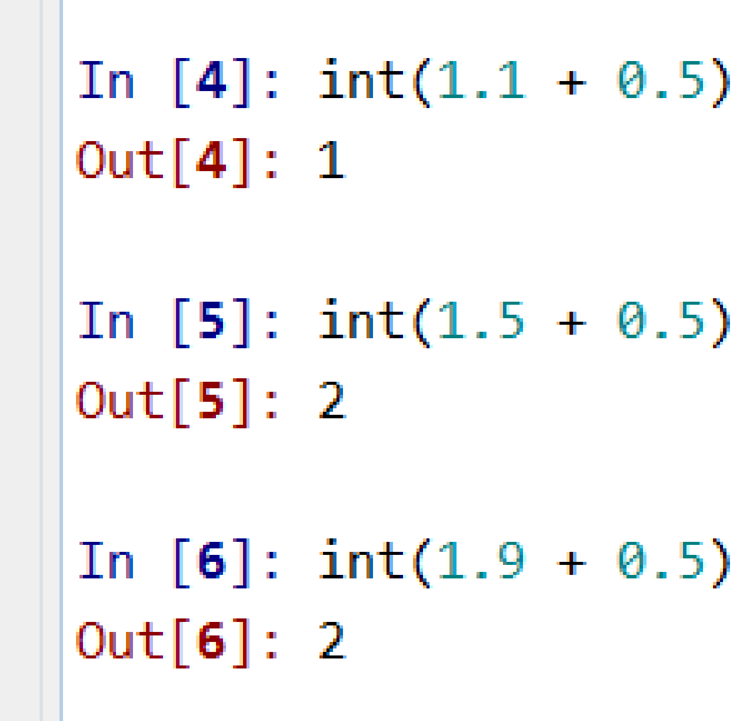
a trick to round a float number to the nearest integer

int(x)



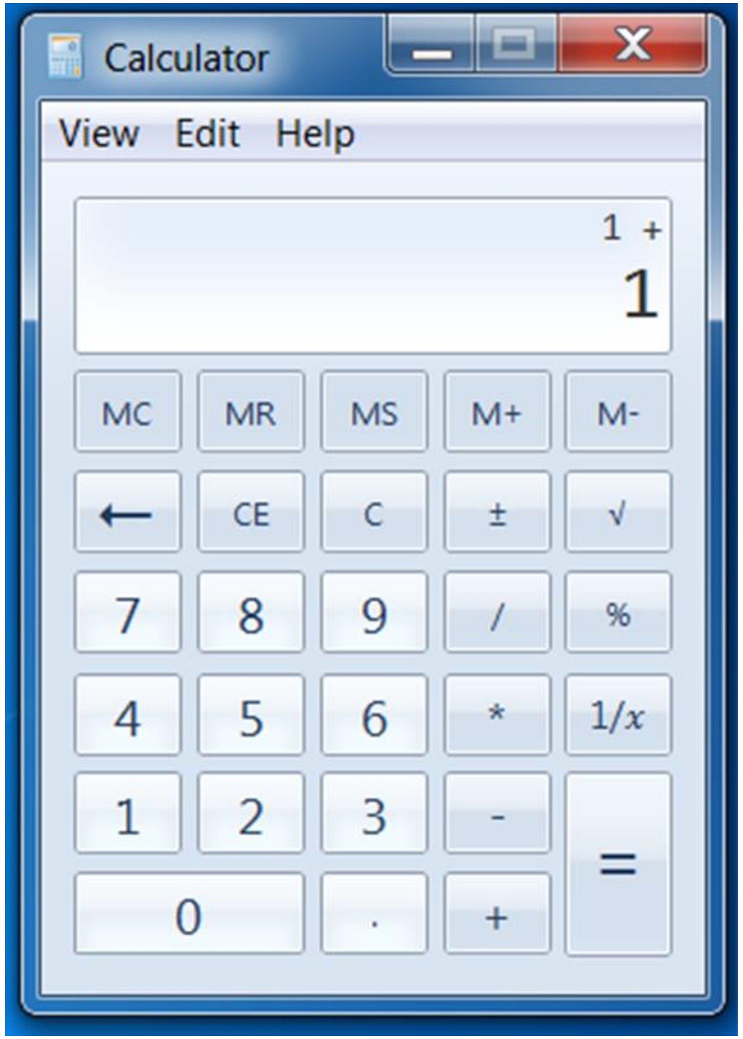
```
In [1]: int(1.1)
Out[1]: 1

In [2]: int(1.5)
Out[2]: 1

In [3]: int(1.9)
Out[3]: 1
```

int(x + 0.5)



```
In [4]: int(1.1 + 0.5)
Out[4]: 1

In [5]: int(1.5 + 0.5)
Out[5]: 2

In [6]: int(1.9 + 0.5)
Out[6]: 2
```

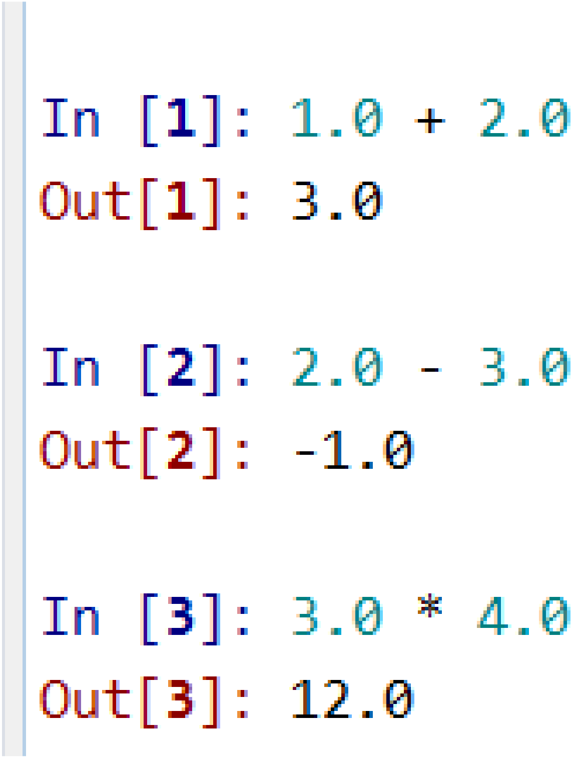
Use Python as a Calculator



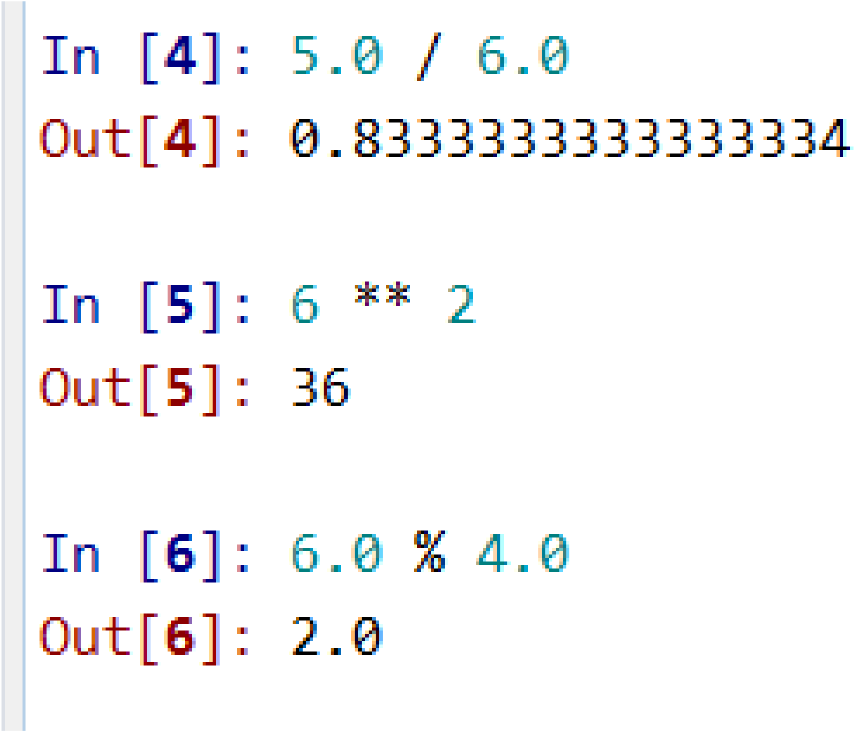
Operator	Operation
+	Addition
-	Subtraction
*	Multiplication
/	Division
**	Power
%	Remainder

Use Python as a Calculator

- The result of any basic operation on float numbers is a float number



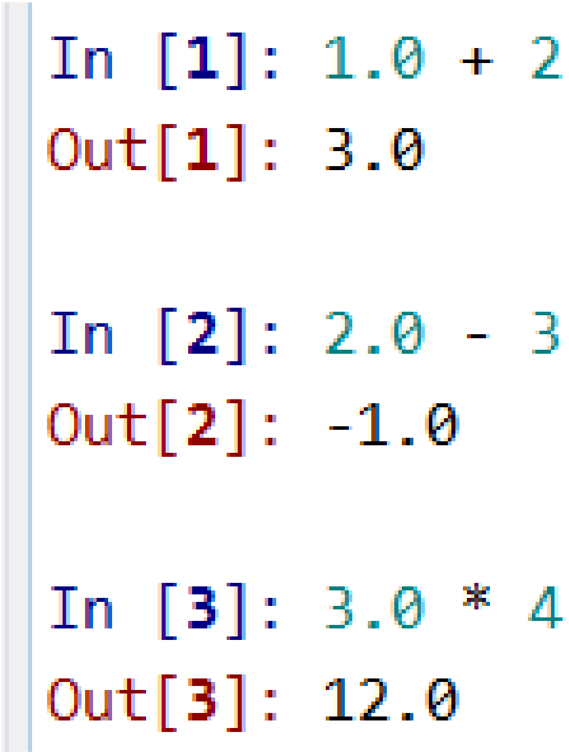
```
In [1]: 1.0 + 2.0
Out[1]: 3.0

In [2]: 2.0 - 3.0
Out[2]: -1.0

In [3]: 3.0 * 4.0
Out[3]: 12.0
```



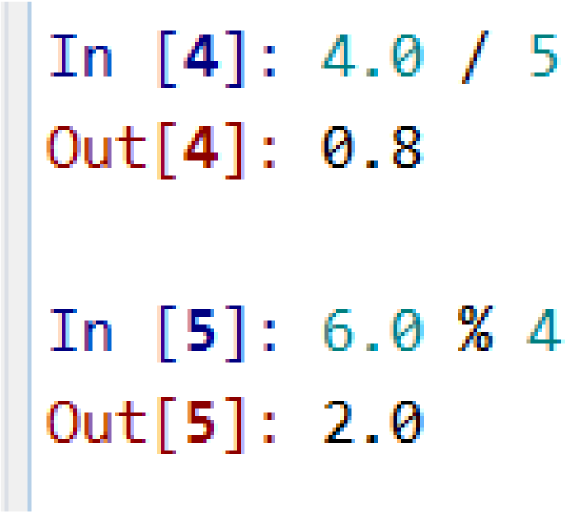
```
In [4]: 5.0 / 6.0
Out[4]: 0.8333333333333334

In [5]: 6 ** 2
Out[5]: 36

In [6]: 6.0 % 4.0
Out[6]: 2.0
```

Use Python as a Calculator

- The result of any operation between float and integer is a float number



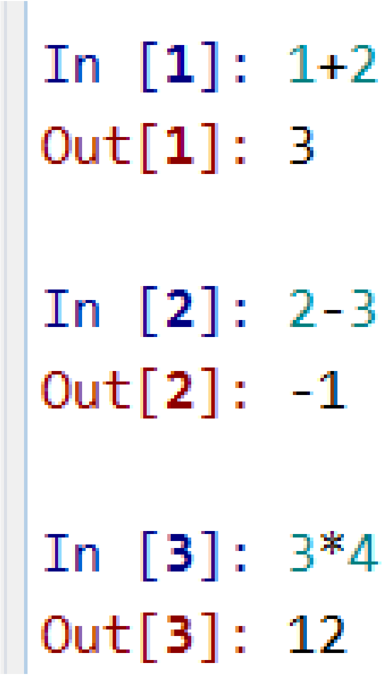
```
In [1]: 1.0 + 2
Out[1]: 3.0

In [2]: 2.0 - 3
Out[2]: -1.0

In [3]: 3.0 * 4
Out[3]: 12.0
```



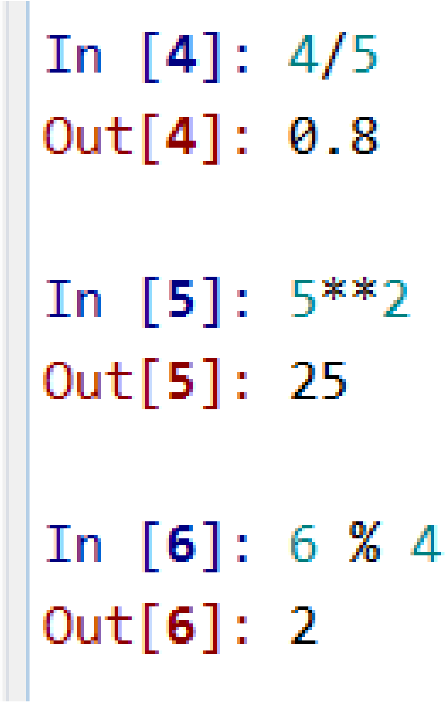
```
In [4]: 4.0 / 5
Out[4]: 0.8

In [5]: 6.0 % 4
Out[5]: 2.0
```

Use Python as a Calculator

- The result of operations between integers could be integer or float



```
In [1]: 1+2
Out[1]: 3

In [2]: 2-3
Out[2]: -1

In [3]: 3*4
Out[3]: 12
```



```
In [4]: 4/5
Out[4]: 0.8

In [5]: 5**2
Out[5]: 25

In [6]: 6 % 4
Out[6]: 2
```

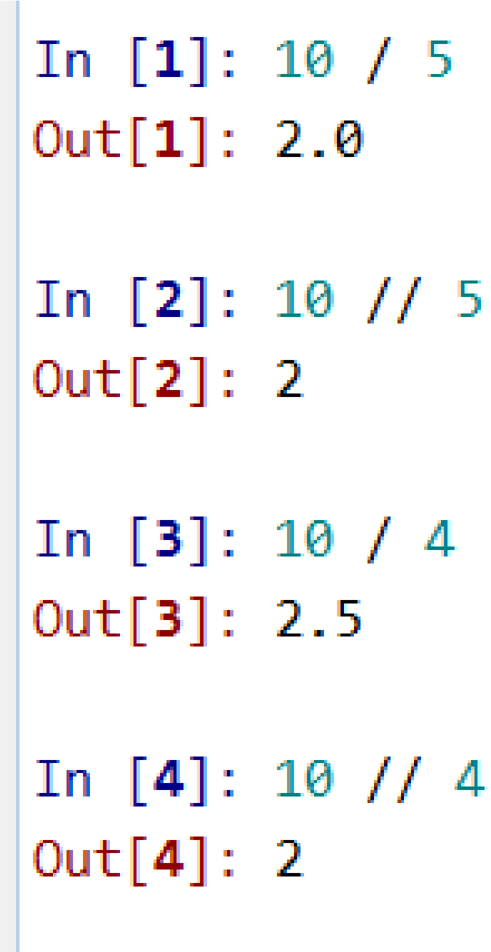
in Python 3 (or higher)
integer / integer is float

in Python 2.7
integer / integer is integer

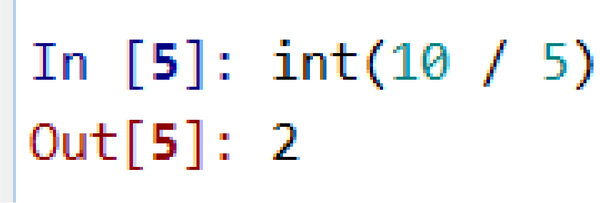
Use Python as a Calculator

The divide-divide operator //

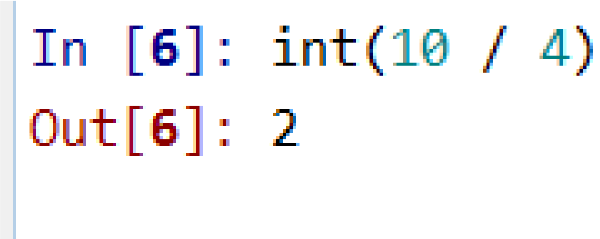
x // y is the same as int(x / y)



```
In [1]: 10 / 5
Out[1]: 2.0

In [2]: 10 // 5
Out[2]: 2

In [3]: 10 / 4
Out[3]: 2.5

In [4]: 10 // 4
Out[4]: 2
```



```
In [5]: int(10 / 5)
Out[5]: 2
```

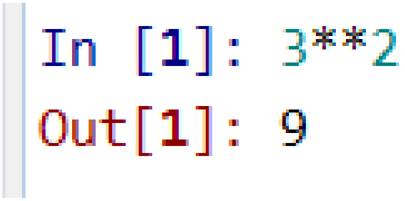


```
In [6]: int(10 / 4)
Out[6]: 2
```

Use Python as a Calculator

Power/Exponentiation operator **

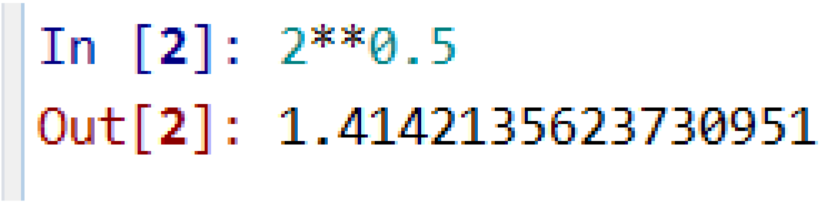
3^2 is 3 ** 2 in Python



```
In [1]: 3**2
Out[1]: 9
```

3** 2 is **Not** 3×2

$\sqrt{2}$ is 2 ** 0.5 in Python



```
In [2]: 2**0.5
Out[2]: 1.4142135623730951
```

2** 0.5 is **Not** 2×0.5

Use Python as a Calculator

- Numeric Expression: a sequence of operations on numbers

  1 + 2

  1 + 2 * 3

  An expression will be evaluated (by Python Interpreter) to a value

  1 + 2 => 3

- The Order of Evaluation is called “operator precedence”

Evaluate: 1 + 2 * 3

step1: 2*3 => 6

step2: 1 + 6 =>7

common sense…

Evaluate: 1 + 2 * 3 - 4 / 5 ** 6%2

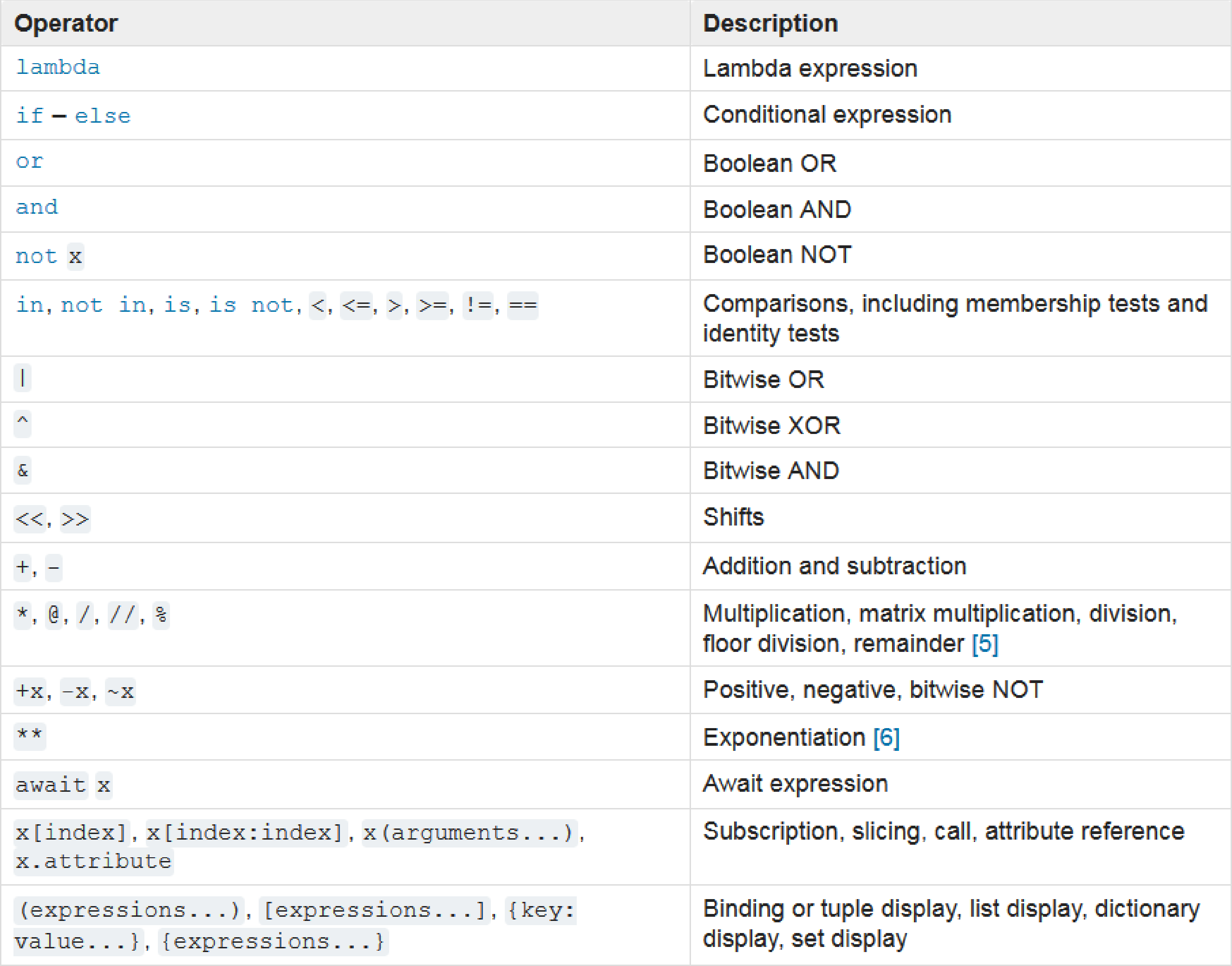
What is result of 4 / 5 ** 6%2 ?

Which operation should go first?

confused …..

Use Python as a Calculator

- The Order of Evaluation - "operator precedence"



Operator	Description
`lambda`	Lambda expression
`if` – `else`	Conditional expression
`or`	Boolean OR
`and`	Boolean AND
`not x`	Boolean NOT
`in`, `not in`, `is`, `is not`, `<`, `<=`, `>`, `>=`, `!=`, `==`	Comparisons, including membership tests and identity tests
`|`	Bitwise OR
`^`	Bitwise XOR
`&`	Bitwise AND
`<<`, `>>`	Shifts
`+`, `-`	Addition and subtraction
`*`, `@`, `/`, `//`, `%`	Multiplication, matrix multiplication, division, floor division, remainder [5]
`+x`, `-x`, `~x`	Positive, negative, bitwise NOT
`**`	Exponentiation [6]
`await x`	Await expression
`x[index]`, `x[index:index]`, `x(arguments...)`, `x.attribute`	Subscription, slicing, call, attribute reference
`(expressions...)`, `[expressions...]`, `{key: value...}`, `{expressions...}`	Binding or tuple display, list display, dictionary display, set display

https://docs.python.org/3/reference/expressions.html

lowest precedence

Memorize this table ? !!!

highest precedence

Use Python as a Calculator

- The Order of Evaluation - called “operator precedence”

do not need to memorize the whole table, use parentheses ( ) to explicitly define the order

Do not write like this:
1 + 2 * 3 - 4 / 5 ** 6%2

write like this:
1 + 2 * 3 - 4 /((5 ** 6)%2)

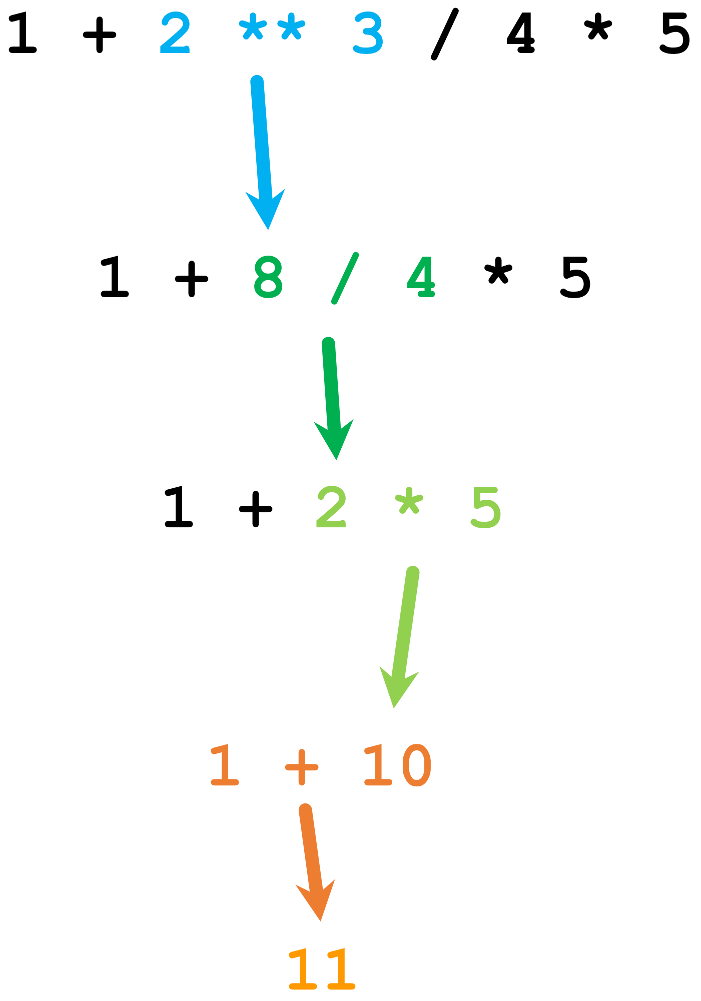
not like this:
(1 + (2 * 3)) - (4 /((5 ** 6)%2))

remember some simple rules:
Multiplication* and division / have higher priorities than addition + and subtraction -
Power ** is more powerful than multiplication, division, addition, and subtraction
Evaluation goes from the left to the right

Evaluate this Expression

```
1 + 2 ** 3 / 4 * 5
```

Parenthesis
Power
Multiplication
Addition
Left to Right

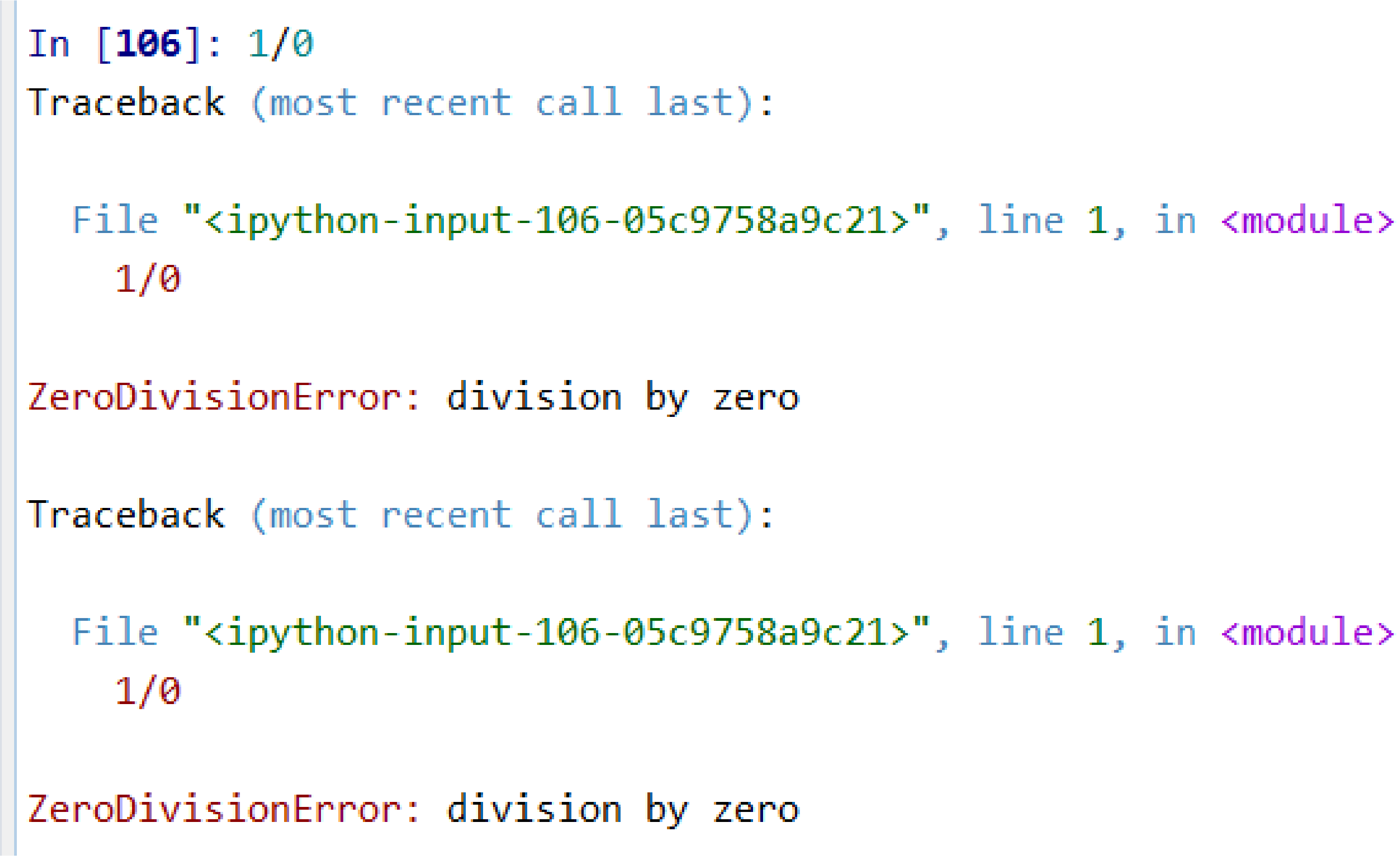


```
1 + 2 ** 3 / 4 * 5
1 + 8 / 4 * 5
1 + 2 * 5
1 + 10
11
```

Use Python as a Calculator

Special Cases: **1/0**, 1/inf, inf/inf, 0*inf

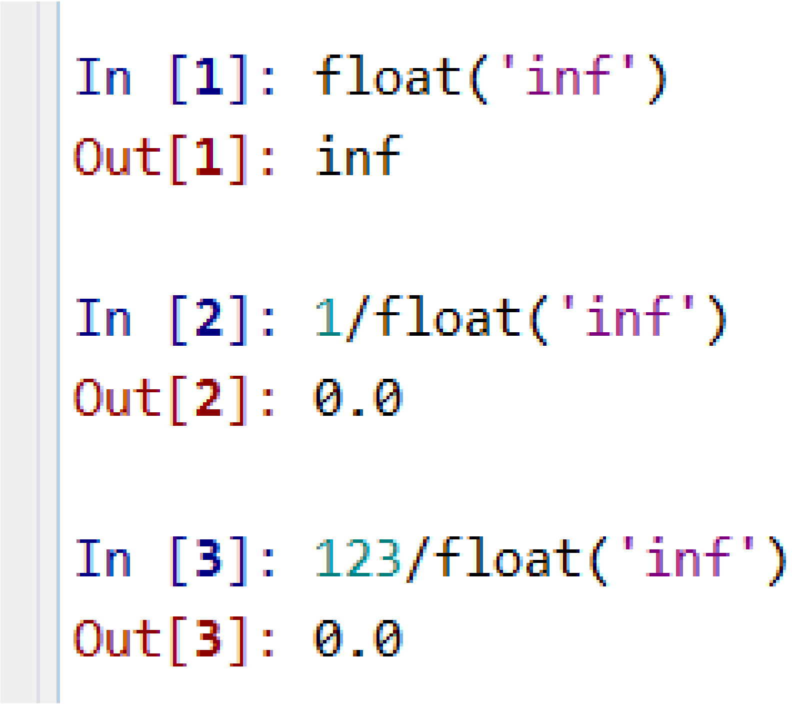
Crash !



```
In [106]: 1/0
Traceback (most recent call last):

  File "<ipython-input-106-05c9758a9c21>", line 1, in <module>
    1/0

ZeroDivisionError: division by zero

Traceback (most recent call last):

  File "<ipython-input-106-05c9758a9c21>", line 1, in <module>
    1/0

ZeroDivisionError: division by zero
```

Use Python as a Calculator

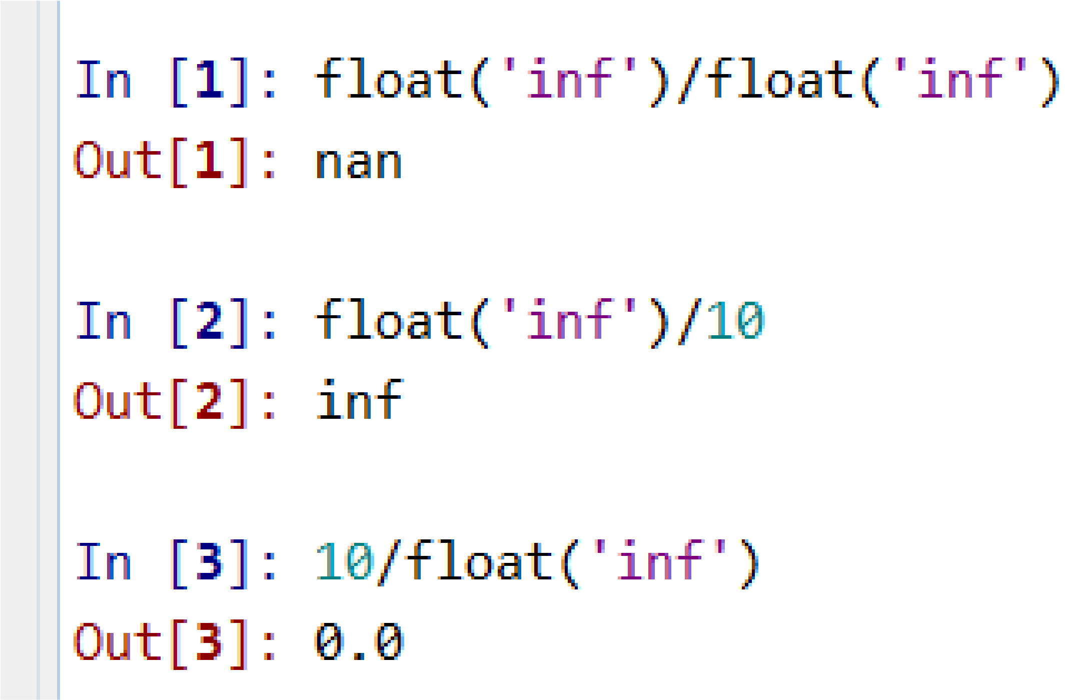
Special Cases: 1/0 = Crash, **1/inf**, inf/inf, 0*inf



```
In [1]: float('inf')
Out[1]: inf

In [2]: 1/float('inf')
Out[2]: 0.0

In [3]: 123/float('inf')
Out[3]: 0.0
```

Use Python as a Calculator

Special Cases: 1/0 = Crash, 1/inf = 0, **inf/inf**, 0*inf

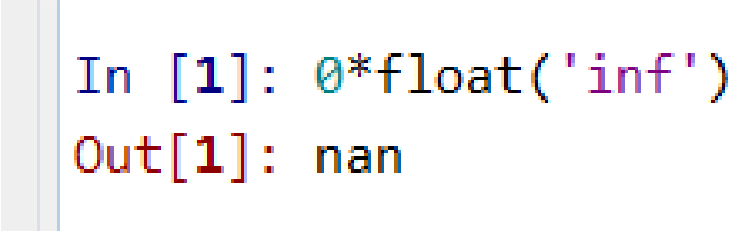


```
In [1]: float('inf')/float('inf')
Out[1]: nan

In [2]: float('inf')/10
Out[2]: inf

In [3]: 10/float('inf')
Out[3]: 0.0
```

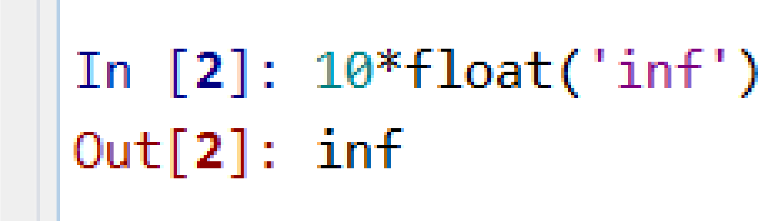
nan: Not A Number

Use Python as a Calculator

Special Cases: 1/0 = Crash, 1/inf = 0, inf/inf=nan, **0*inf**



```
In [1]: 0*float('inf')
Out[1]: nan
```



```
In [2]: 10*float('inf')
Out[2]: inf
```

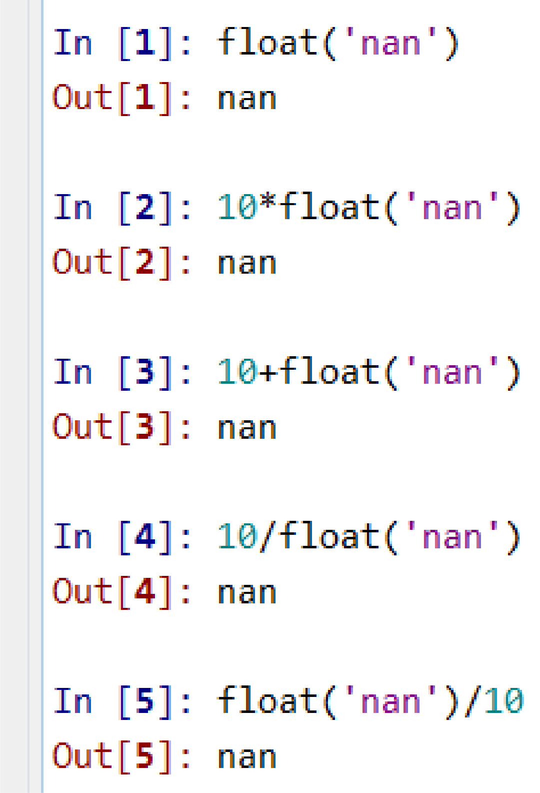
nan: Not A Number

nan infection

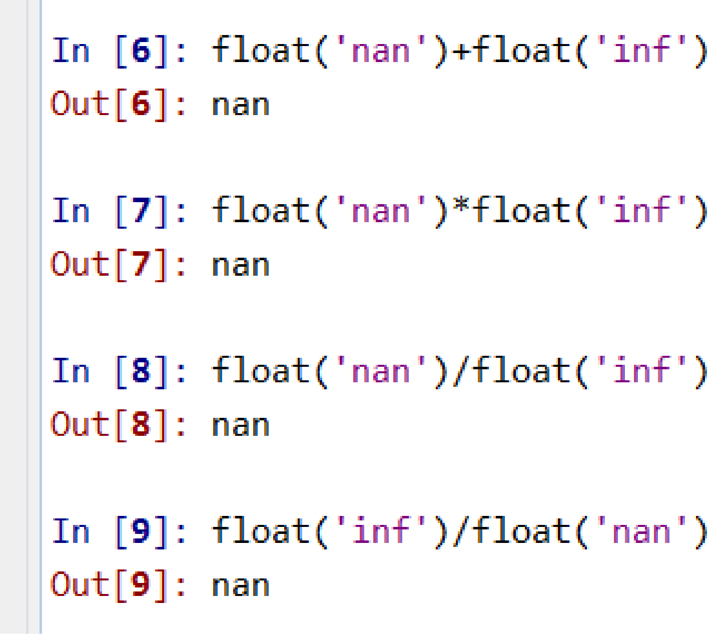
Special Cases: **1/0 = Crash, 1/inf = 0, inf/inf = nan, 0*inf = nan**

nan is like a zombie:

every number ‘touched’ by nan will turn into nan, except (nan/0 : crash)



```
In [1]: float('nan')
Out[1]: nan

In [2]: 10*float('nan')
Out[2]: nan

In [3]: 10+float('nan')
Out[3]: nan

In [4]: 10/float('nan')
Out[4]: nan

In [5]: float('nan')/10
Out[5]: nan
```



```
In [6]: float('nan')+float('inf')
Out[6]: nan

In [7]: float('nan')*float('inf')
Out[7]: nan

In [8]: float('nan')/float('inf')
Out[8]: nan

In [9]: float('inf')/float('nan')
Out[9]: nan
```

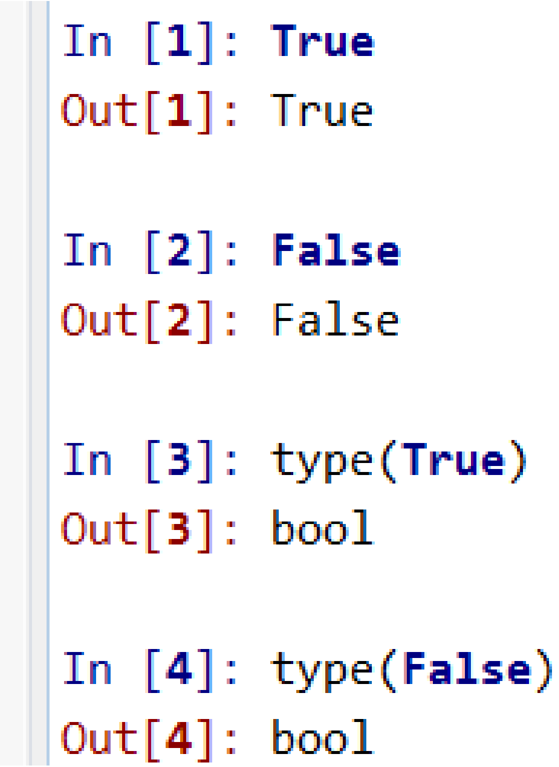
nan infection could be a big problem in numerical computing

It can only be solved after locating "the first zombie" in the program.

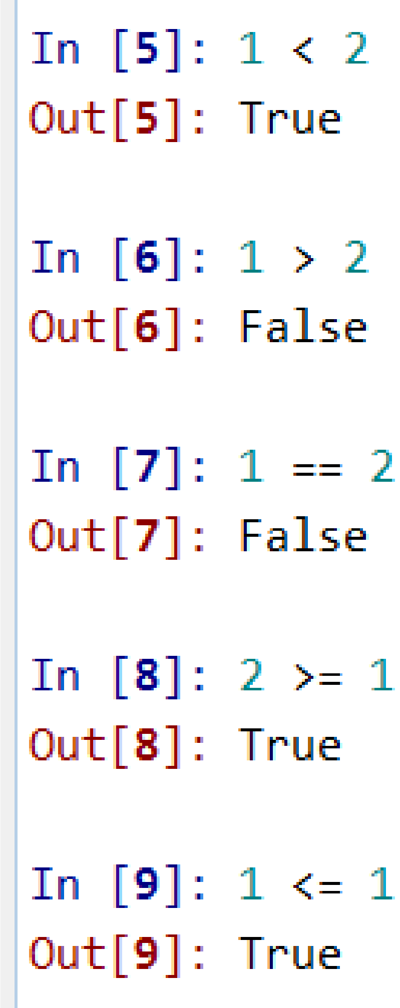
Boolean Expression

Python has a Boolean (bool) data type.
a Boolean value is True or False

a Boolean Expression is an expression that is evaluated (by Python) to True or False.



```
In [1]: True
Out[1]: True

In [2]: False
Out[2]: False

In [3]: type(True)
Out[3]: bool

In [4]: type(False)
Out[4]: bool
```



```
In [5]: 1 < 2
Out[5]: True

In [6]: 1 > 2
Out[6]: False

In [7]: 1 == 2
Out[7]: False

In [8]: 2 >= 1
Out[8]: True

In [9]: 1 <= 1
Out[9]: True
```

Boolean Expression

operators

<
>
<=
>=
==
!=
and
or

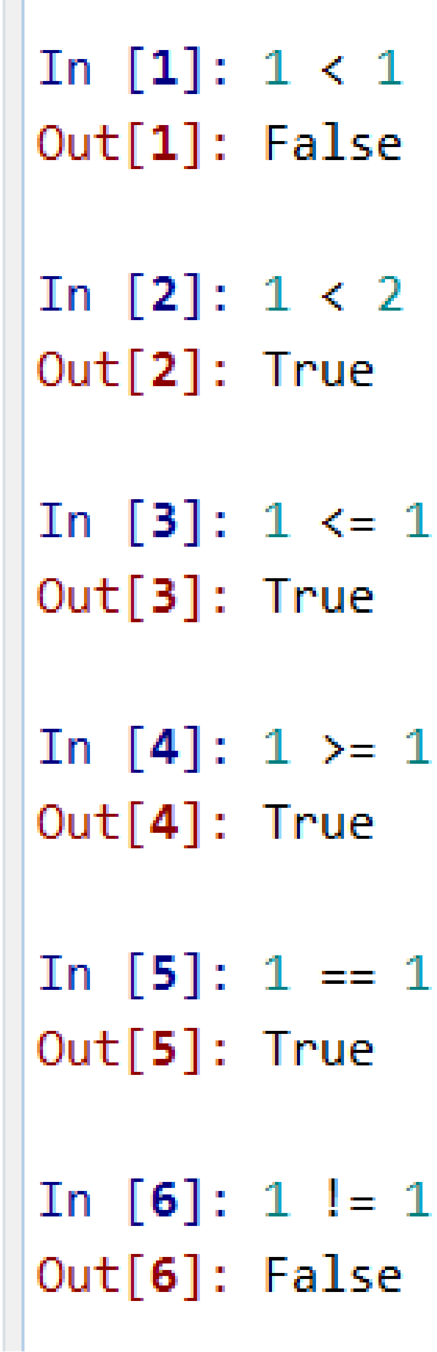
a < b : a is less than b

a <=b: a is less than or equal to b

a > b : a is greater than b

a >=b: a is greater than or equal to b

and is not &&
or is not ||



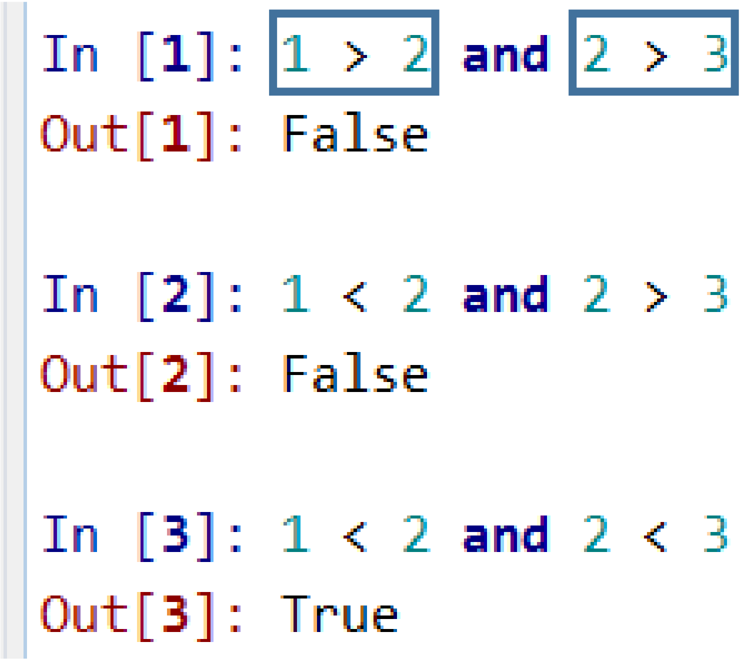
```
In [1]: 1 < 1
Out[1]: False

In [2]: 1 < 2
Out[2]: True

In [3]: 1 <= 1
Out[3]: True

In [4]: 1 >= 1
Out[4]: True

In [5]: 1 == 1
Out[5]: True

In [6]: 1 != 1
Out[6]: False
```

Boolean Expression

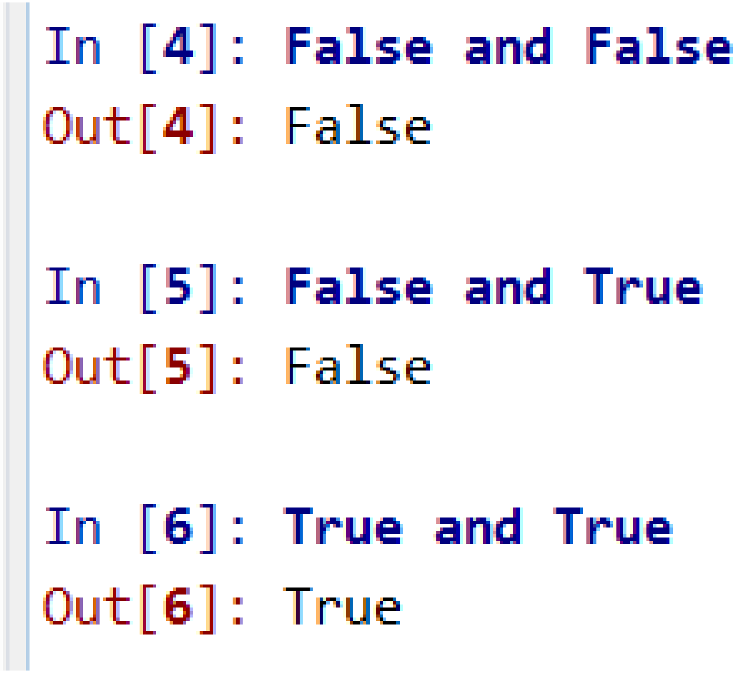
operators

<
>
<=
>=
==
!=
and
or

False False



```
In [1]: 1 > 2 and 2 > 3
Out[1]: False

In [2]: 1 < 2 and 2 > 3
Out[2]: False

In [3]: 1 < 2 and 2 < 3
Out[3]: True
```

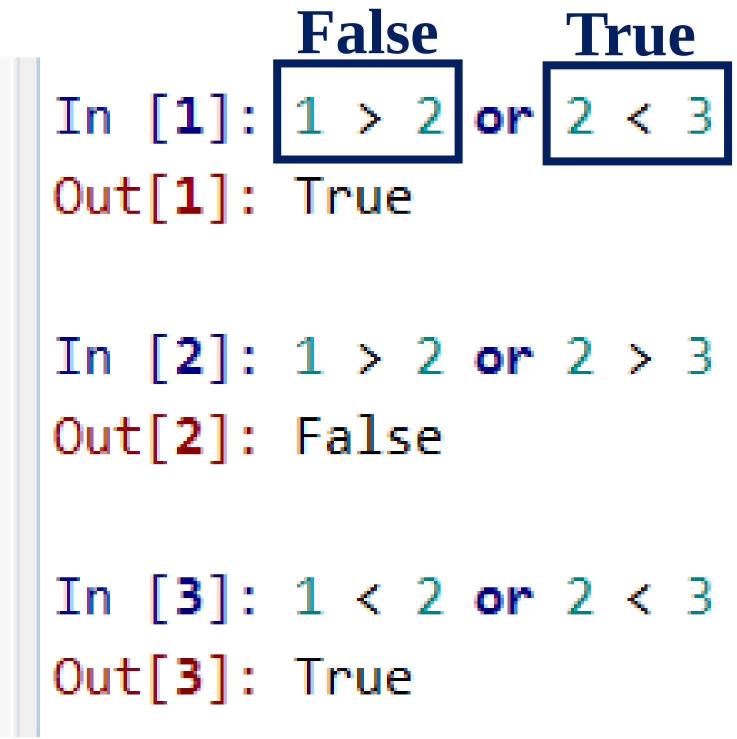


```
In [4]: False and False
Out[4]: False

In [5]: False and True
Out[5]: False

In [6]: True and True
Out[6]: True
```

Boolean Expression

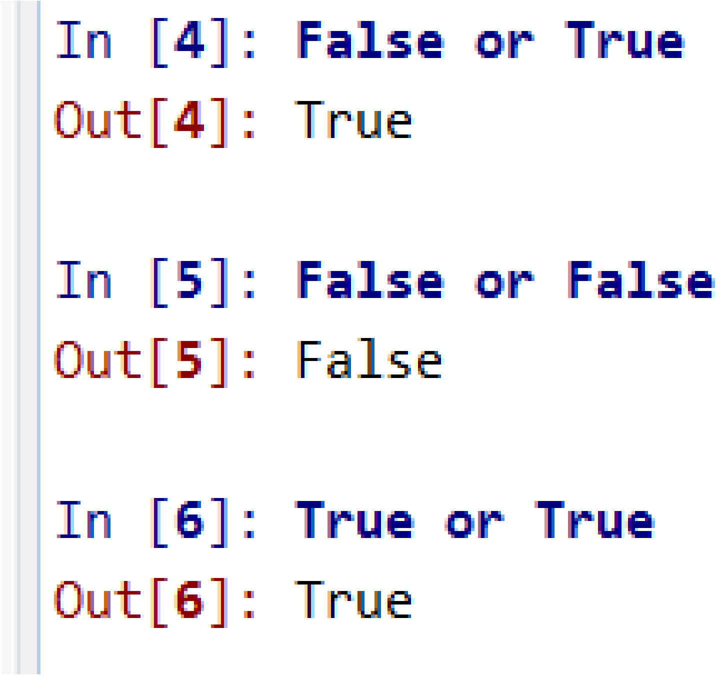
operators

<
>
<=
>=
==
!=
and
or



False
True

```
In [1]: 1 > 2 or 2 < 3
Out[1]: True

In [2]: 1 > 2 or 2 > 3
Out[2]: False

In [3]: 1 < 2 or 2 < 3
Out[3]: True
```

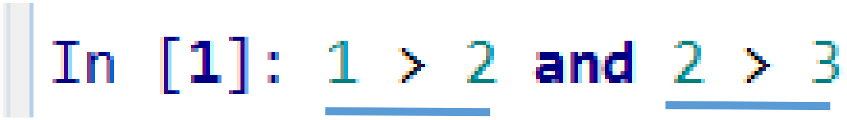


```
In [4]: False or True
Out[4]: True

In [5]: False or False
Out[5]: False

In [6]: True or True
Out[6]: True
```

Boolean Expression

operators

<
>
<=
>=
==
!=
and
or



```
In [1]: 1 > 2 and 2 > 3
```

False and False

False

Evaluate the expression: 1 > 2
Evaluate the expression : 2 > 3
Evaluate the expression : False and False

If the logic is complicated, use parentheses () to ensure the order of evaluations.
this is confusing

a > b **or** a > c **and** b > c   which expression to evaluate first ?

this is clear

(a > b or a > c) and b > c

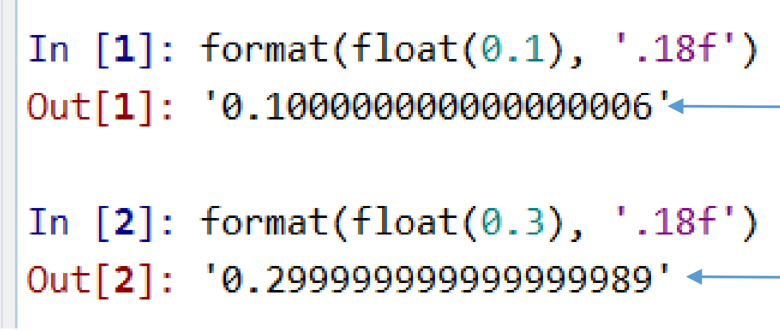
this is also clear

a > b or (a > c and b > c)

Compare Float Numbers

Compare 0.3 with 0.1 + 0.1 + 0.1

note: 0.1 can not be precisely represented by a float number



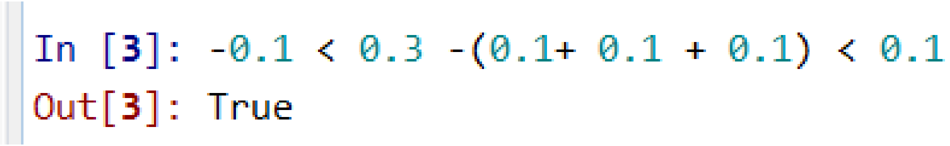
```
In [1]: format(float(0.1), '.18f')
Out[1]: '0.100000000000000006'

In [2]: format(float(0.3), '.18f')
Out[2]: '0.299999999999999989'
```

this is the number that float(0.1) actually represents

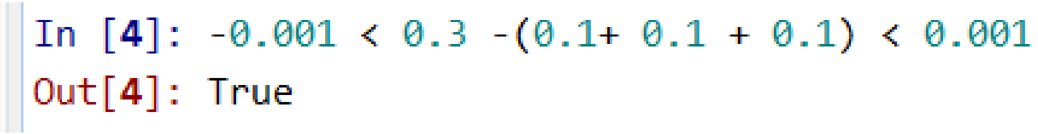
this is the number that float(0.3) actually represents

How close is the result of 0.1 + 0.1 + 0.1 to the number 0.3 ?

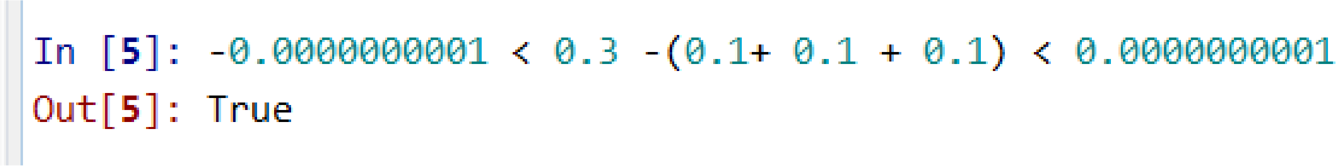
error = 0.3 - (0.1 + 0.1 + 0.1)



```
In [3]: -0.1 < 0.3 -(0.1+ 0.1 + 0.1) < 0.1
Out[3]: True
```



```
In [4]: -0.001 < 0.3 -(0.1+ 0.1 + 0.1) < 0.001
Out[4]: True
```



```
In [5]: -0.0000000001 < 0.3 -(0.1+ 0.1 + 0.1) < 0.0000000001
Out[5]: True
```

What is the value/output of this expression ?

0.3 == 0.1 + 0.1 + 0.1

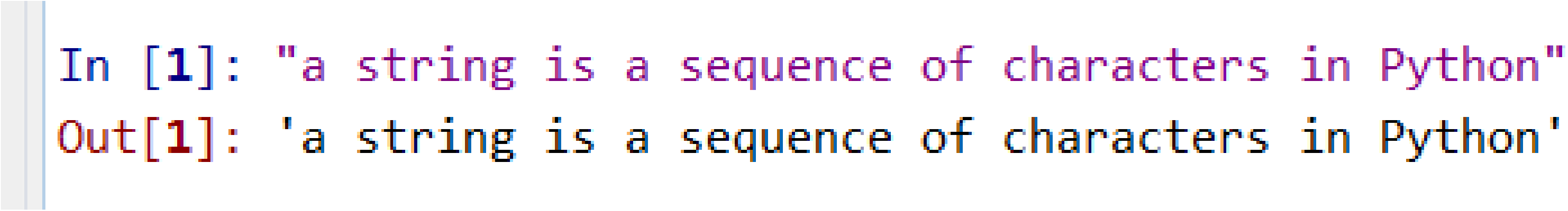
(try this on your computer)

Basic Data Types

type	description
int	integer number
float	real number within a finite range
string	a sequence of characters
list	a sequence of objects
tuple	a sequence of objects
range	a sequence of integers
set	a collection of unique objects
dictionary	a collection of key : value pairs

String in Python

- a string is a sequence of characters in Python



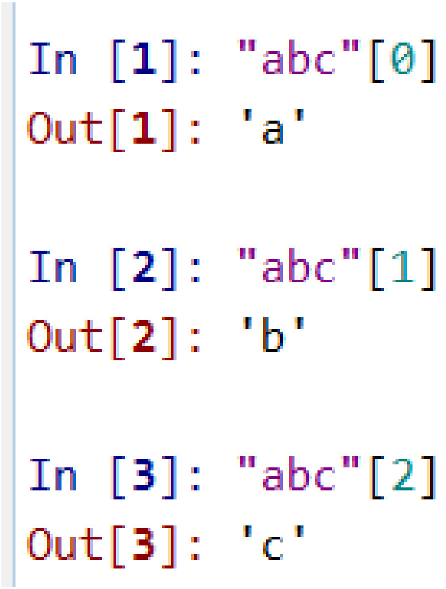
```
In [1]: "a string is a sequence of characters in Python"
Out[1]: 'a string is a sequence of characters in Python'
```

single quotes '   ' or double quotes "   "

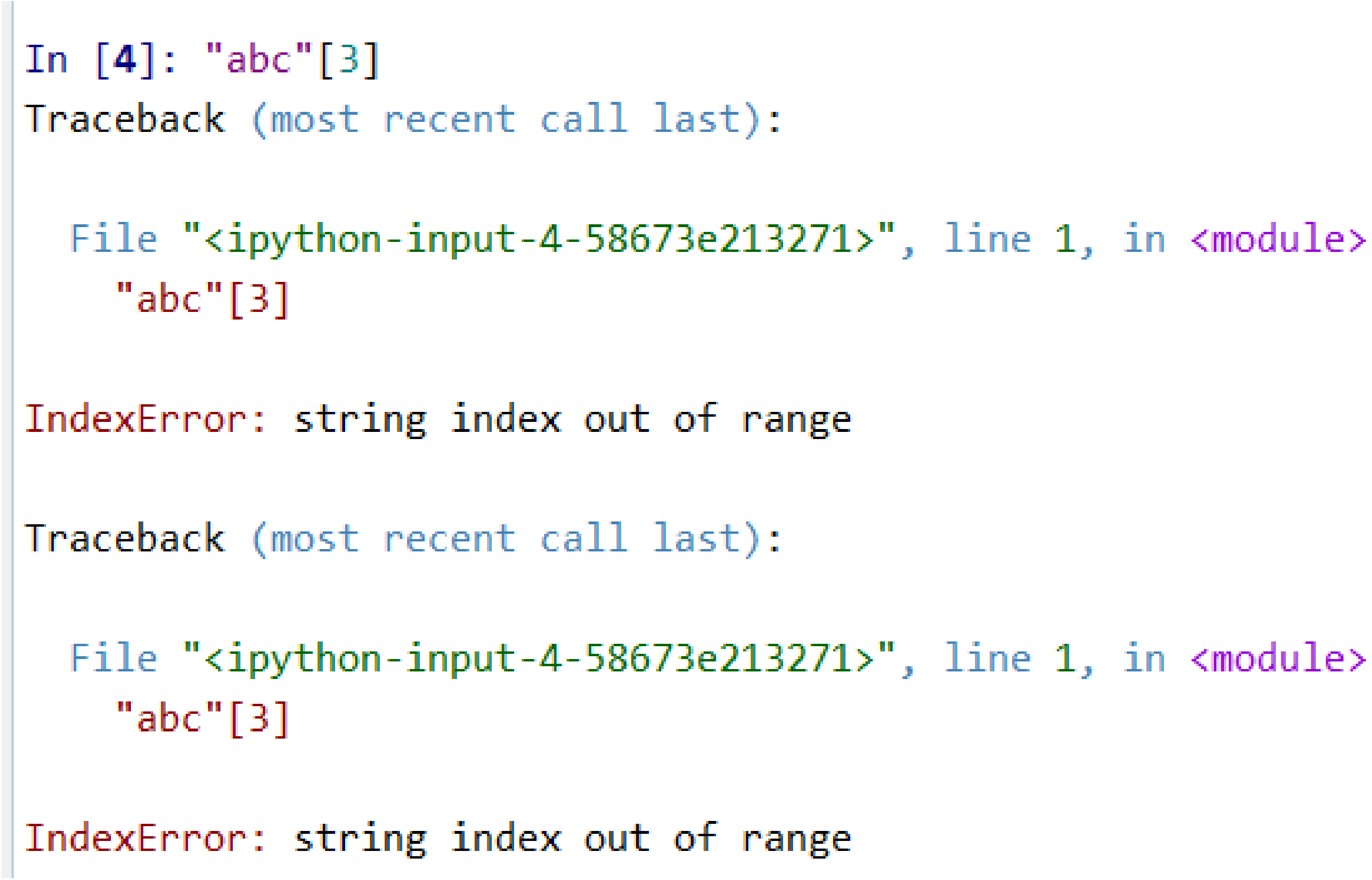
String in Python

- Obtain a character in a string using index (starting from 0)

a string: "abc"



```
In [1]: "abc"[0]
Out[1]: 'a'

In [2]: "abc"[1]
Out[2]: 'b'

In [3]: "abc"[2]
Out[3]: 'c'
```

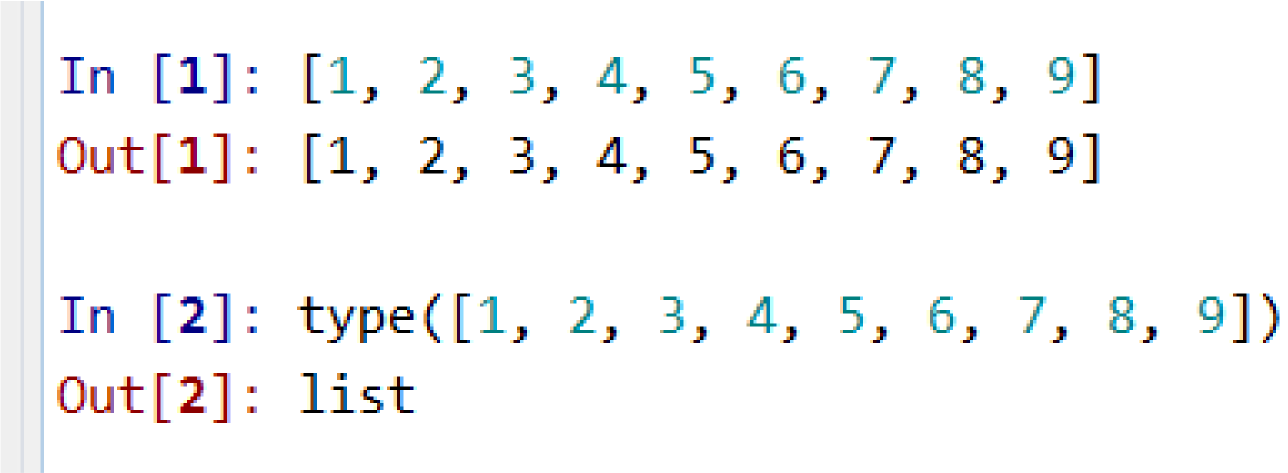


```
In [4]: "abc"[3]
Traceback (most recent call last):

  File "<ipython-input-4-58673e213271>", line 1, in <module>
    "abc"[3]

IndexError: string index out of range

Traceback (most recent call last):

  File "<ipython-input-4-58673e213271>", line 1, in <module>
    "abc"[3]

IndexError: string index out of range
```

List in Python

- a list is a sequence of objects (e.g. integers, or float numbers) in Python

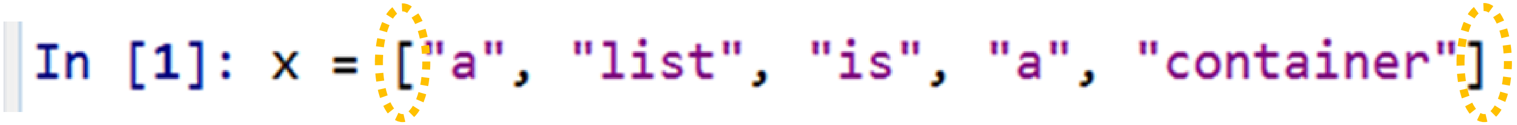
a list of integers



```
In [1]: [1, 2, 3, 4, 5, 6, 7, 8, 9]
Out[1]: [1, 2, 3, 4, 5, 6, 7, 8, 9]

In [2]: type([1, 2, 3, 4, 5, 6, 7, 8, 9])
Out[2]: list
```

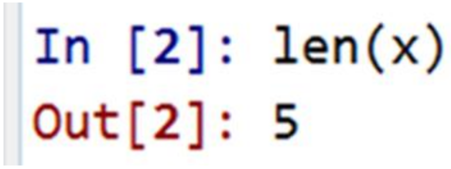
Define a list and Measure the length of a list

define a list using
square brackets



```
In [1]: x = ["a", "list", "is", "a", "container"]
```

use function **len** to measure
the length of a list:
the number of elements

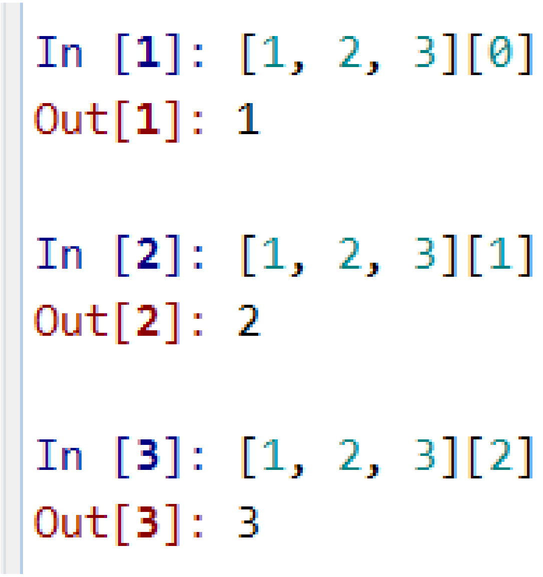


```
In [2]: len(x)
Out[2]: 5
```

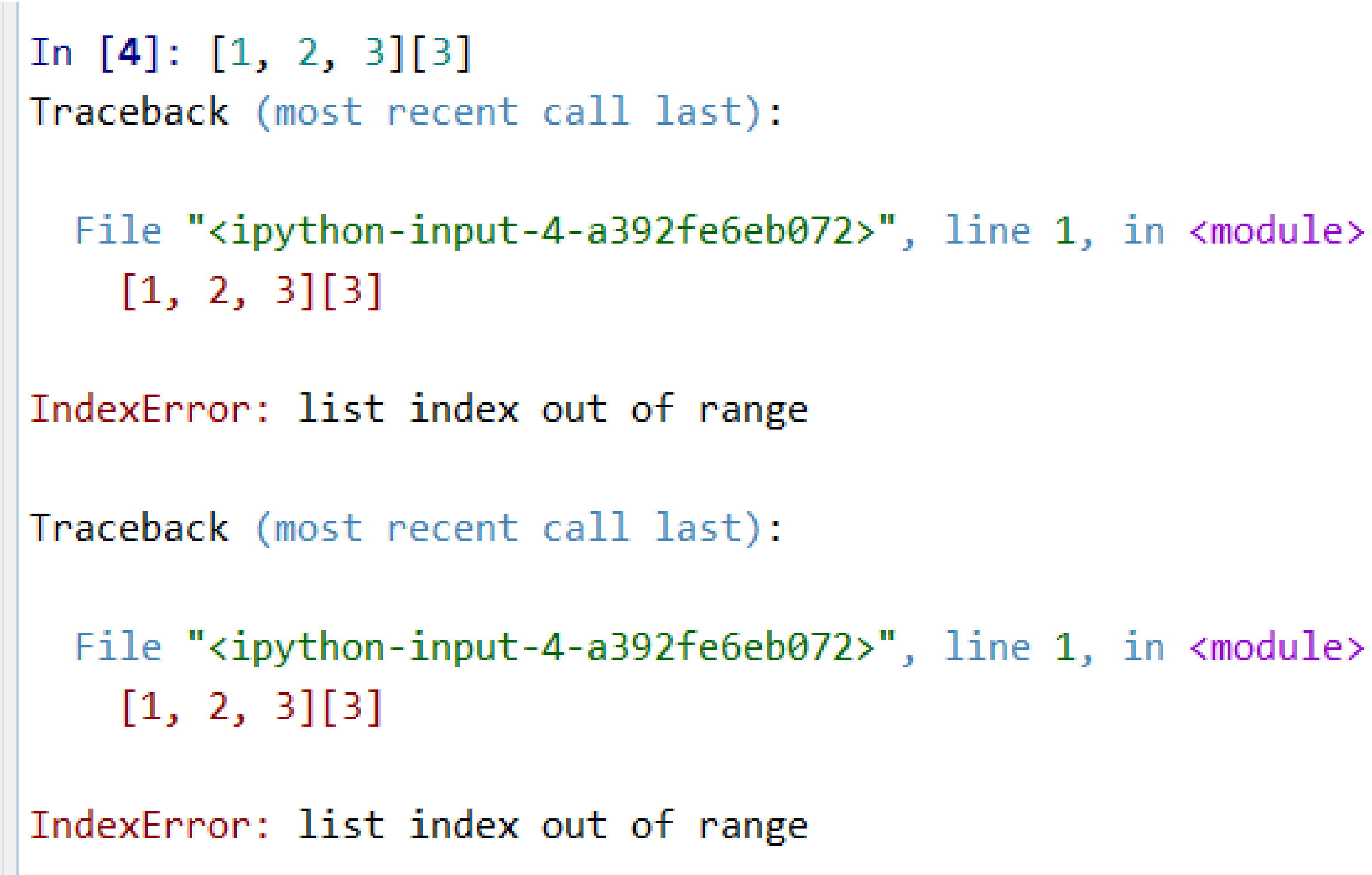
List in Python

- Obtain an element in a list using index (starting from 0)

a list: [1, 2, 3]



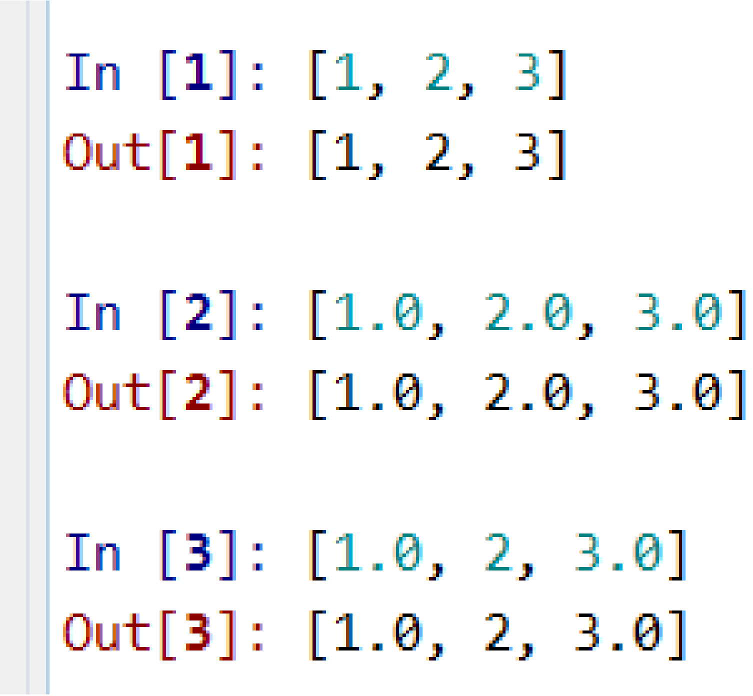
```
In [1]: [1, 2, 3][0]
Out[1]: 1

In [2]: [1, 2, 3][1]
Out[2]: 2

In [3]: [1, 2, 3][2]
Out[3]: 3
```



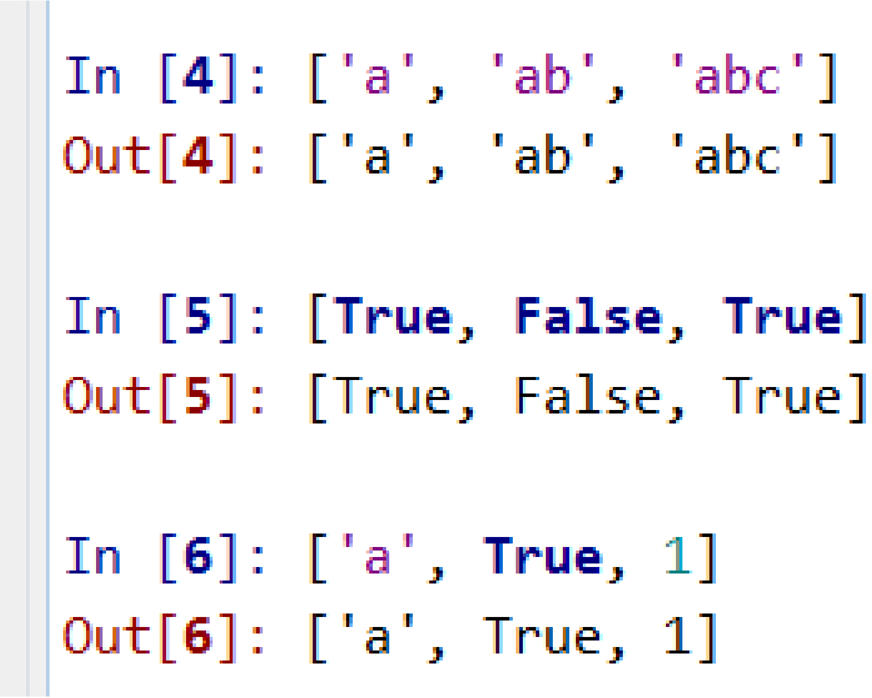
```
In [4]: [1, 2, 3][3]
Traceback (most recent call last):

  File "<ipython-input-4-a392fe6eb072>", line 1, in <module>
    [1, 2, 3][3]

IndexError: list index out of range

Traceback (most recent call last):

  File "<ipython-input-4-a392fe6eb072>", line 1, in <module>
    [1, 2, 3][3]

IndexError: list index out of range
```

List in Python

- a list is a sequence of objects.
- elements in a list can be any objects in Python



```
In [1]: [1, 2, 3]
Out[1]: [1, 2, 3]

In [2]: [1.0, 2.0, 3.0]
Out[2]: [1.0, 2.0, 3.0]

In [3]: [1.0, 2, 3.0]
Out[3]: [1.0, 2, 3.0]
```



```
In [4]: ['a', 'ab', 'abc']
Out[4]: ['a', 'ab', 'abc']

In [5]: [True, False, True]
Out[5]: [True, False, True]

In [6]: ['a', True, 1]
Out[6]: ['a', True, 1]
```

Summary on some Basic Data **Types**

- **int**

  python integer can represent any integer number

  as long as it can be stored in computer memory

- **float**

  float numbers can only represent a subset of real numbers.

  If the absolute value of a number is too large, it will be represented by +inf or -inf

- **bool (True/False)** : used in Boolean operations
- **string**

  a string is a sequence of characters

  each char of a string can be accessed by index

- **list**

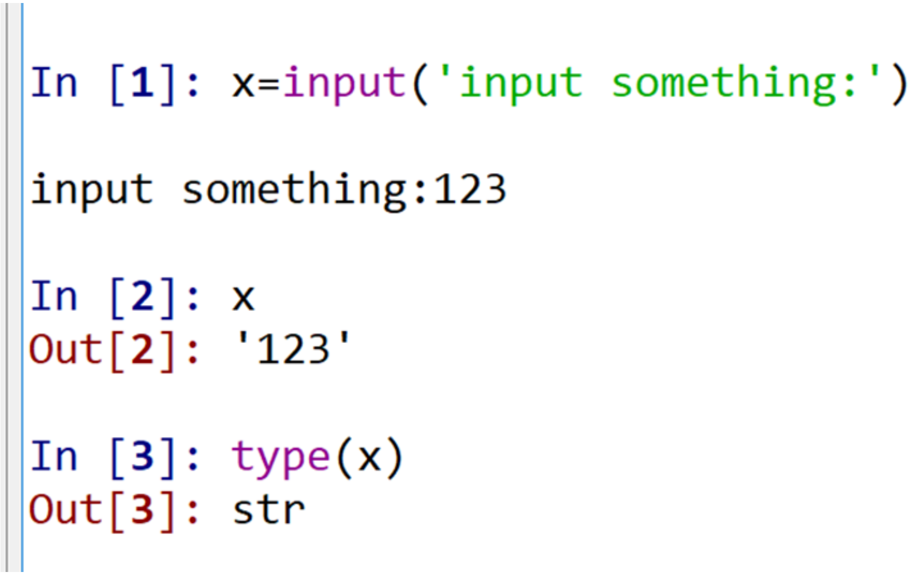
  a list is a sequence of objects.

  each element of a list can be accessed by index

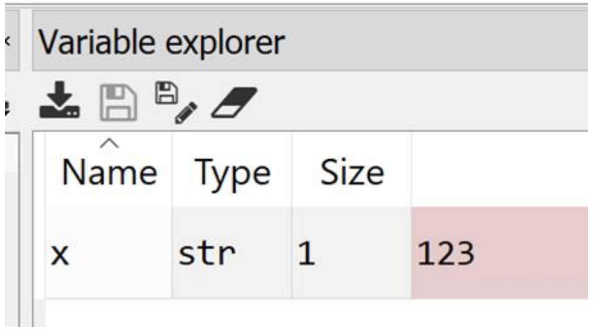
Use Function **input** to get user input from keyboard

- If the **input** function is called, the program flow will be stopped until the user has given an input and has ended the input with the return key. The text of the optional parameter, i.e. the prompt, will be printed on the screen.

- The input of the user will be returned as a string, regardless of the ‘data type’ of the input.



```
In [1]: x=input('input something:')

input something:123

In [2]: x
Out[2]: '123'

In [3]: type(x)
Out[3]: str
```



Variable explorer

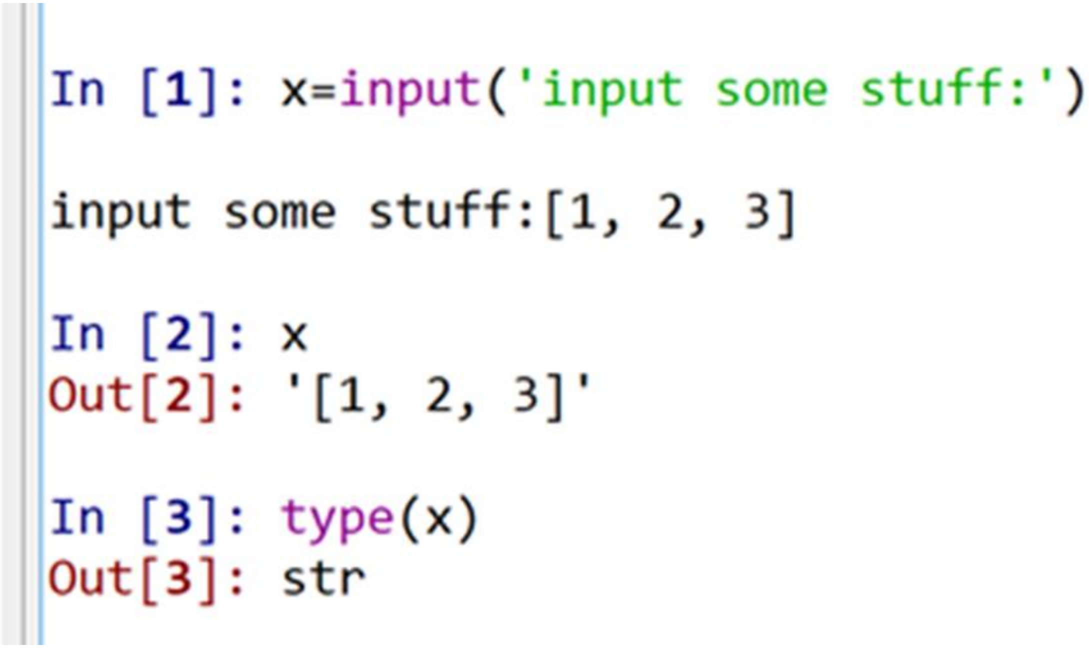
Name	Type	Size	
x	str	1	123

input something: is **prompt**
Type is str, not int

Use Function **input** to get user input from keyboard

- If the **input** function is called, the program flow will be stopped until the user has given an input and has ended the input with the return key. The text of the optional parameter, i.e. the prompt, will be printed on the screen.

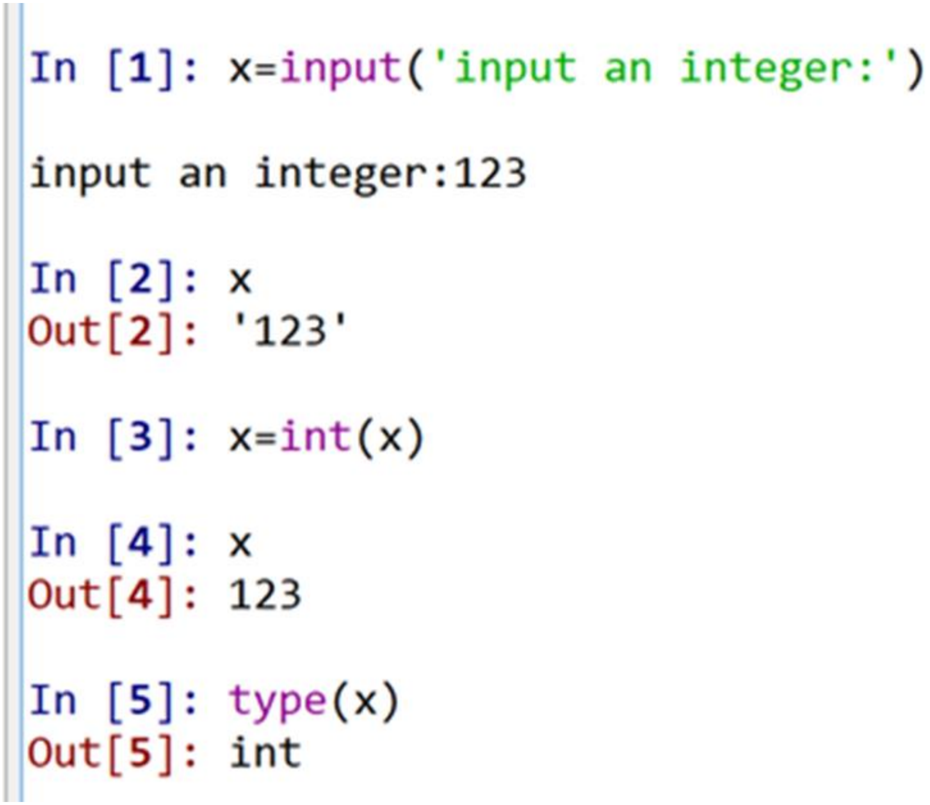
- **The input of the user will be returned as a string, regardless of the ‘data type’ of the input.**



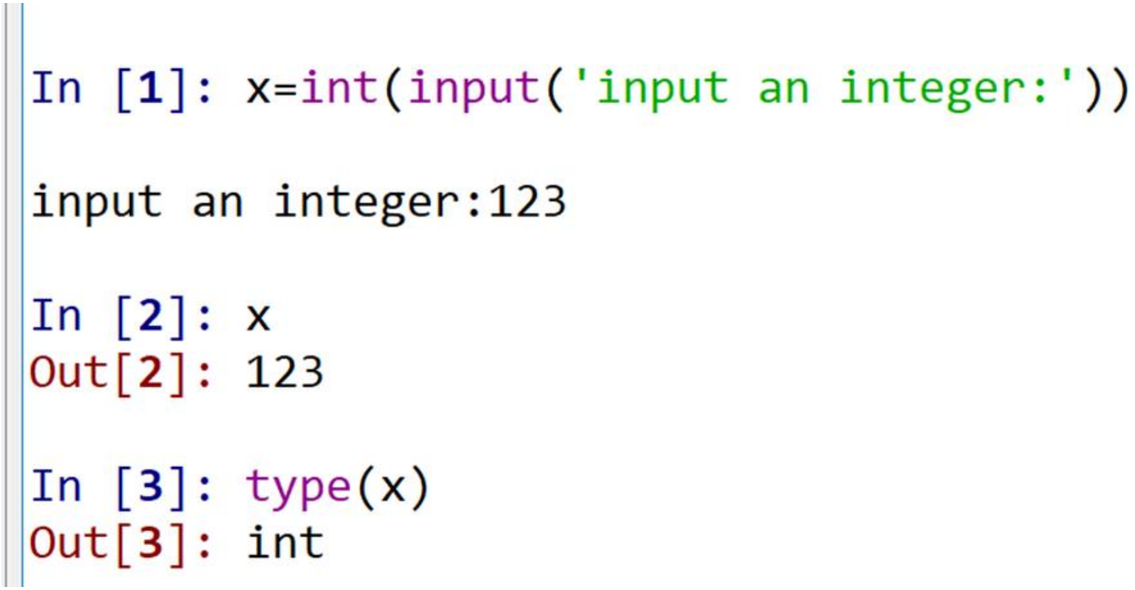
```
In [1]: x=input('input some stuff:')

input some stuff:[1, 2, 3]

In [2]: x
Out[2]: '[1, 2, 3]'

In [3]: type(x)
Out[3]: str
```

input some stuff: is **prompt**
Type is str, not list

Get an integer (int) from the user via the keyboard

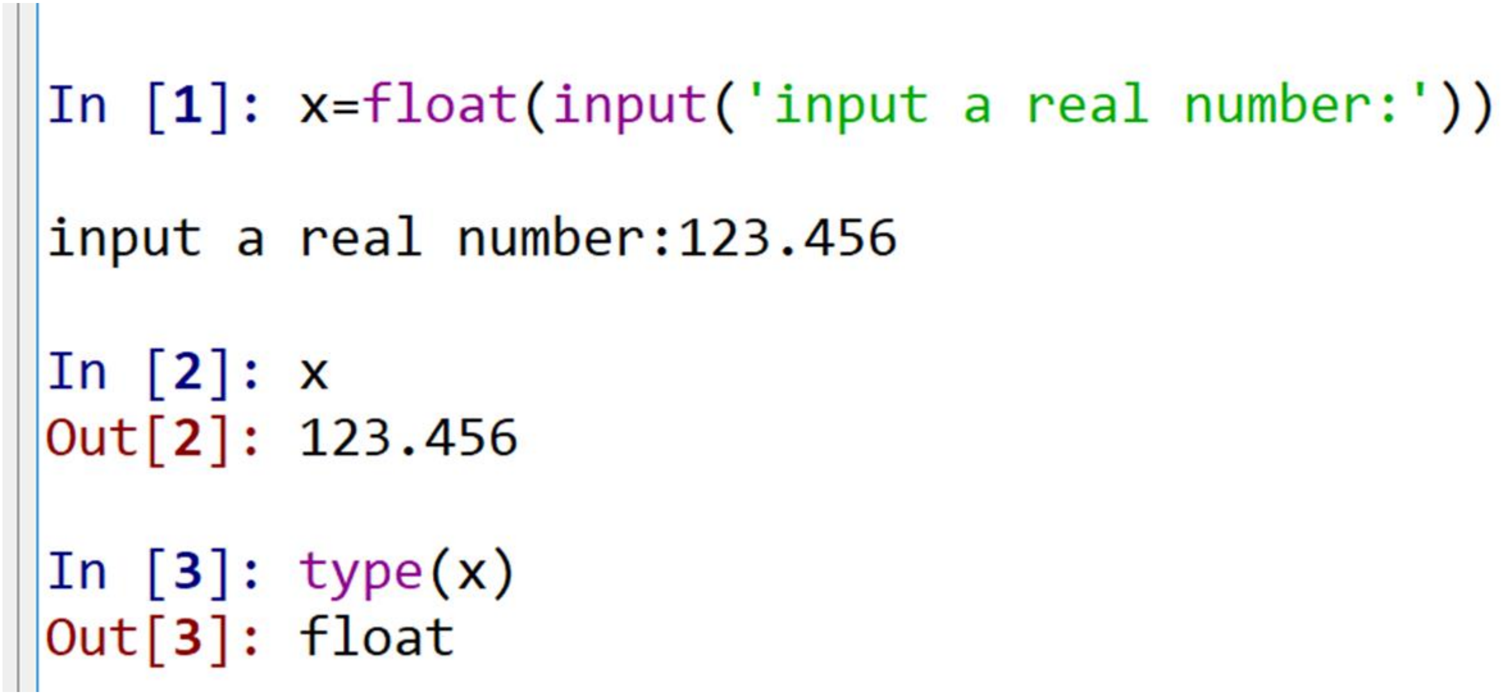


```
In [1]: x=input('input an integer:')

input an integer:123

In [2]: x
Out[2]: '123'

In [3]: x=int(x)

In [4]: x
Out[4]: 123

In [5]: type(x)
Out[5]: int
```



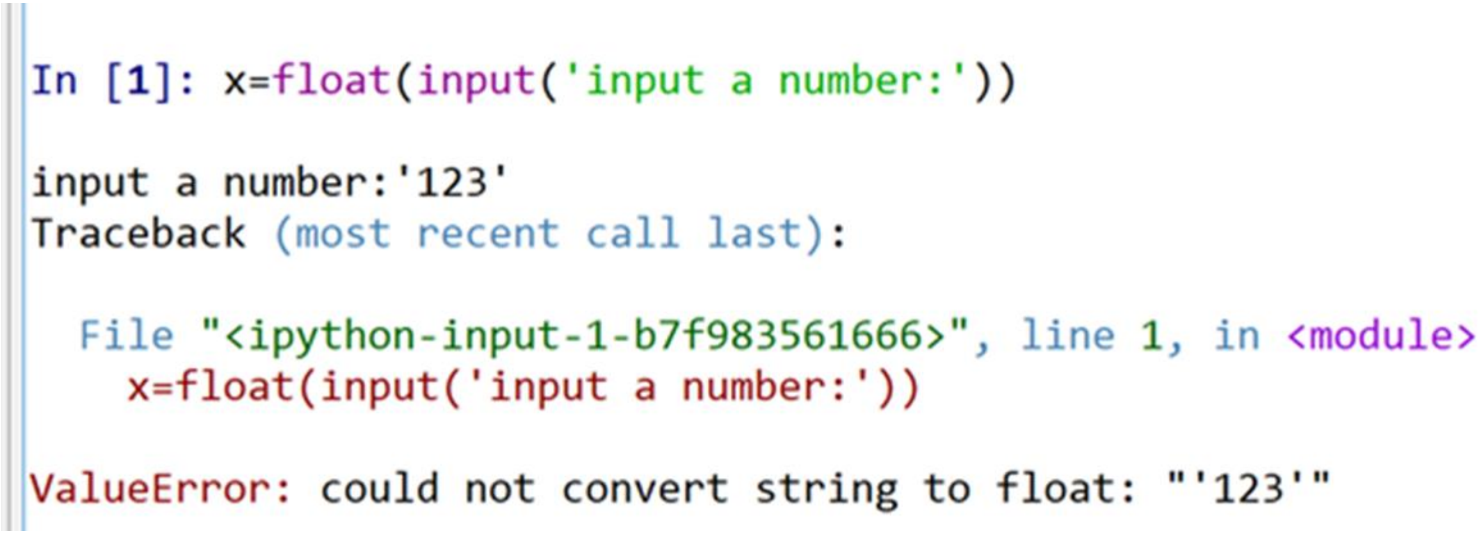
```
In [1]: x=int(input('input an integer:'))

input an integer:123

In [2]: x
Out[2]: 123

In [3]: type(x)
Out[3]: int
```

Get a real number (float) from the user via the keyboard

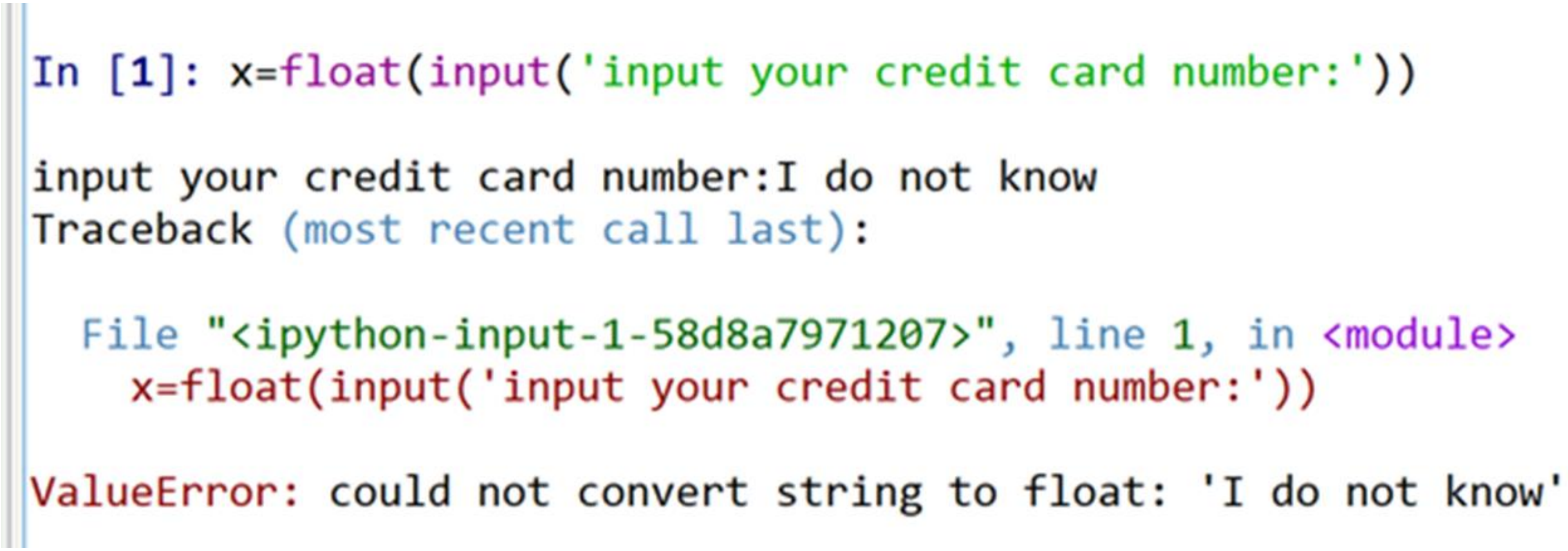


```
In [1]: x=float(input('input a real number:'))

input a real number:123.456

In [2]: x
Out[2]: 123.456

In [3]: type(x)
Out[3]: float
```

Oops...



```
In [1]: x=float(input('input a number:'))

input a number:'123'
Traceback (most recent call last):

  File "<ipython-input-1-b7f983561666>", line 1, in <module>
    x=float(input('input a number:'))

ValueError: could not convert string to float: "'123'"
```



```
In [1]: x=float(input('input your credit card number:'))

input your credit card number:I do not know
Traceback (most recent call last):

  File "<ipython-input-1-58d8a7971207>", line 1, in <module>
    x=float(input('input your credit card number:'))

ValueError: could not convert string to float: 'I do not know'
```