Module

Core Objects and Built-in Functions

Objective – Module 2

Core Objects and built in functions

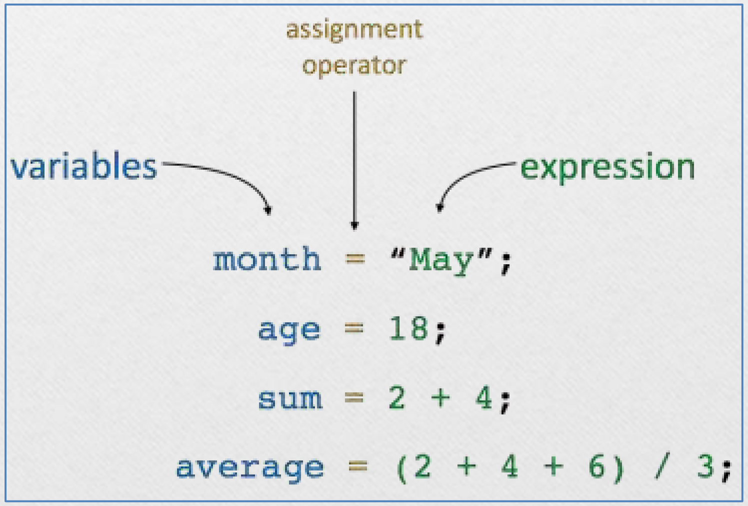
- Python Core Objects and builtin functions
- Number Object and operations
- String Object and Operations
- List Object and Operations
- Tuple Object and operations
- Dictionary Object and operations
- Set object and operations
- Boolean Object and None Object
- Different data Structures, data processing

Core Objects

- Numbers - ex: 1, 1.1, 3 + 4j and 200L
- Strings - ex: “Hi" "Tech"
- List - ex: [“ravi”, “zoya”]
- Tuple - ex: (‘Ethans’, “Tech”)
- Dictionary - - ex: (‘Institute’:GLA’)
- Set - ex: set([1,2,3,4])
- Files - ex: file = open(‘file.txt’)
- None - ex: None
- Boolean - ex: True, False

Assignment Operator

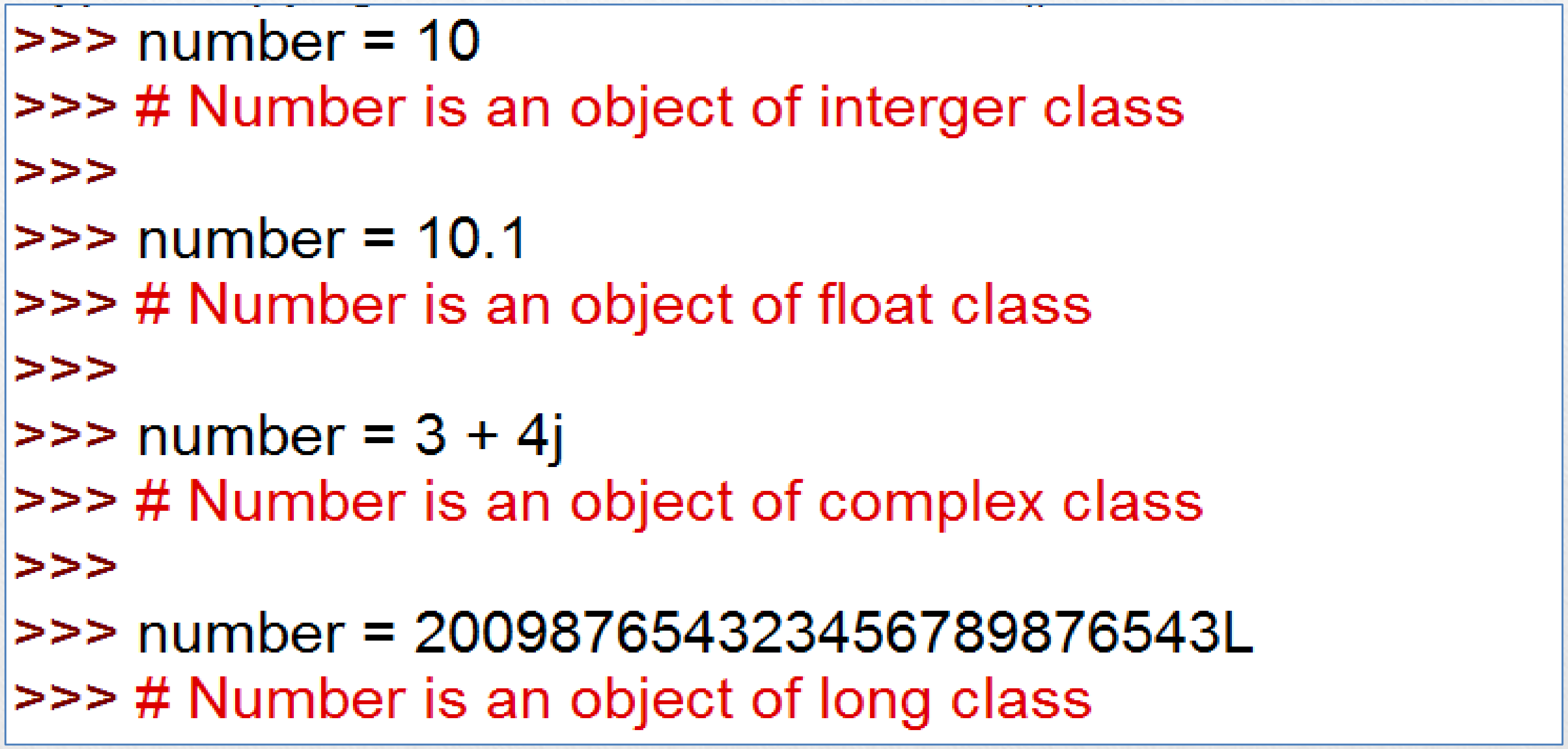
```
>>> number = 10
>>> name = 'Ethan'
>>> location = 'Pune'
>>> number1 = number2 = number3 = 1
>>> # Multiple assignment
>>> number1, number2 = 10, 20
```



```
    month = "May";
      age = 18;
      sum = 2 + 4;
  average = (2 + 4 + 6) / 3;
```

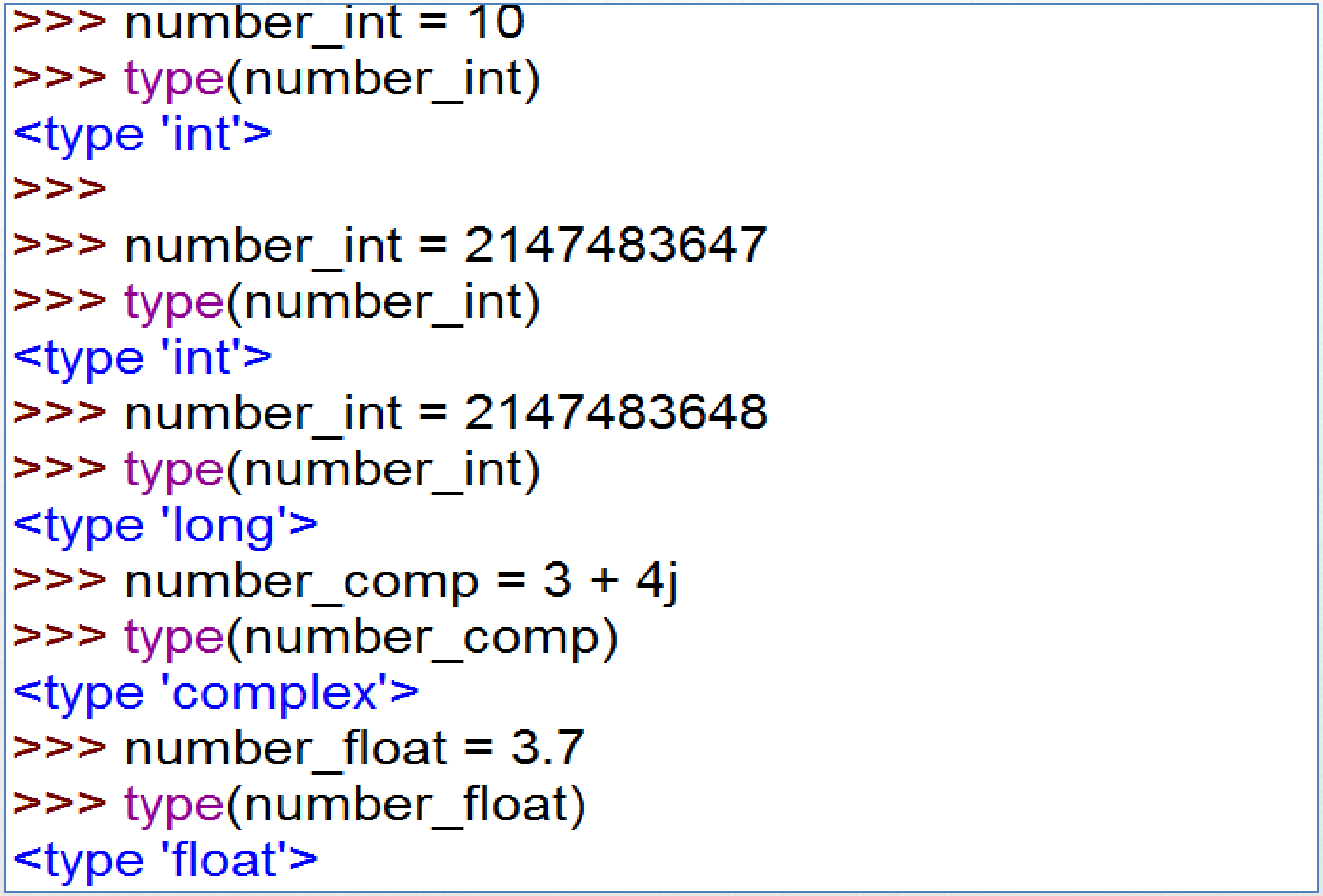
Numbers

- ✓Integers
- ✓Float
- ✓Long
- ✓Complex



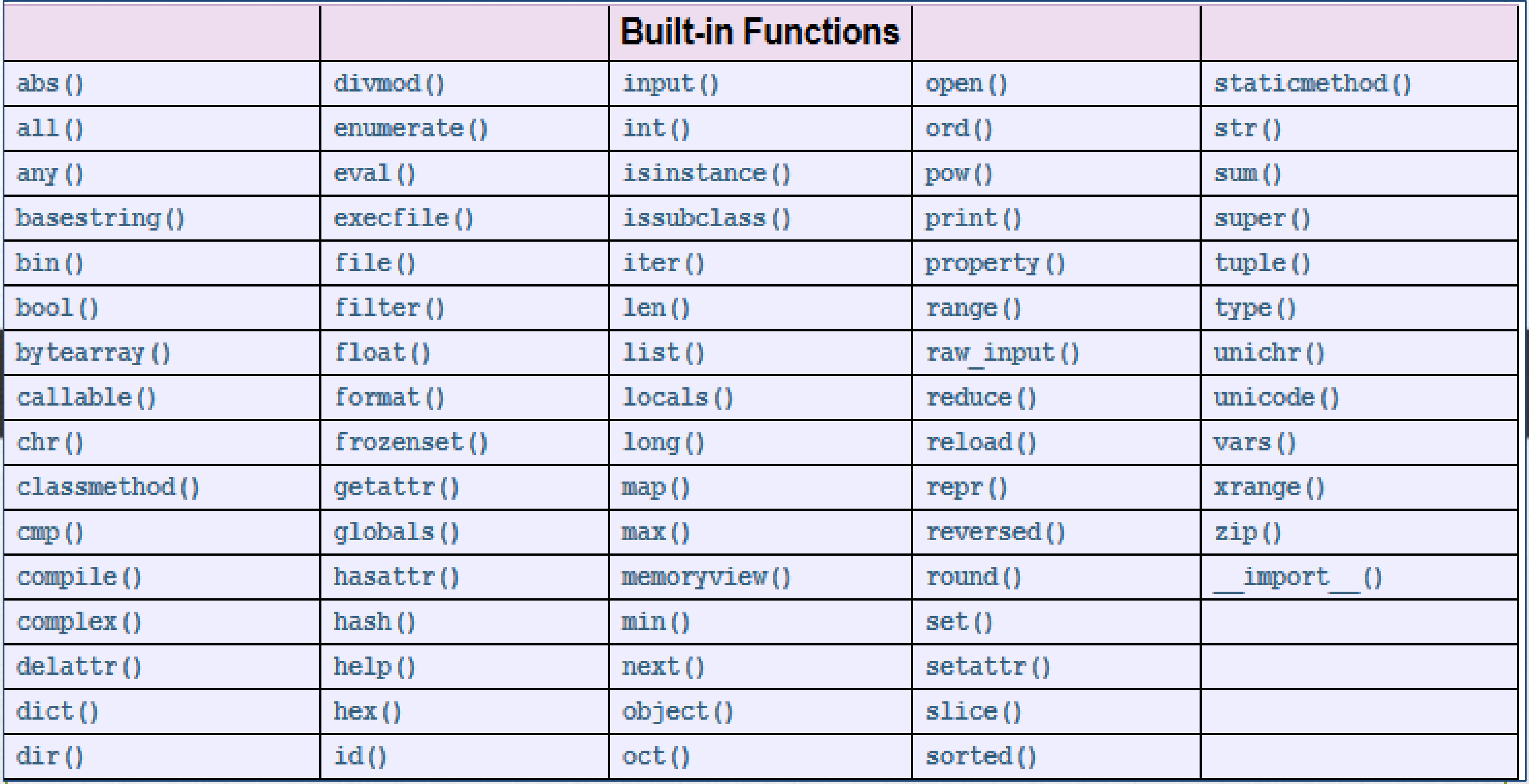
```
>>> number = 10
>>> # Number is an object of interger class
>>>
>>> number = 10.1
>>> # Number is an object of float class
>>>
>>> number = 3 + 4j
>>> # Number is an object of complex class
>>>
>>> number = 20098765432345678987654 3L
>>> # Number is an object of long class
```

type function



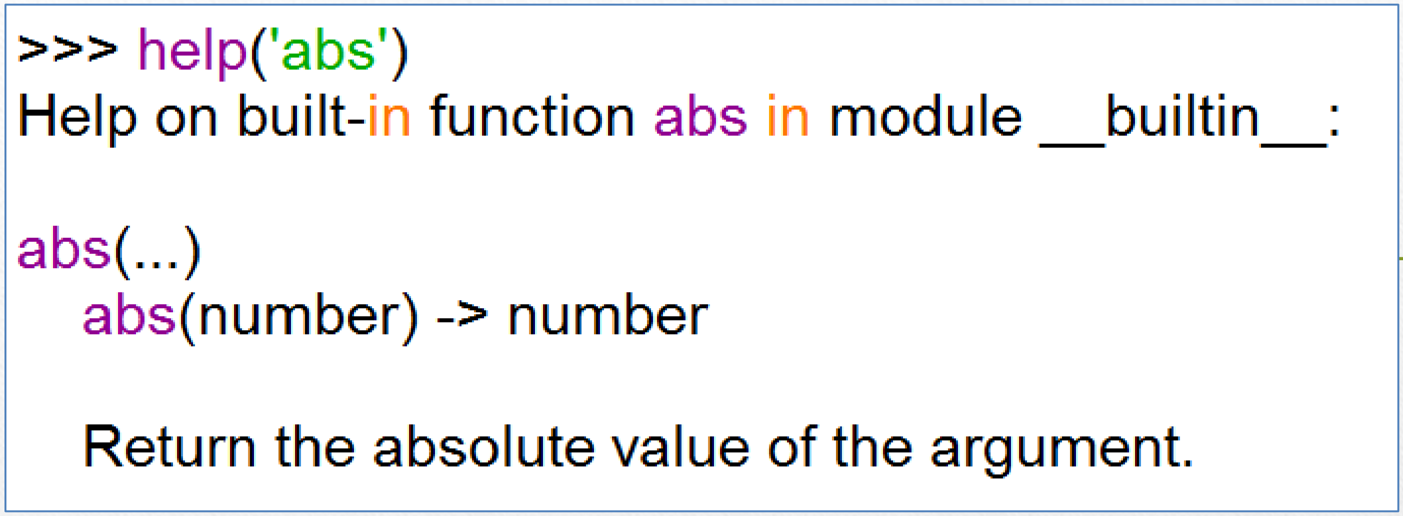
```
>>> number_int = 10
>>> type(number_int)
<type 'int'>
>>>
>>> number_int = 2147483647
>>> type(number_int)
<type 'int'>
>>> number_int = 2147483648
>>> type(number_int)
<type 'long'>
>>> number_comp = 3 + 4j
>>> type(number_comp)
<type 'complex'>
>>> number_float = 3.7
>>> type(number_float)
<type 'float'>
```

Builtin functions(in updated version function may be changed)

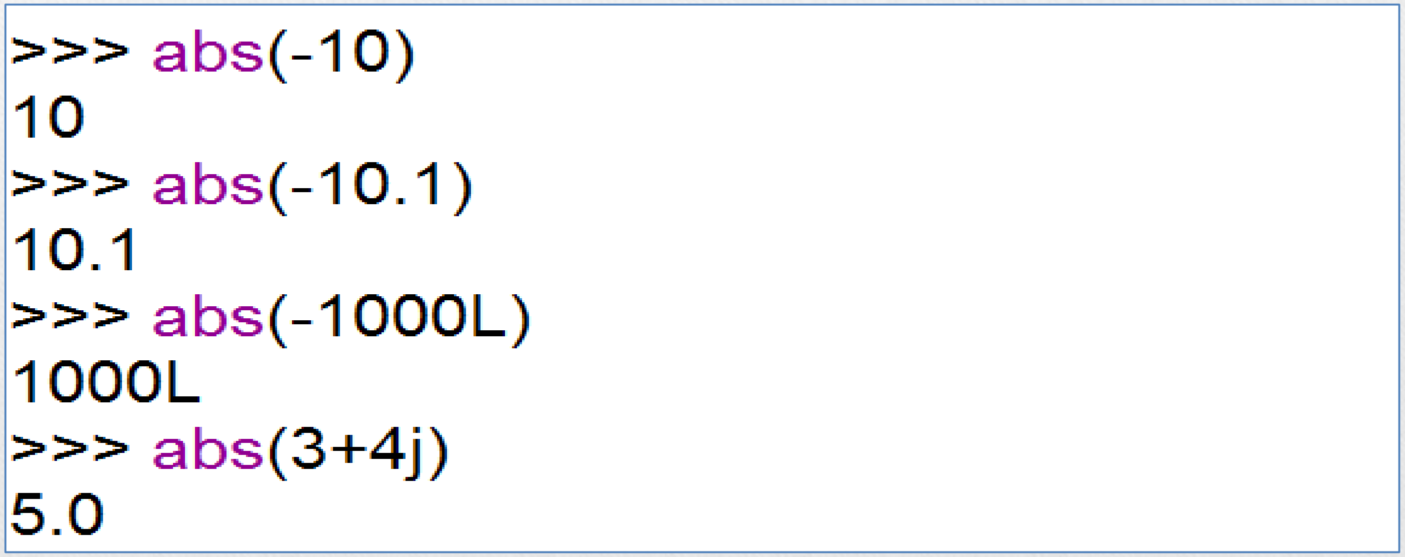


		Built-in Functions		
abs()	divmod()	input()	open()	staticmethod()
all()	enumerate()	int()	ord()	str()
any()	eval()	isinstance()	pow()	sum()
basestring()	execfile()	issubclass()	print()	super()
bin()	file()	iter()	property()	tuple()
bool()	filter()	len()	range()	type()
bytearray()	float()	list()	raw_input()	unichr()
callable()	format()	locals()	reduce()	unicode()
chr()	frozenset()	long()	reload()	vars()
classmethod()	getattr()	map()	repr()	xrange()
cmp()	globals()	max()	reversed()	zip()
compile()	hasattr()	memoryview()	round()	__import__()
complex()	hash()	min()	set()	
delattr()	help()	next()	setattr()	
dict()	hex()	object()	slice()	
dir()	id()	oct()	sorted()	

help



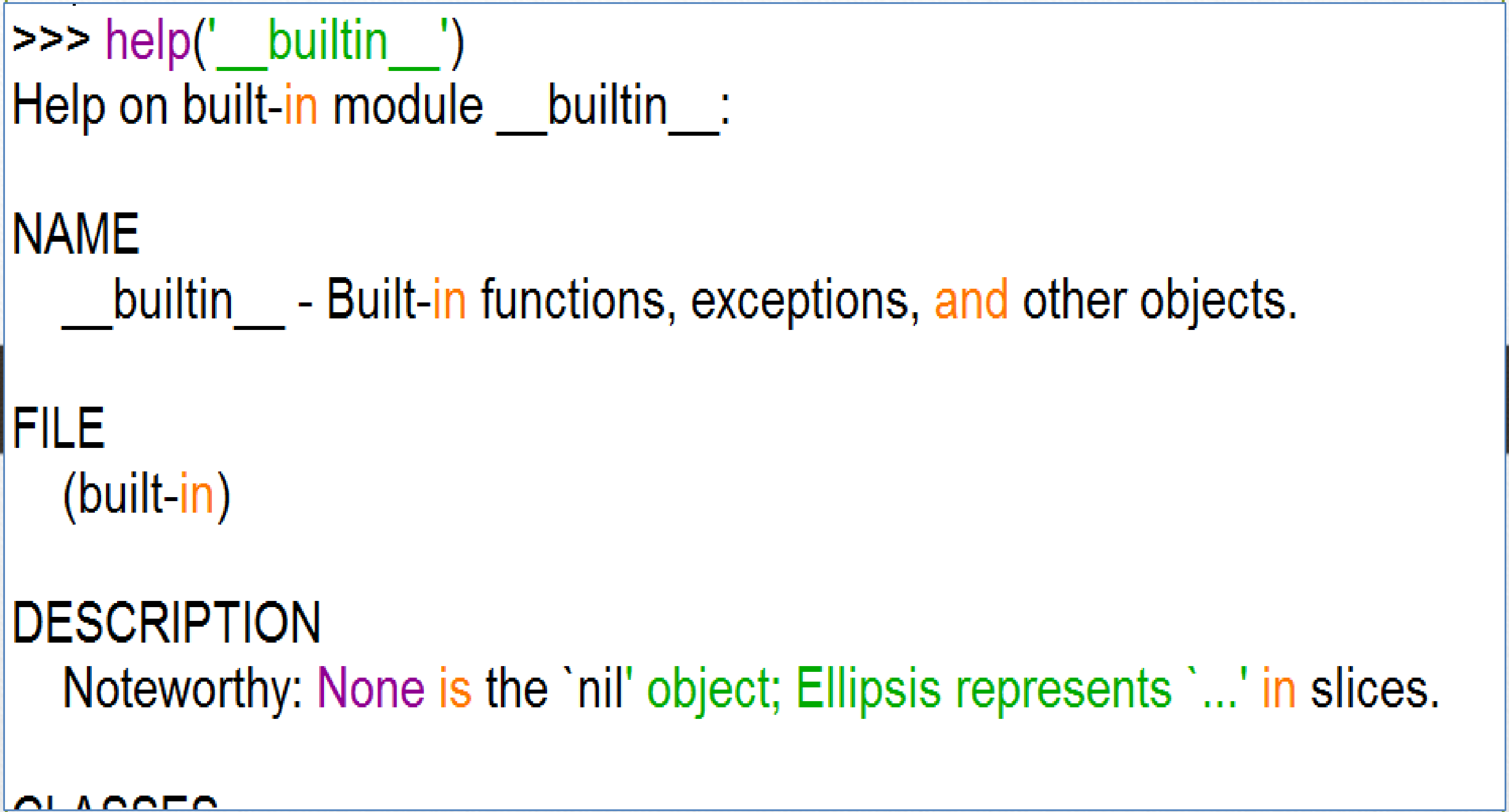
```
>>> help('abs')
Help on built-in function abs in module __builtin__:

abs(...)
    abs(number) -> number

    Return the absolute value of the argument.
```



```
>>> abs(-10)
10
>>> abs(-10.1)
10.1
>>> abs(-1000L)
1000L
>>> abs(3+4j)
5.0
```

www.ethans.co.in

Builtin help



```
>>> help('__builtin__')
Help on built-in module __builtin__:

NAME
    __builtin__ - Built-in functions, exceptions, and other objects.

FILE
    (built-in)

DESCRIPTION
    Noteworthy: None is the `nil' object; Ellipsis represents `...' in slices.
```

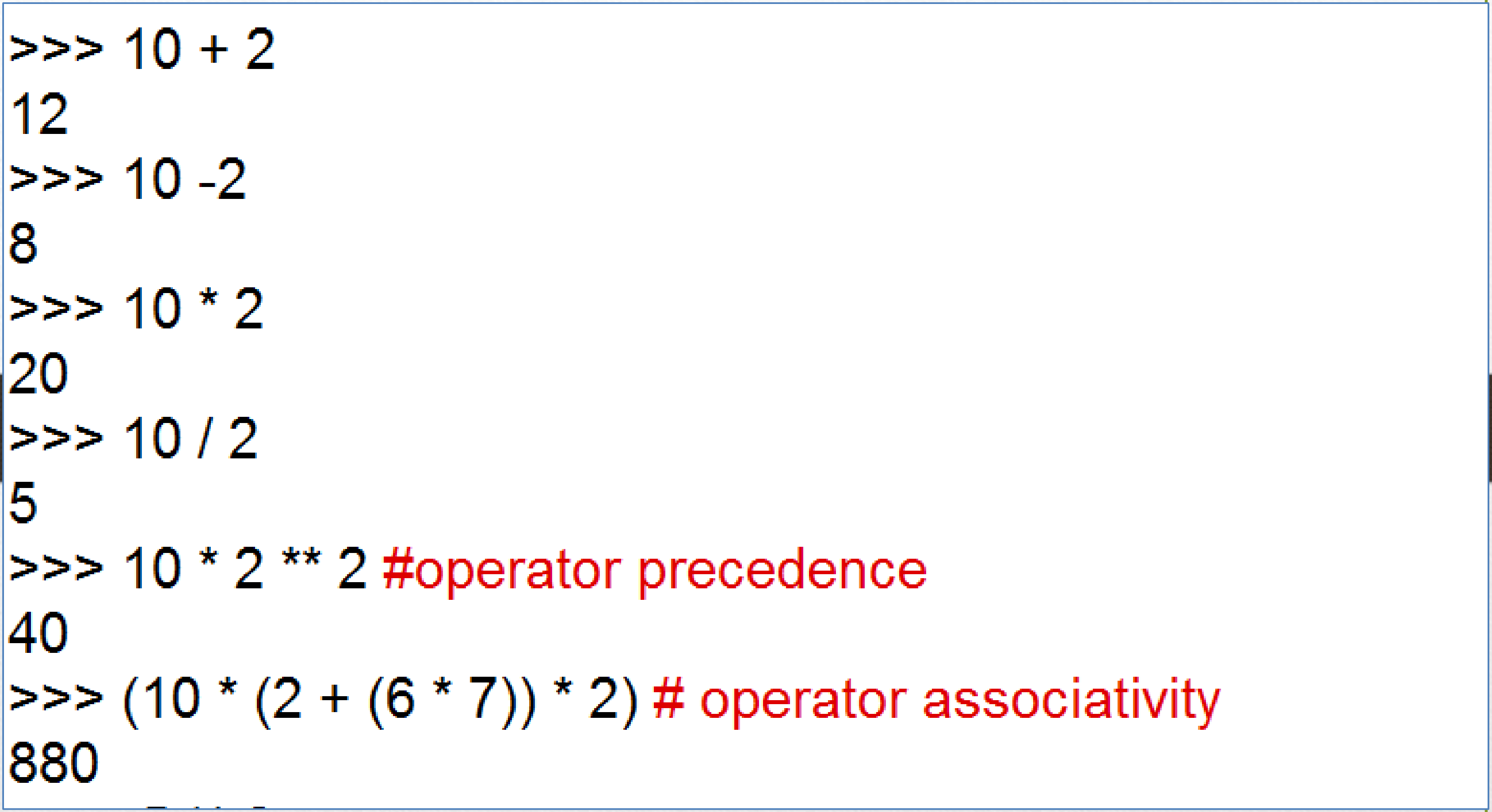
Number functions

```
>>> int('1010', 2)
10
>>> float(10)
10.0
>>> long(12)
12L
>>> complex(1, 2)
(1+2j)
```

```
>>> isinstance(1, int)
True
>>> isinstance(1.1, float)
True
>>> isinstance(1L, float)
False
>>> isinstance(complex(1,2), complex)
True
```

```
>>> cmp(1,2)
-1
>>> cmp(3,2)
1
>>> cmp(2,2)
0
```

Operation on numbers

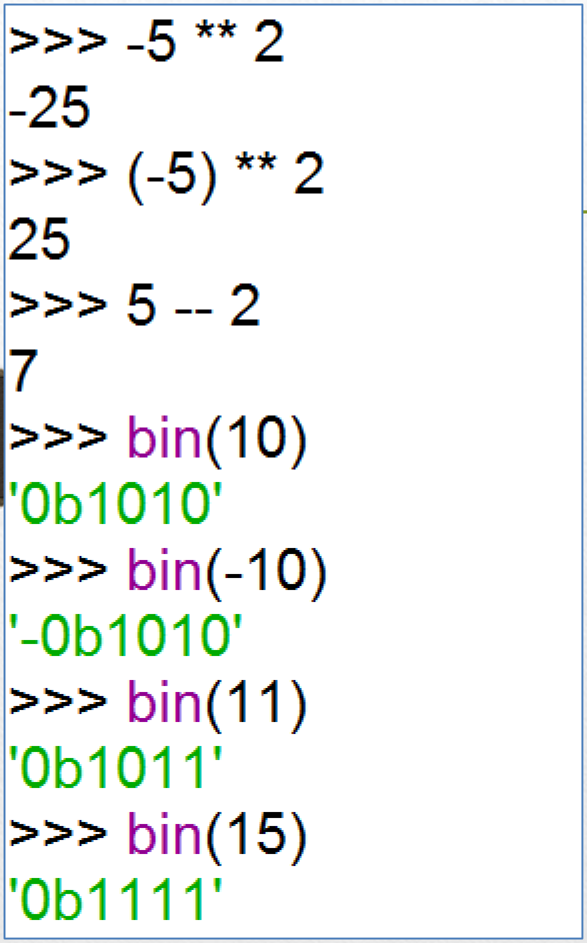


```
>>> 10 + 2
12
>>> 10 -2
8
>>> 10 * 2
20
>>> 10 / 2
5
>>> 10 * 2 ** 2 #operator precedence
40
>>> (10 * (2 + (6 * 7)) * 2) # operator associativity
880
```

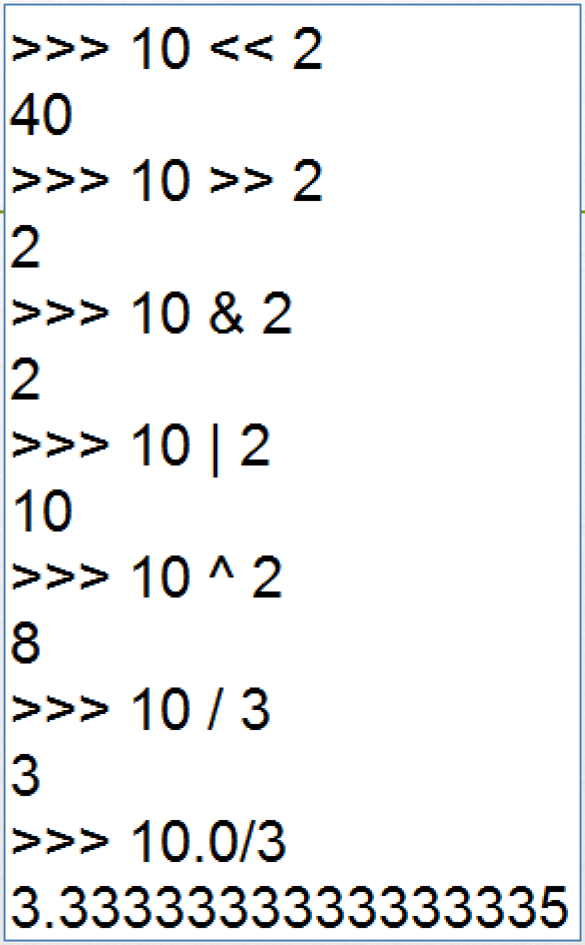
Precedence Table

Operator	Description
**	Exponentiation (raise to the power)
~ + -	Complement, unary plus and minus (method names for the last two are +@ and -@)
* / % //	Multiply, divide, modulo and floor division
+ -	Addition and subtraction
>> <<	Right and left bitwise shift
&	Bitwise 'AND'
^ |	Bitwise exclusive `OR' and regular `OR'
<= < > >=	Comparison operators
<> == !=	Equality operators
= %= /= //= -= += *= **=	Assignment operators
is, is not	Identity operators
in, not in	Membership operators
and, or, not	Logical operators

Operation on numbers

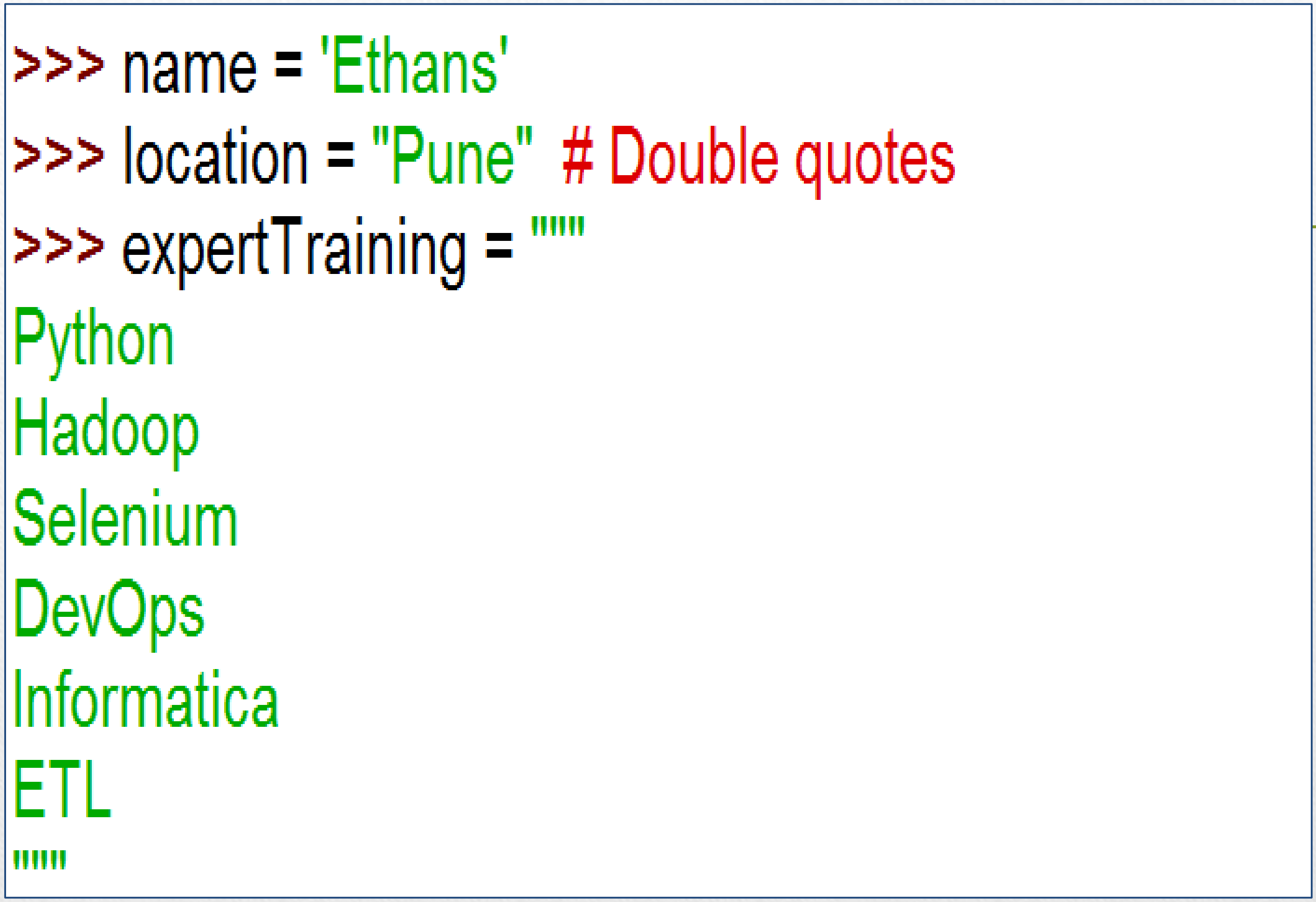


```
>>> -5 ** 2
-25
>>> (-5) ** 2
25
>>> 5 -- 2
7
>>> bin(10)
'0b1010'
>>> bin(-10)
'-0b1010'
>>> bin(11)
'0b1011'
>>> bin(15)
'0b1111'
```



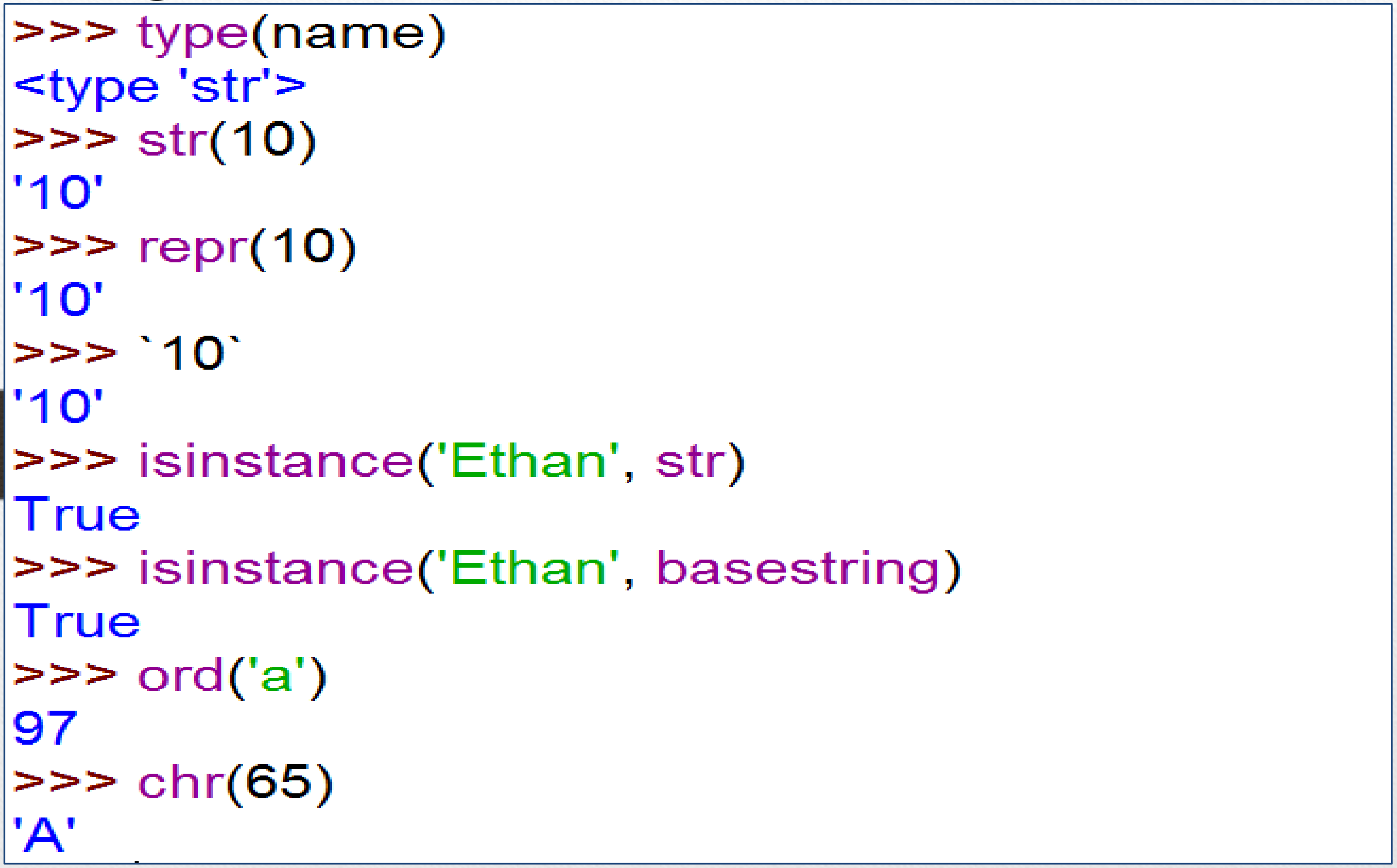
```
>>> 10 << 2
40
>>> 10 >> 2
2
>>> 10 & 2
2
>>> 10 | 2
10
>>> 10 ^ 2
8
>>> 10 / 3
3
>>> 10.0/3
3.3333333333333335
```

String

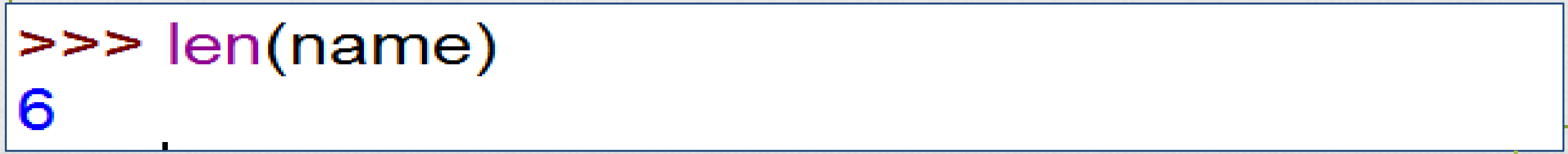


```
>>> name = 'Ethans'
>>> location = "Pune"  # Double quotes
>>> expertTraining = """
Python
Hadoop
Selenium
DevOps
Informatica
ETL
"""
```

String Functions



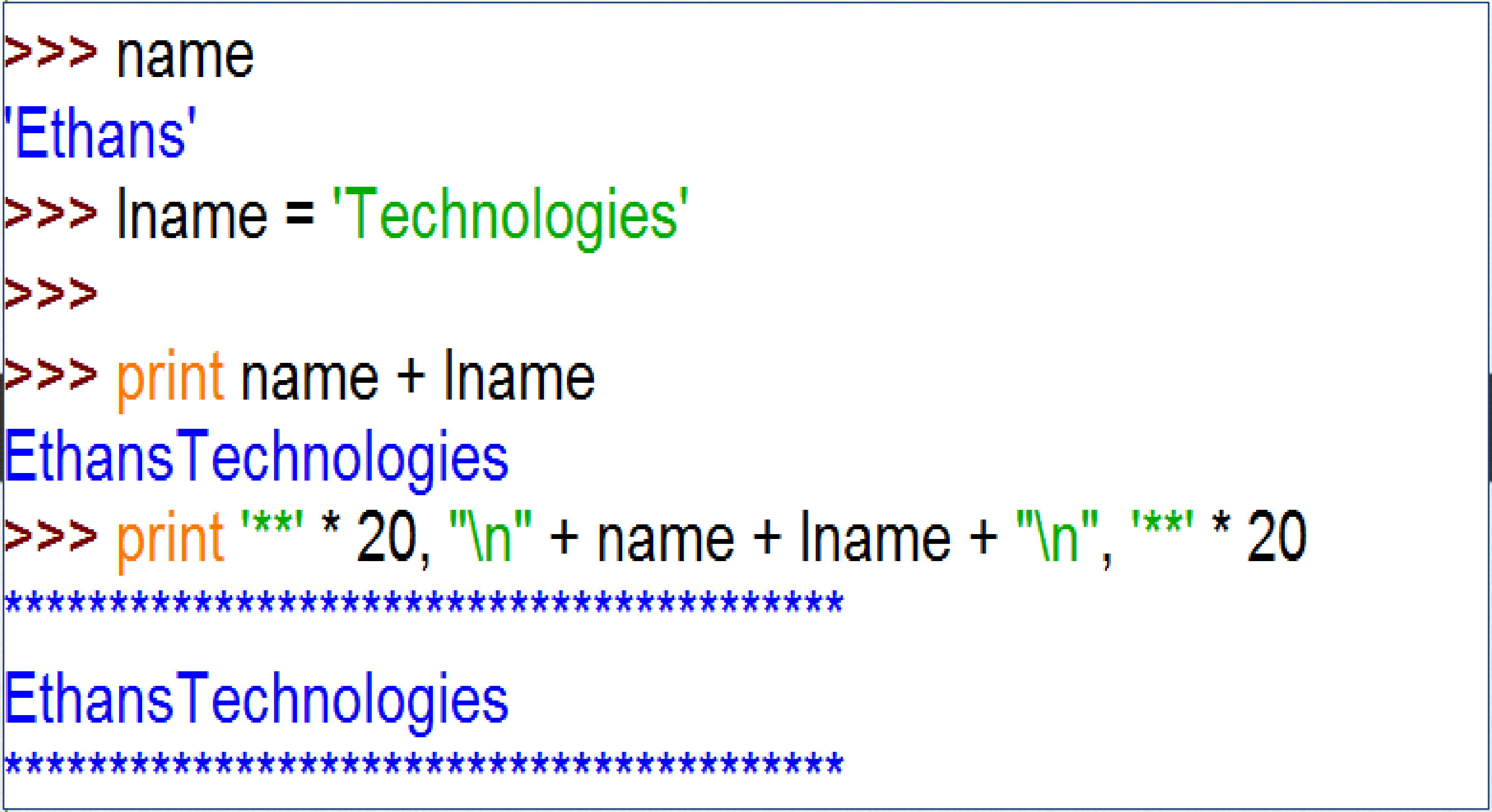
```
>>> type(name)
<type 'str'>
>>> str(10)
'10'
>>> repr(10)
'10'
>>> `10`
'10'
>>> isinstance('Ethan', str)
True
>>> isinstance('Ethan', basestring)
True
>>> ord('a')
97
>>> chr(65)
'A'
```



```
>>> len(name)
6
```

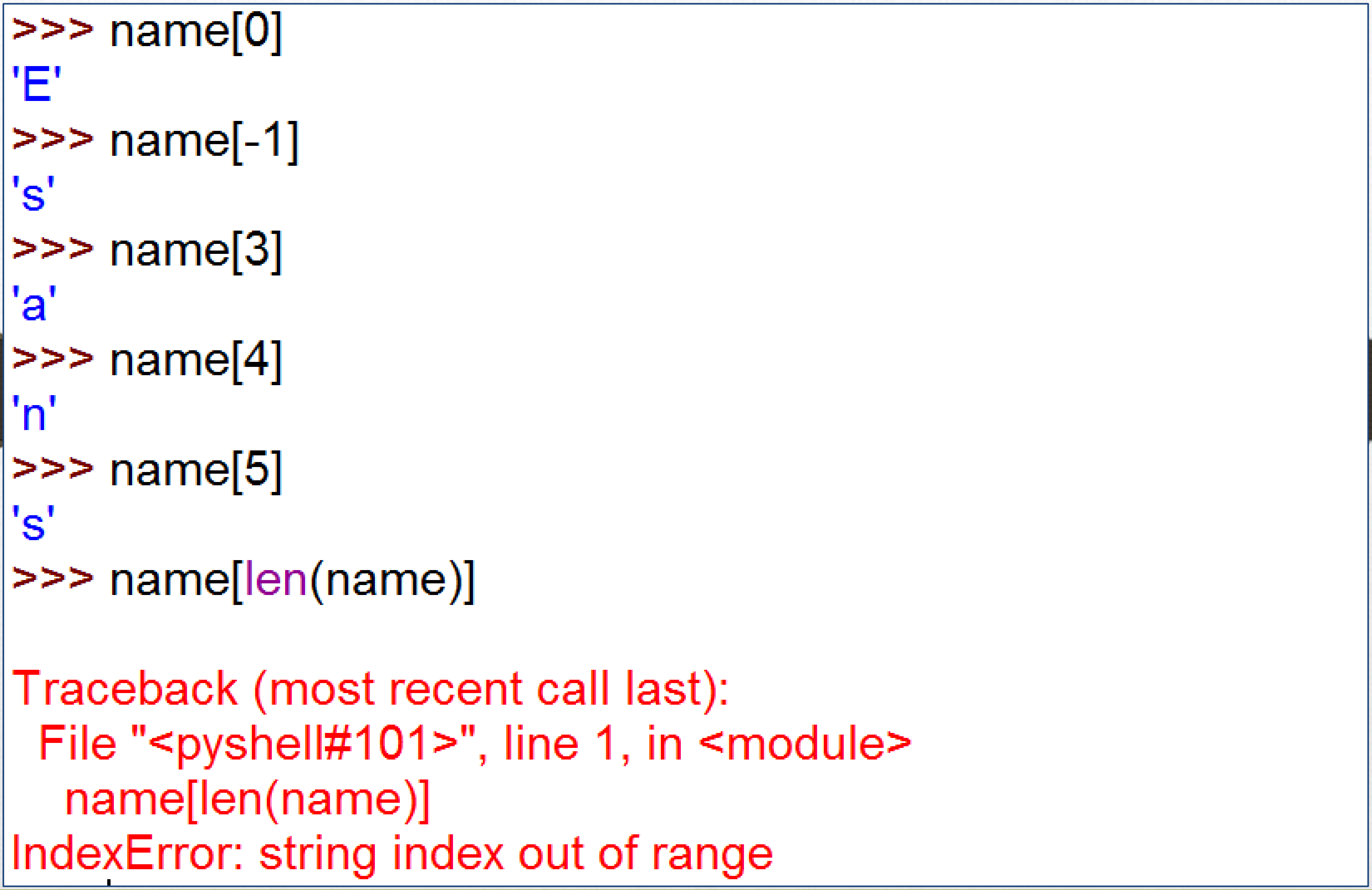
www.ethans.co.in

String Operations



```
>>> name
'Ethans'
>>> lname = 'Technologies'
>>>
>>> print name + lname
EthansTechnologies
>>> print '**' * 20, "\n" + name + lname + "\n", '**' * 20
****************************************
EthansTechnologies
****************************************
```

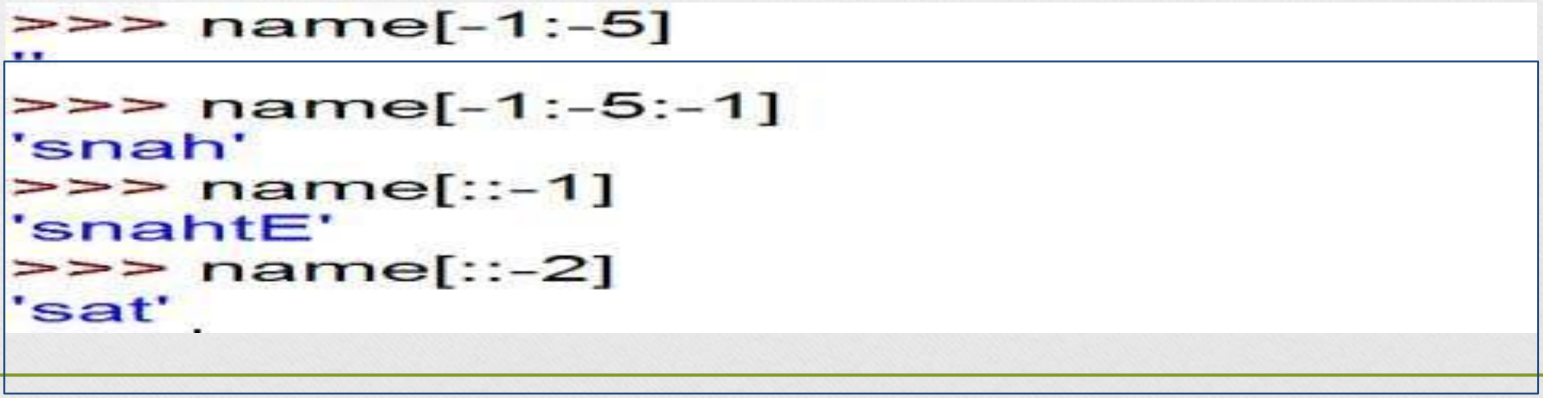
String Indexing



```
>>> name[0]
'E'
>>> name[-1]
's'
>>> name[3]
'a'
>>> name[4]
'n'
>>> name[5]
's'
>>> name[len(name)]

Traceback (most recent call last):
  File "<pyshell#101>", line 1, in <module>
    name[len(name)]
IndexError: string index out of range
```

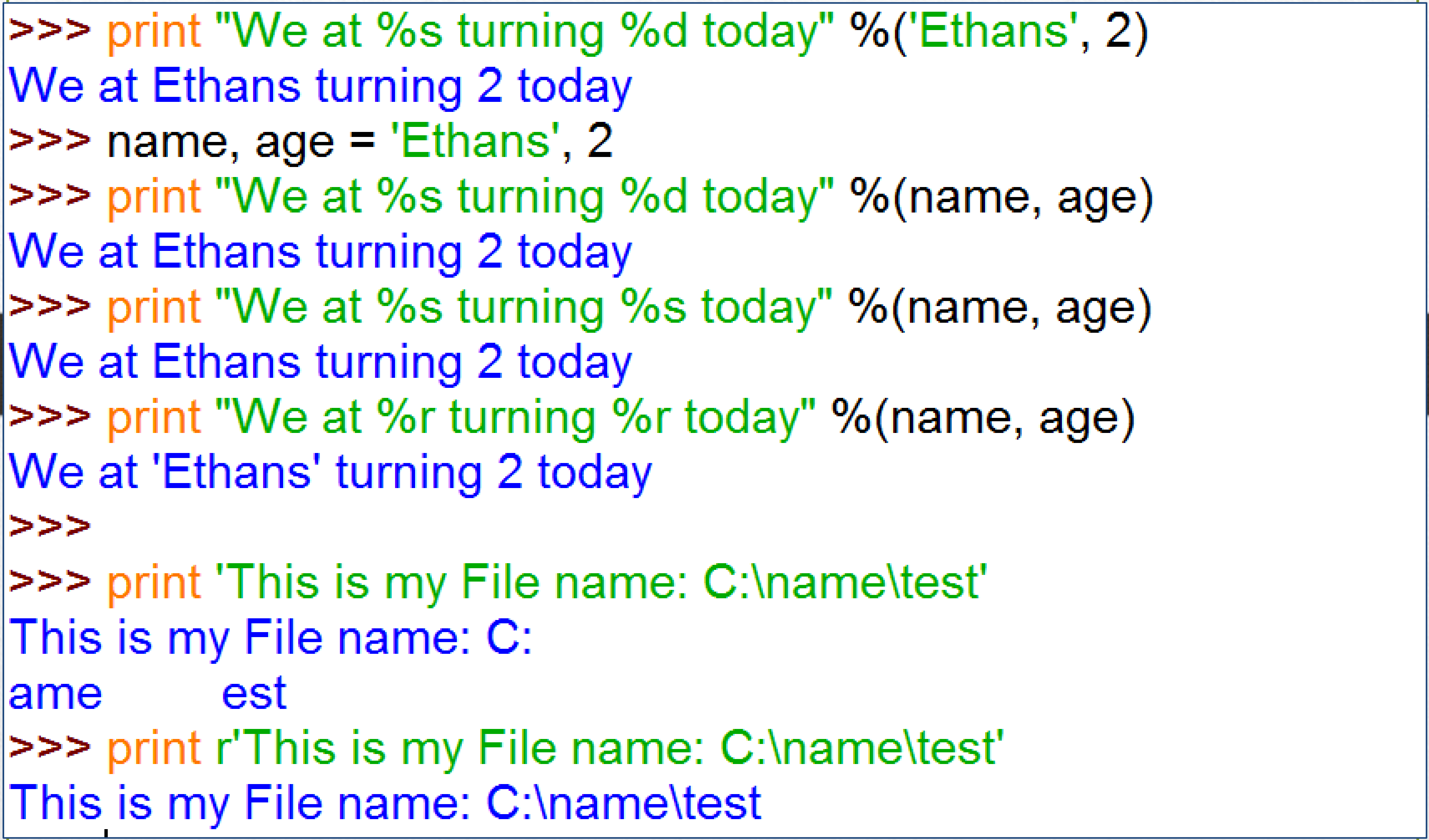
String Slicing

```
>>> # Slicing Syntax
>>> # string[START:STOP:STEP]
>>>
>>> name[0:5:1]
'Ethan'
>>> name[0:]
'Ethans'
>>> name[:]
'Ethans'
>>> name[:9]
'Ethans'
>>> name[::2]
'Ehn'
```



```
>>> name[-1:-5]
''
>>> name[-1:-5:-1]
'snah'
>>> name[::-1]
'snahtE'
>>> name[::-2]
'sat'
```

String Formatting

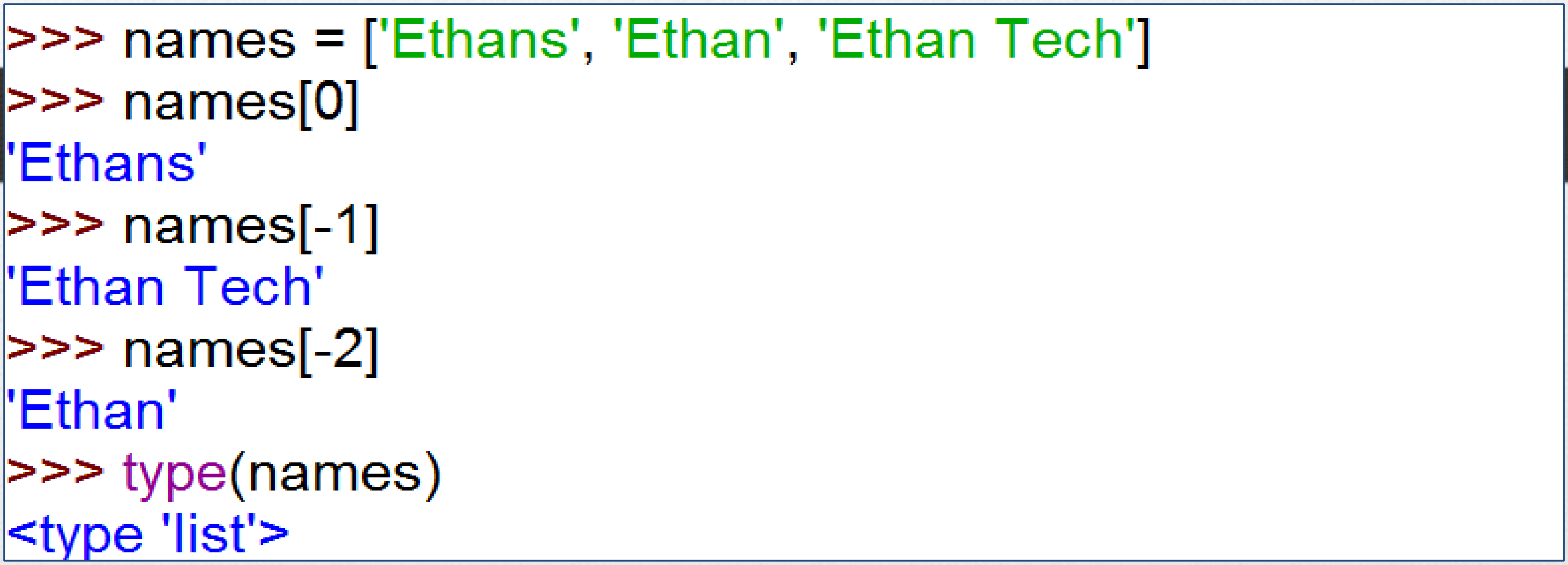


```
>>> print "We at %s turning %d today" %('Ethans', 2)
We at Ethans turning 2 today
>>> name, age = 'Ethans', 2
>>> print "We at %s turning %d today" %(name, age)
We at Ethans turning 2 today
>>> print "We at %s turning %s today" %(name, age)
We at Ethans turning 2 today
>>> print "We at %r turning %r today" %(name, age)
We at 'Ethans' turning 2 today
>>>
>>> print 'This is my File name: C:\name\test'
This is my File name: C:
ame        est
>>> print r'This is my File name: C:\name\test'
This is my File name: C:\name\test
```

List

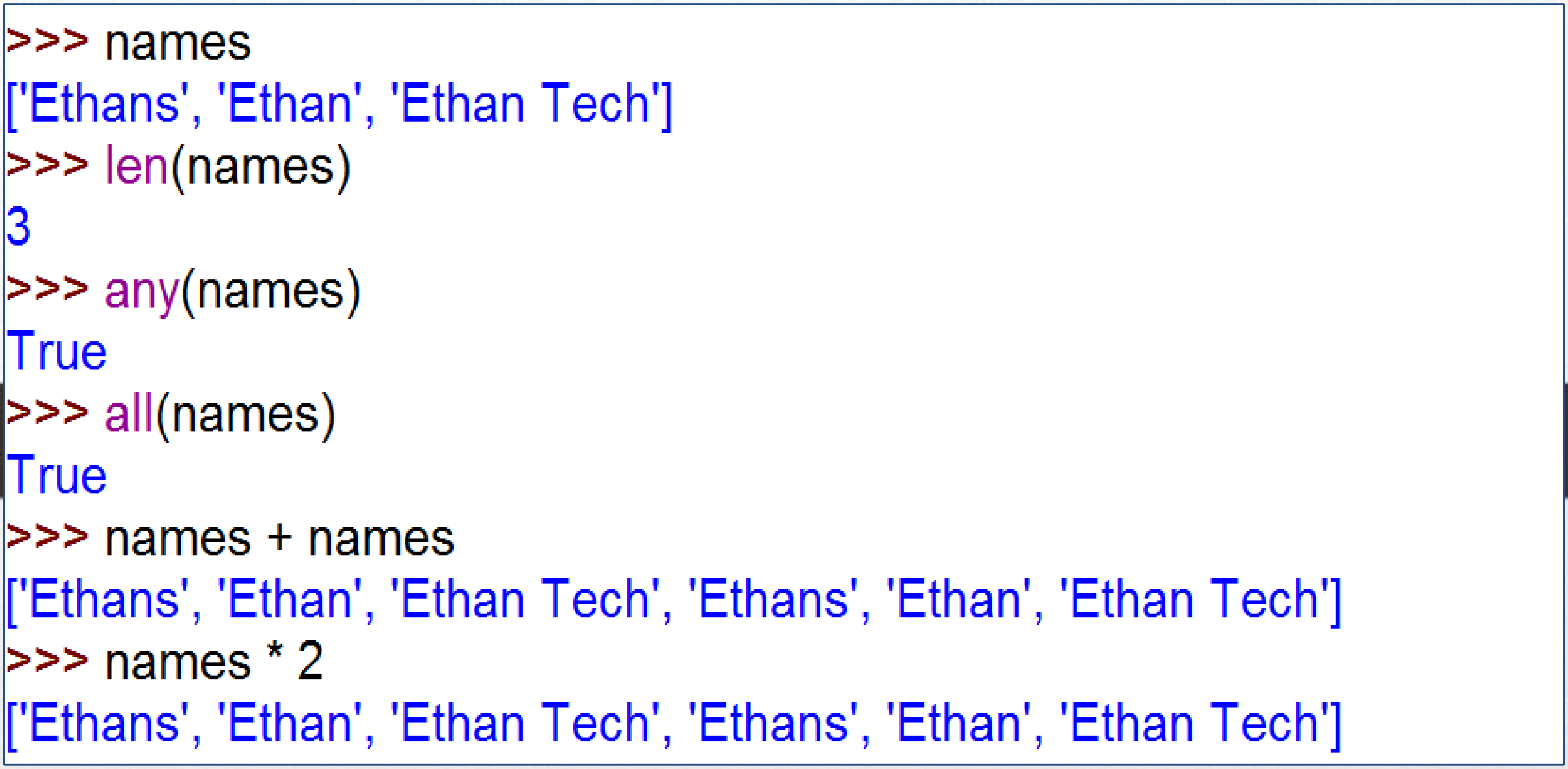
The list is another datatype in Python which can be written with comma-separated values (items) in a square brackets.

Items in a list should be of same type or different type.



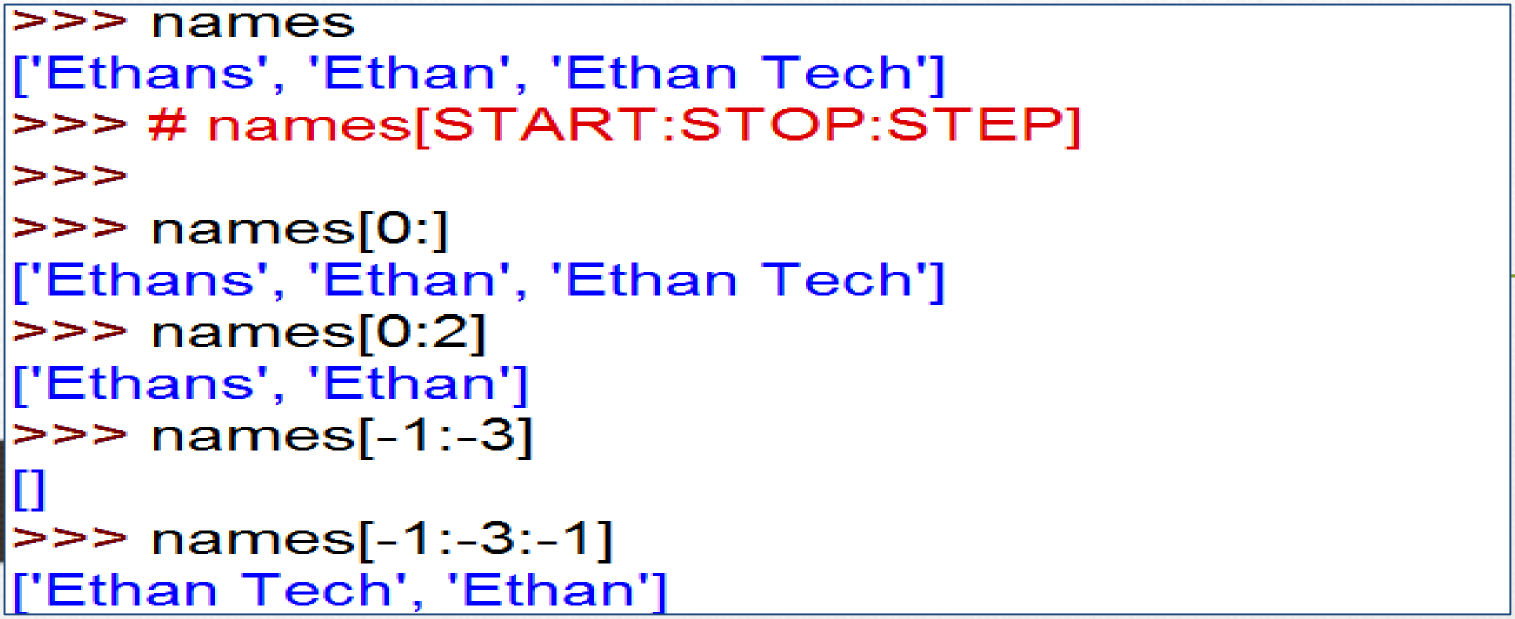
```
>>> names = ['Ethans', 'Ethan', 'Ethan Tech']
>>> names[0]
'Ethans'
>>> names[-1]
'Ethan Tech'
>>> names[-2]
'Ethan'
>>> type(names)
<type 'list'>
```

List Functions

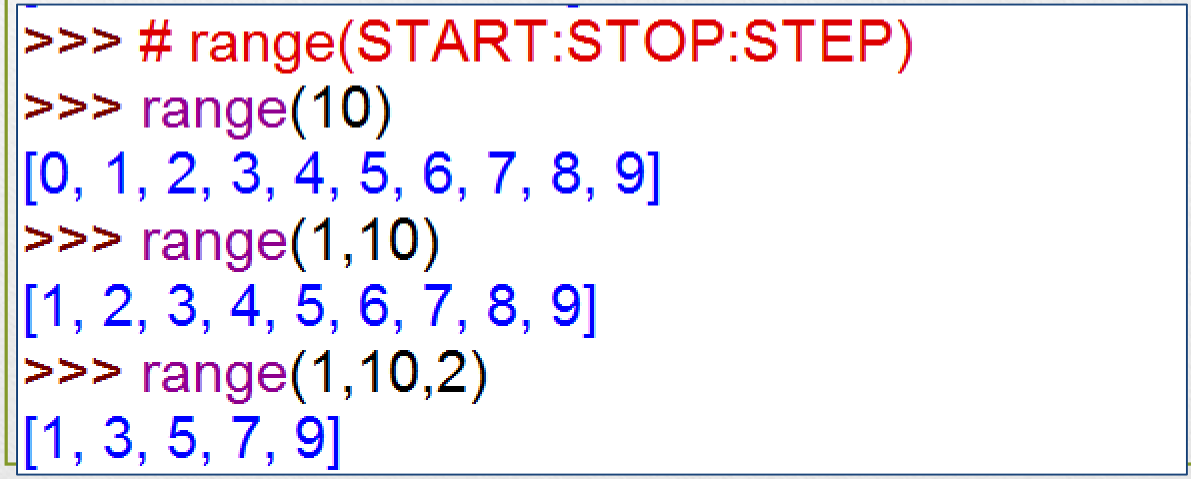


```
>>> names
['Ethans', 'Ethan', 'Ethan Tech']
>>> len(names)
3
>>> any(names)
True
>>> all(names)
True
>>> names + names
['Ethans', 'Ethan', 'Ethan Tech', 'Ethans', 'Ethan', 'Ethan Tech']
>>> names * 2
['Ethans', 'Ethan', 'Ethan Tech', 'Ethans', 'Ethan', 'Ethan Tech']
```

List Slicing and range function

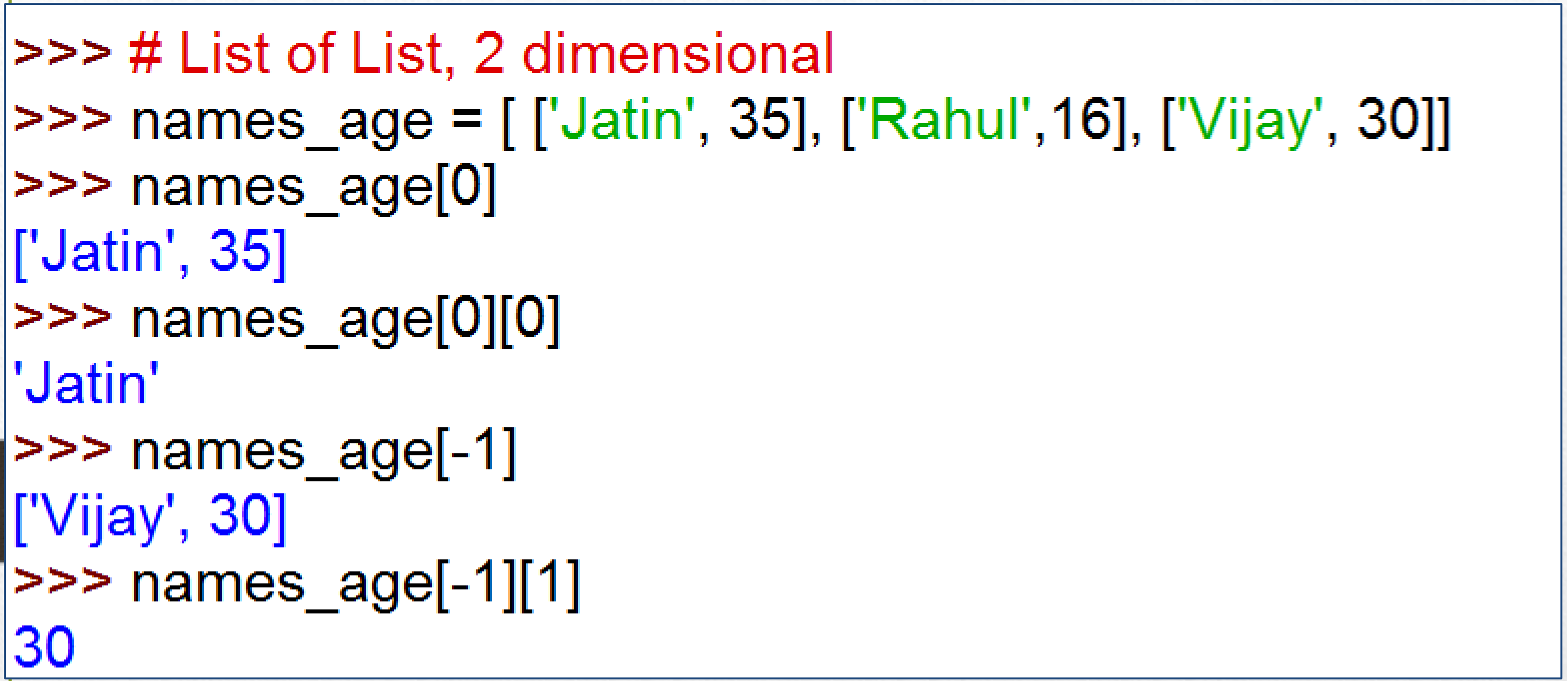


```
>>> names
['Ethans', 'Ethan', 'Ethan Tech']
>>> # names[START:STOP:STEP]
>>>
>>> names[0:]
['Ethans', 'Ethan', 'Ethan Tech']
>>> names[0:2]
['Ethans', 'Ethan']
>>> names[-1:-3]
[]
>>> names[-1:-3:-1]
['Ethan Tech', 'Ethan']
```



```
>>> # range(START:STOP:STEP)
>>> range(10)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> range(1,10)
[1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> range(1,10,2)
[1, 3, 5, 7, 9]
```

List Data Structure

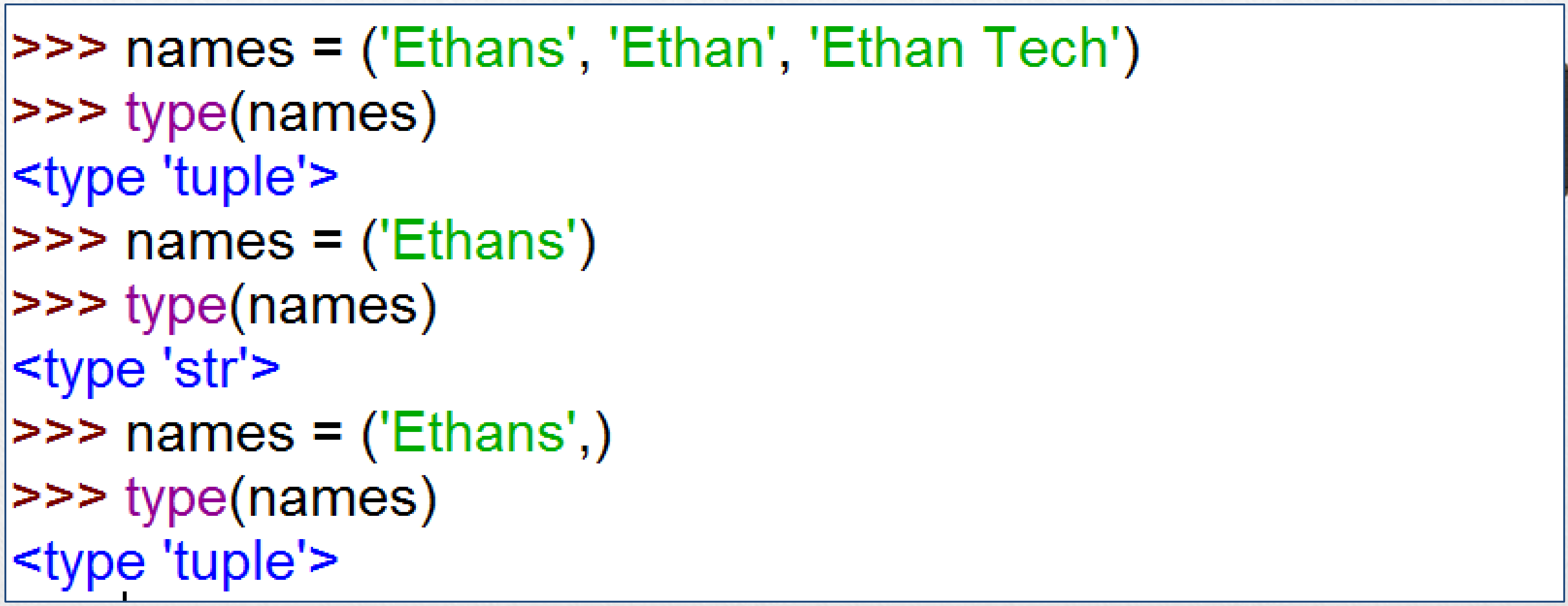


```
>>> # List of List, 2 dimensional
>>> names_age = [ ['Jatin', 35], ['Rahul',16], ['Vijay', 30]]
>>> names_age[0]
['Jatin', 35]
>>> names_age[0][0]
'Jatin'
>>> names_age[-1]
['Vijay', 30]
>>> names_age[-1][1]
30
```

Tuple

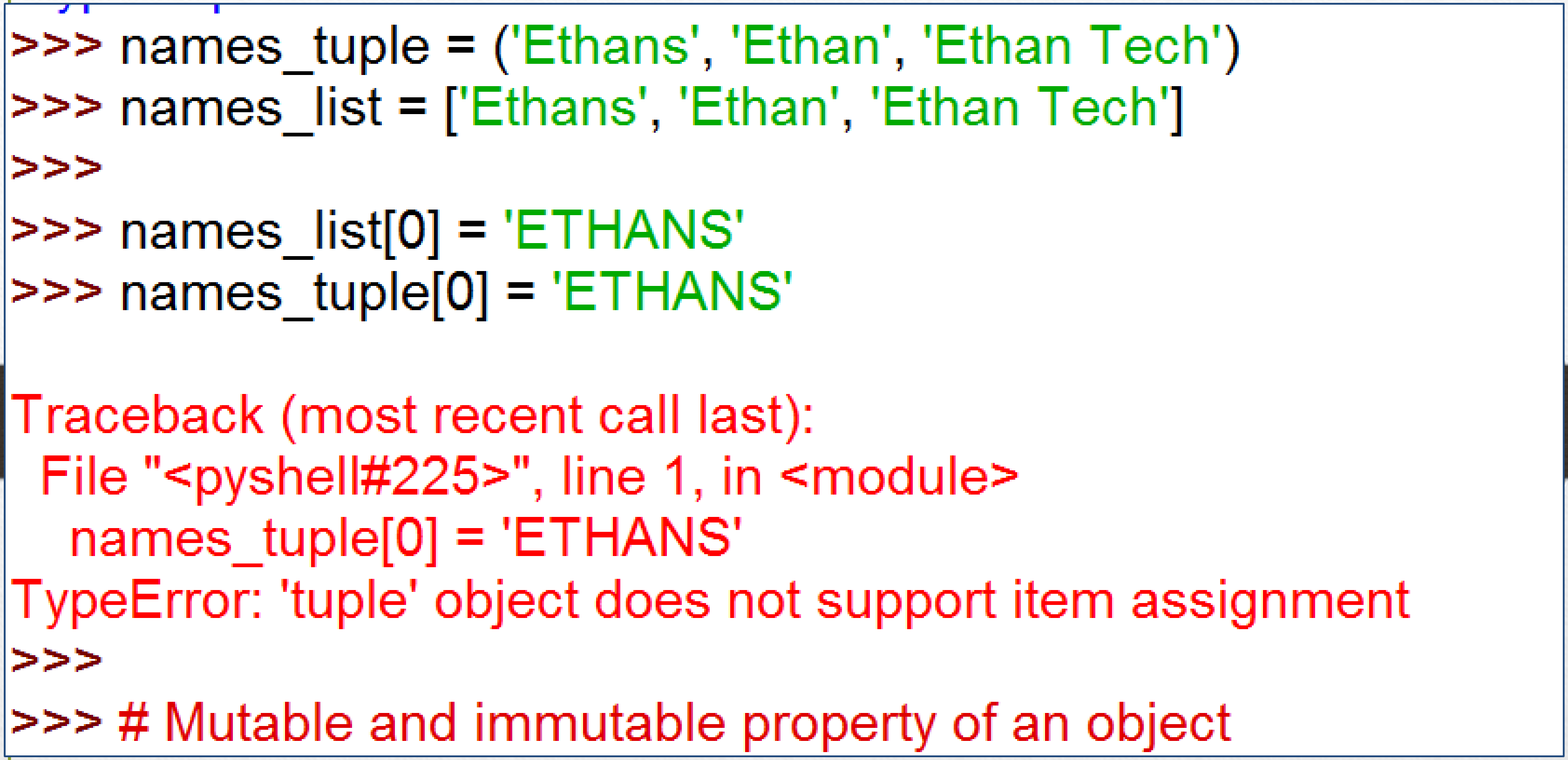
Tuple is another datatype in Python which can be written with comma-separated values (items) in a brackets.

Items in a tuple should be of same type or different type doesn't matter.



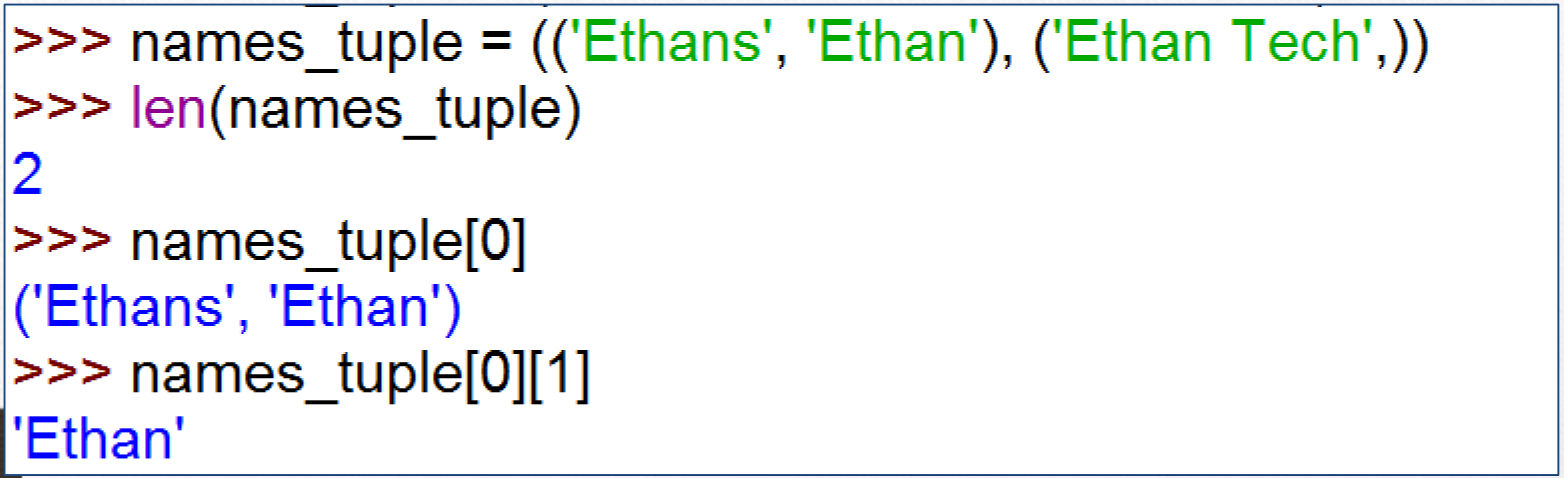
```
>>> names = ('Ethans', 'Ethan', 'Ethan Tech')
>>> type(names)
<type 'tuple'>
>>> names = ('Ethans')
>>> type(names)
<type 'str'>
>>> names = ('Ethans',)
>>> type(names)
<type 'tuple'>
```

Difference between Tuple and List

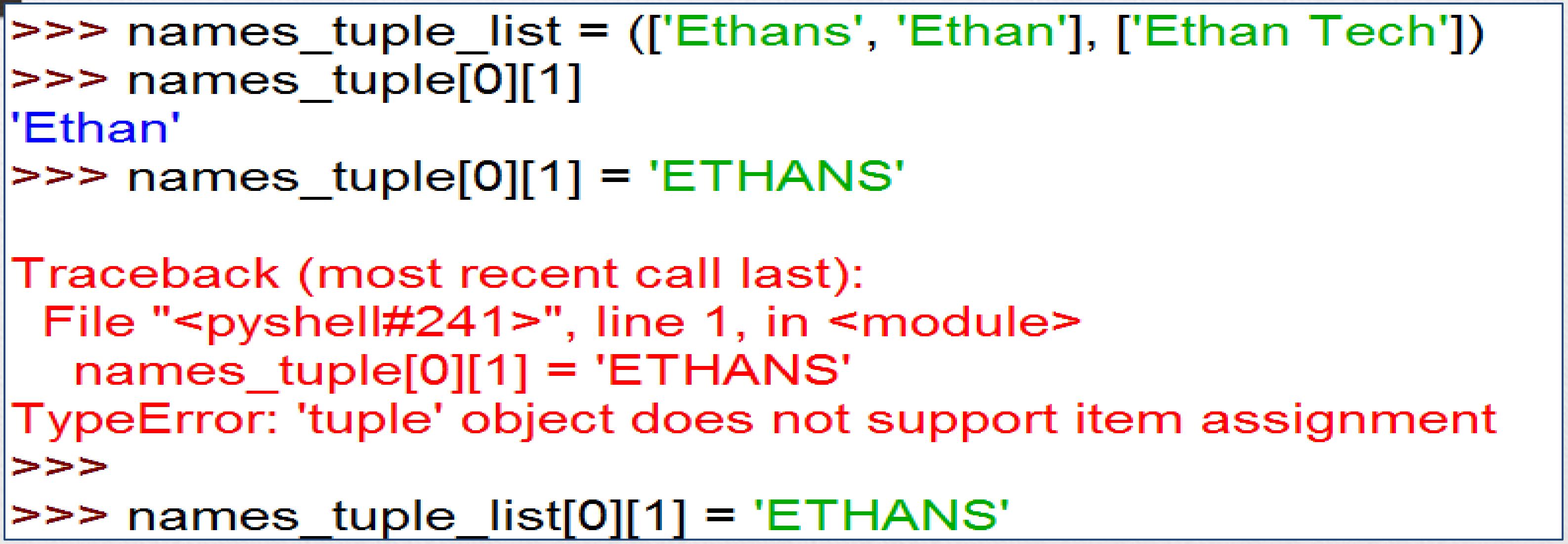


```
>>> names_tuple = ('Ethans', 'Ethan', 'Ethan Tech')
>>> names_list = ['Ethans', 'Ethan', 'Ethan Tech']
>>>
>>> names_list[0] = 'ETHANS'
>>> names_tuple[0] = 'ETHANS'

Traceback (most recent call last):
  File "<pyshell#225>", line 1, in <module>
    names_tuple[0] = 'ETHANS'
TypeError: 'tuple' object does not support item assignment
>>>
>>> # Mutable and immutable property of an object
```

Tuple Data Structure



```
>>> names_tuple = (('Ethans', 'Ethan'), ('Ethan Tech',))
>>> len(names_tuple)
2
>>> names_tuple[0]
('Ethans', 'Ethan')
>>> names_tuple[0][1]
'Ethan'
```

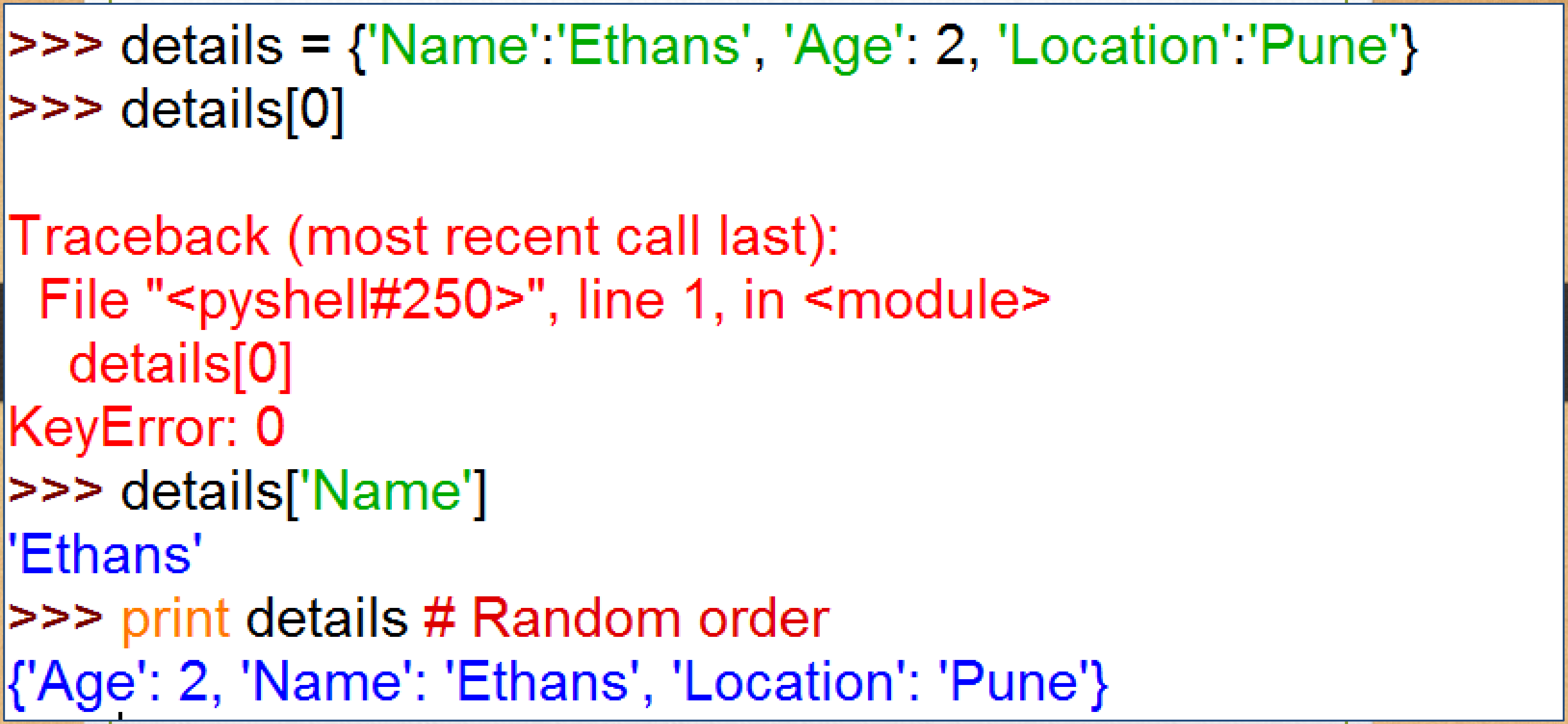


```
>>> names_tuple_list = (['Ethans', 'Ethan'], ['Ethan Tech'])
>>> names_tuple[0][1]
'Ethan'
>>> names_tuple[0][1] = 'ETHANS'

Traceback (most recent call last):
  File "<pyshell#241>", line 1, in <module>
    names_tuple[0][1] = 'ETHANS'
TypeError: 'tuple' object does not support item assignment
>>>
>>> names_tuple_list[0][1] = 'ETHANS'
```

www.ethans.co.i
n

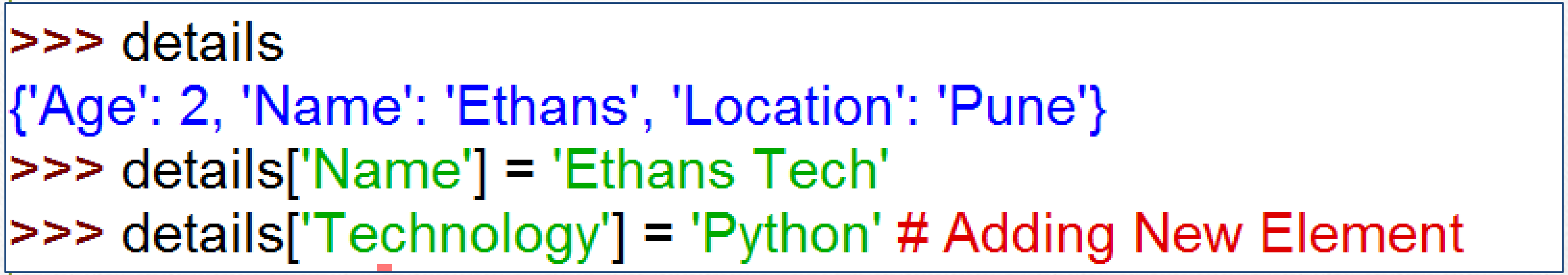
Dictionary

Another builtin datatype in Python where each key is separated from its value by a colon (:), the items are separated by commas, and the whole thing is enclosed in curly braces.

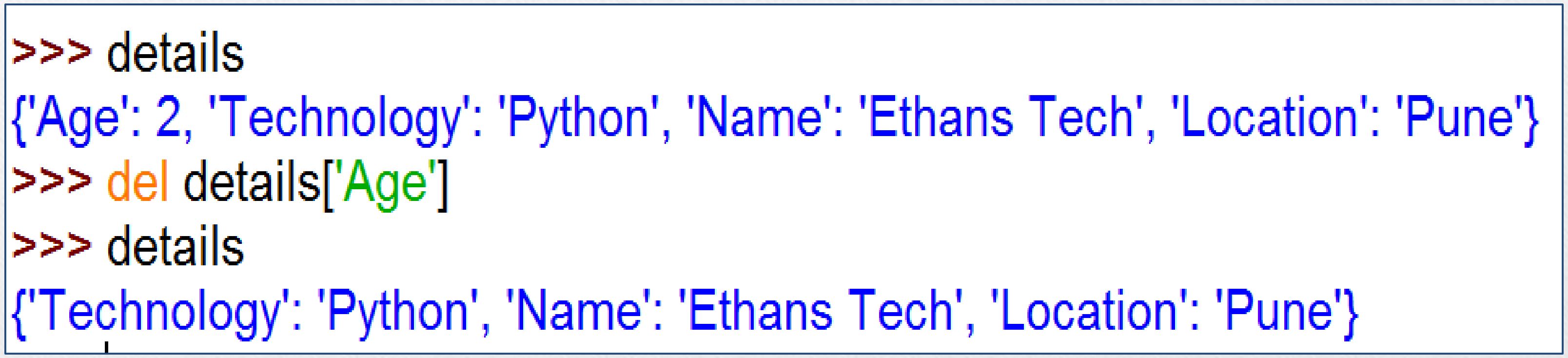


```
>>> details = {'Name':'Ethans', 'Age': 2, 'Location':'Pune'}
>>> details[0]

Traceback (most recent call last):
  File "<pyshell#250>", line 1, in <module>
    details[0]
KeyError: 0
>>> details['Name']
'Ethans'
>>> print details # Random order
{'Age': 2, 'Name': 'Ethans', 'Location': 'Pune'}
```

Add and Remove Dictionary Item

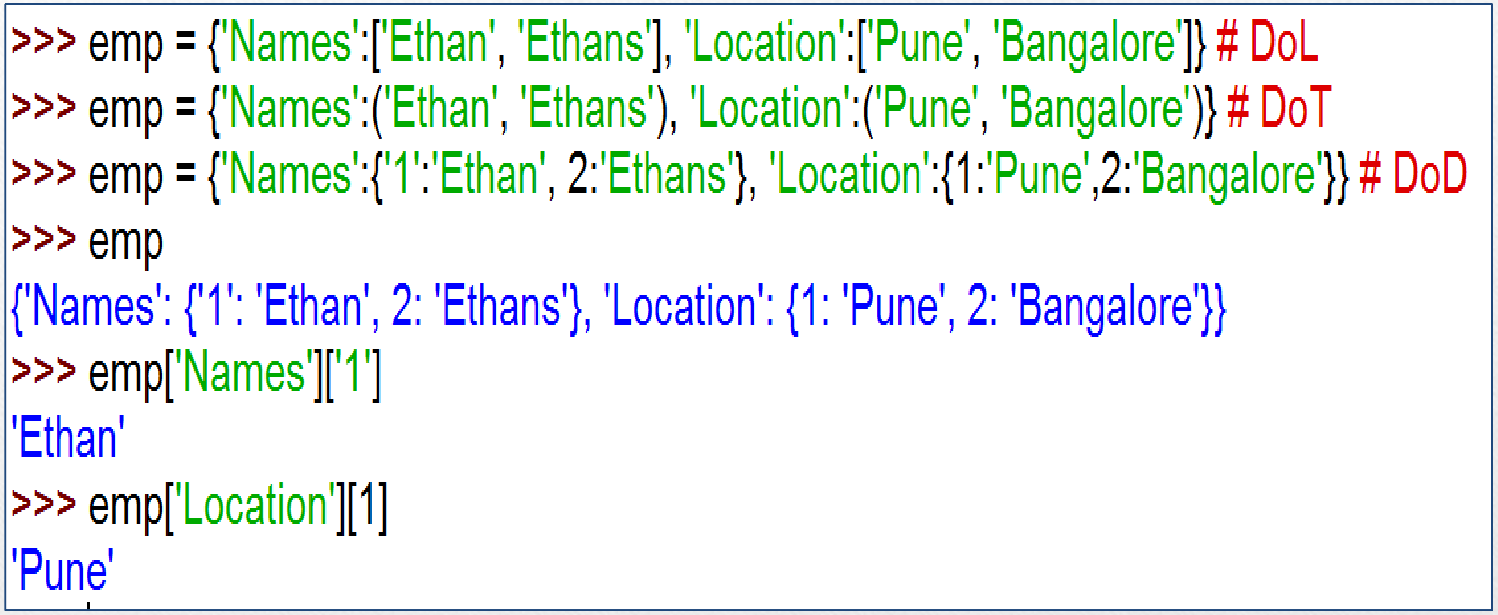


```
>>> details
{'Age': 2, 'Name': 'Ethans', 'Location': 'Pune'}
>>> details['Name'] = 'Ethans Tech'
>>> details['Technology'] = 'Python' # Adding New Element
```



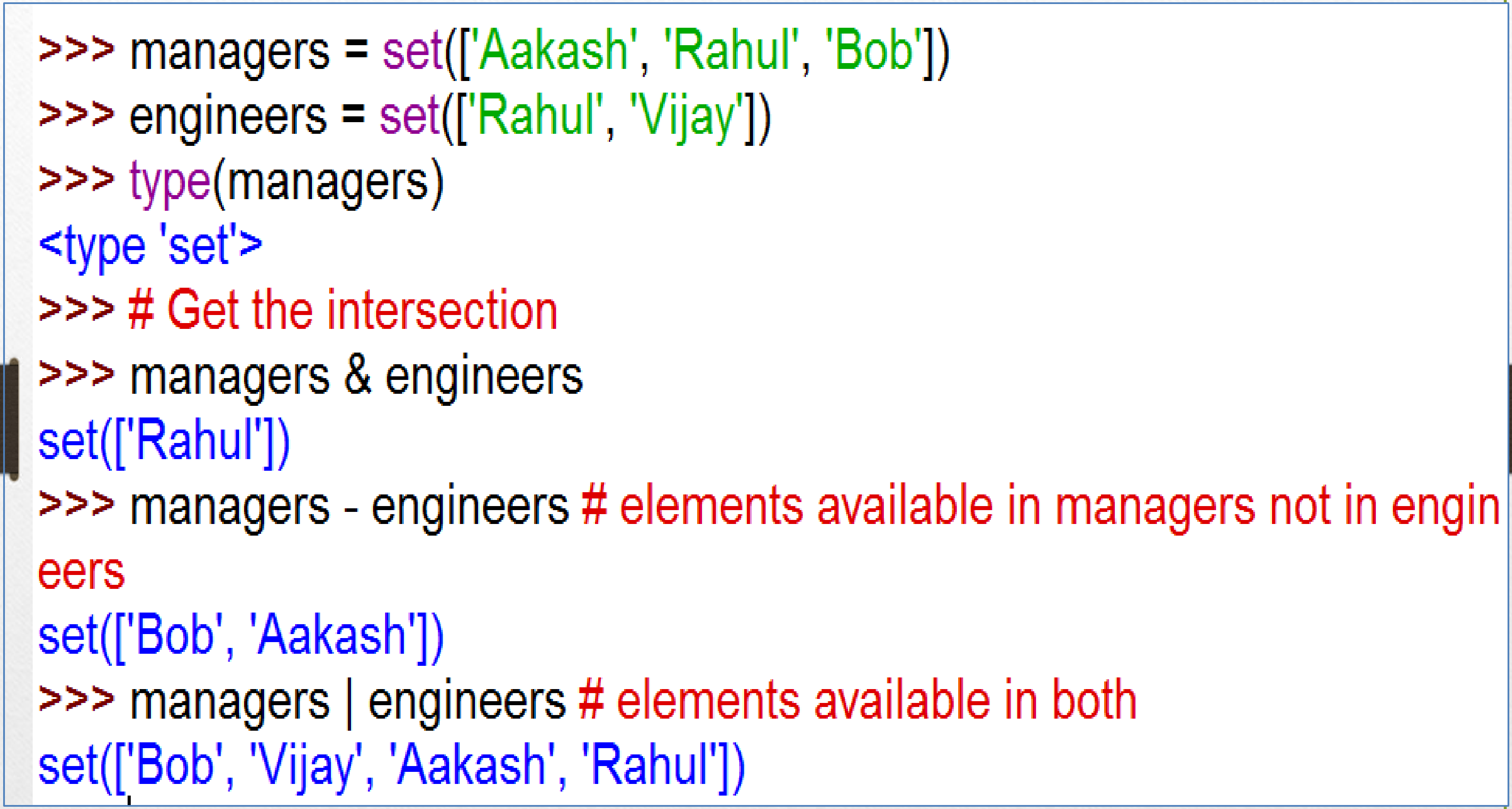
```
>>> details
{'Age': 2, 'Technology': 'Python', 'Name': 'Ethans Tech', 'Location': 'Pune'}
>>> del details['Age']
>>> details
{'Technology': 'Python', 'Name': 'Ethans Tech', 'Location': 'Pune'}
```

Dictionaries data Structure



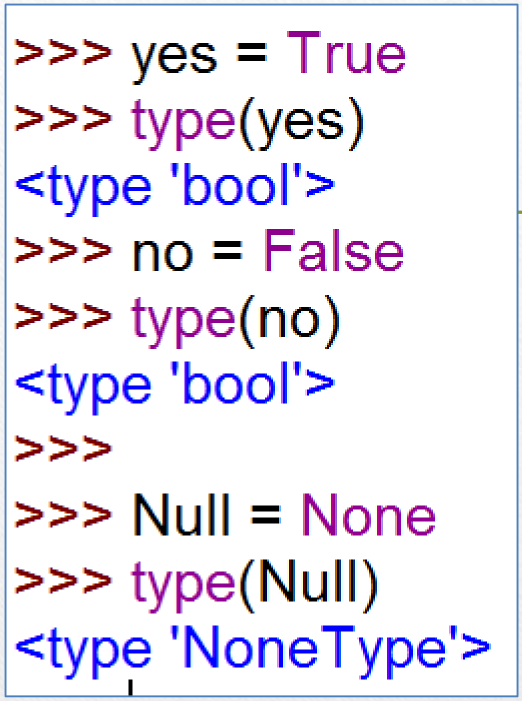
```
>>> emp = {'Names':['Ethan', 'Ethans'], 'Location':['Pune', 'Bangalore']} # DoL
>>> emp = {'Names':('Ethan', 'Ethans'), 'Location':('Pune', 'Bangalore')} # DoT
>>> emp = {'Names':{'1':'Ethan', 2:'Ethans'}, 'Location':{1:'Pune',2:'Bangalore'}} # DoD
>>> emp
{'Names': {'1': 'Ethan', 2: 'Ethans'}, 'Location': {1: 'Pune', 2: 'Bangalore'}}
>>> emp['Names']['1']
'Ethan'
>>> emp['Location'][1]
'Pune'
```

Set Object



```
>>> managers = set(['Aakash', 'Rahul', 'Bob'])
>>> engineers = set(['Rahul', 'Vijay'])
>>> type(managers)
<type 'set'>
>>> # Get the intersection
>>> managers & engineers
set(['Rahul'])
>>> managers - engineers # elements available in managers not in engin
eers
set(['Bob', 'Aakash'])
>>> managers | engineers # elements available in both
set(['Bob', 'Vijay', 'Aakash', 'Rahul'])
```

Boolean and None Object



```
>>> yes = True
>>> type(yes)
<type 'bool'>
>>> no = False
>>> type(no)
<type 'bool'>
>>>
>>> Null = None
>>> type(Null)
<type 'NoneType'>
```

True and False are the pre defined objects in Python, when comparing the value or doing comparison operation Python returns either True or False.

None is another object in Python.
It is similar as NULL in database.

Membership Operators

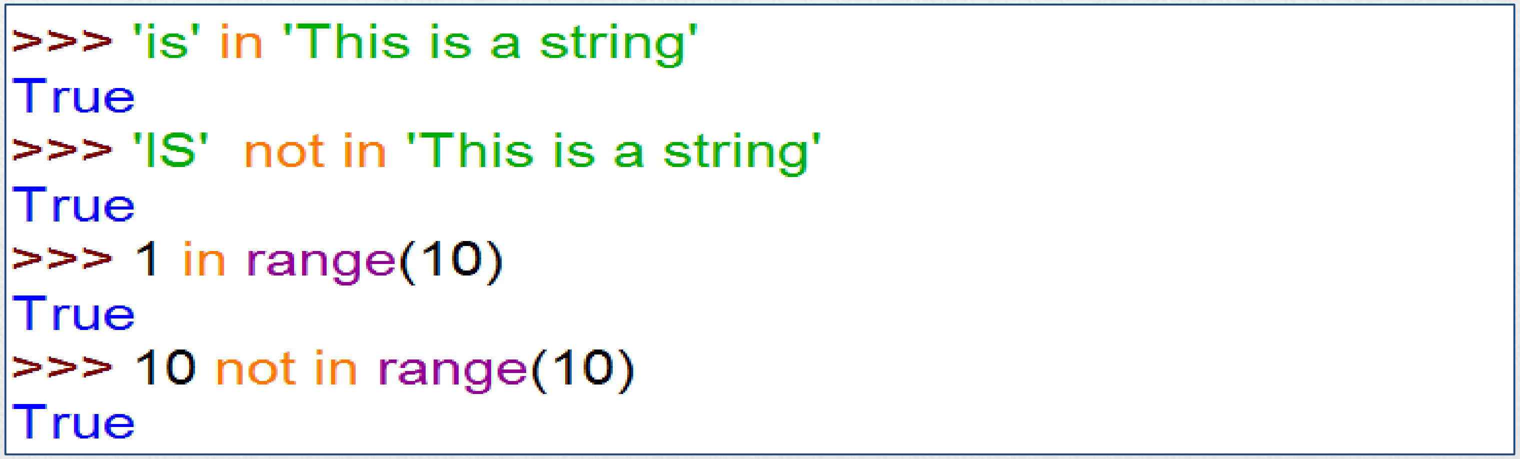
Python membership operators test for membership in a sequence or an iterable, such as strings, lists, or tuples.

`in`

Evaluates to `true` if it finds a variable in the specified sequence and `false` otherwise.

`not in`

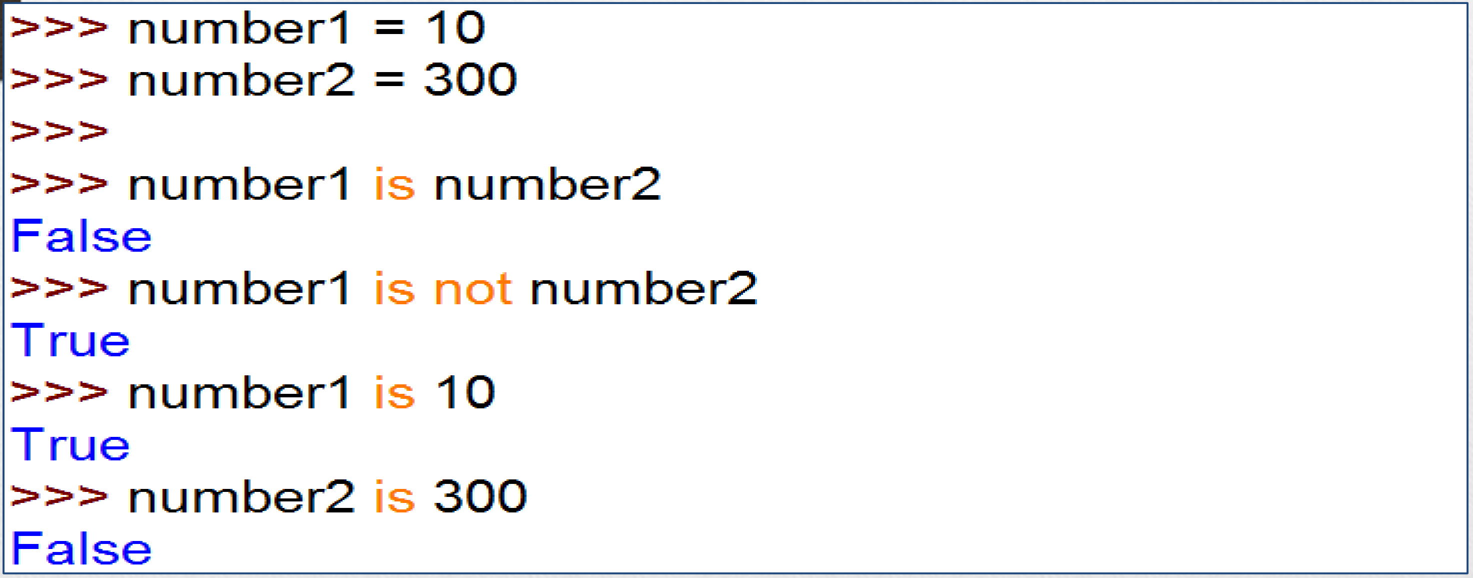
Evaluates to `true` if it does not finds a variable in the specified sequence and `false` otherwise.



```
>>> 'is' in 'This is a string'
True
>>> 'IS'  not in 'This is a string'
True
>>> 1 in range(10)
True
>>> 10 not in range(10)
True
```

Python Identity Operator

`is`	`is not`
Evaluates to `true` if the variables on either side of the operator point to the same object and `false` otherwise.	Evaluates to `false` if the variables on either side of the operator point to the same object and `true` otherwise.

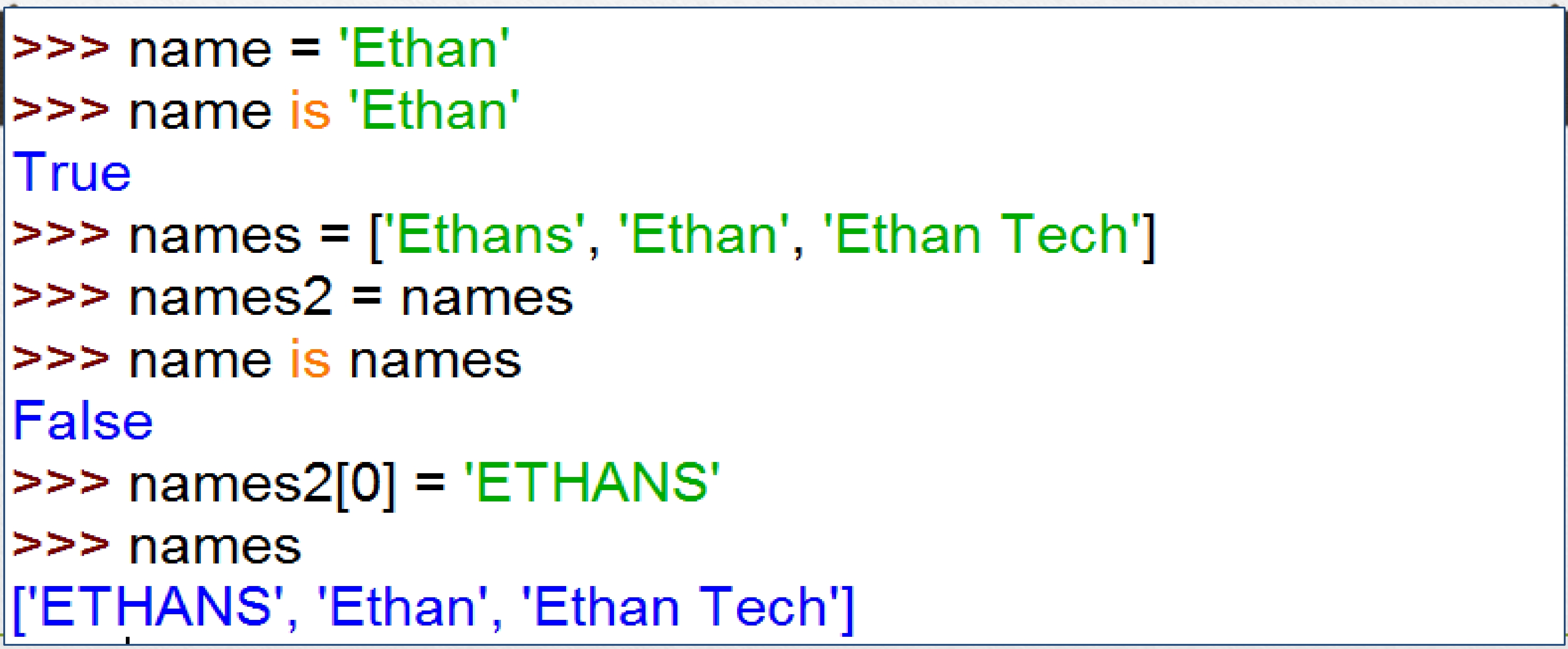


```
>>> number1 = 10
>>> number2 = 300
>>>
>>> number1 is number2
False
>>> number1 is not number2
True
>>> number1 is 10
True
>>> number2 is 300
False
```

www.ethans.co.i
n

Python Identity Operator

`is`	`is not`
Evaluates to `True` if the variables on either side of the operator point to the same object and `False` otherwise.	Evaluates to `False` if the variables on either side of the operator point to the same object and `True` otherwise.



```
>>> name = 'Ethan'
>>> name is 'Ethan'
True
>>> names = ['Ethans', 'Ethan', 'Ethan Tech']
>>> names2 = names
>>> name is names
False
>>> names2[0] = 'ETHANS'
>>> names
['ETHANS', 'Ethan', 'Ethan Tech']
```

Quiz:

1 – What will be the output of below program?

```
>>> sum([1,2,3,4,5])
```

2 – What will be the output of below program?

```
>>> name = 'Ethans Technologies'
>>> name[0:6]
```

3 – What will be the output of below program?

```
>>> name = 'Ethans Technologies‘
>>> name[-1:0]
```

Interesting Fact

Raspberry Pi is a card-sized, inexpensive microcomputer that is being used for a surprising range of exciting do-it-yourself stuff such as robots, remote-controlled cars, and video game consoles.

With Python as its main programming language, the Raspberry Pi is being used even by programmers to build radios, cameras, arcade machines, and pet feeders!

With Raspberry Pi mania on the uptrend, there are countless DIY projects,
tutorials, and books to choose from online.

These will help you branch out from your “hello world” starter programs to something you can truly be proud of.

Have Further Questions?