Python 201

Python 201

- Data Types and the ‘type’ type
- Tuples
- Lists
- Ranges and X Ranges
- for loops

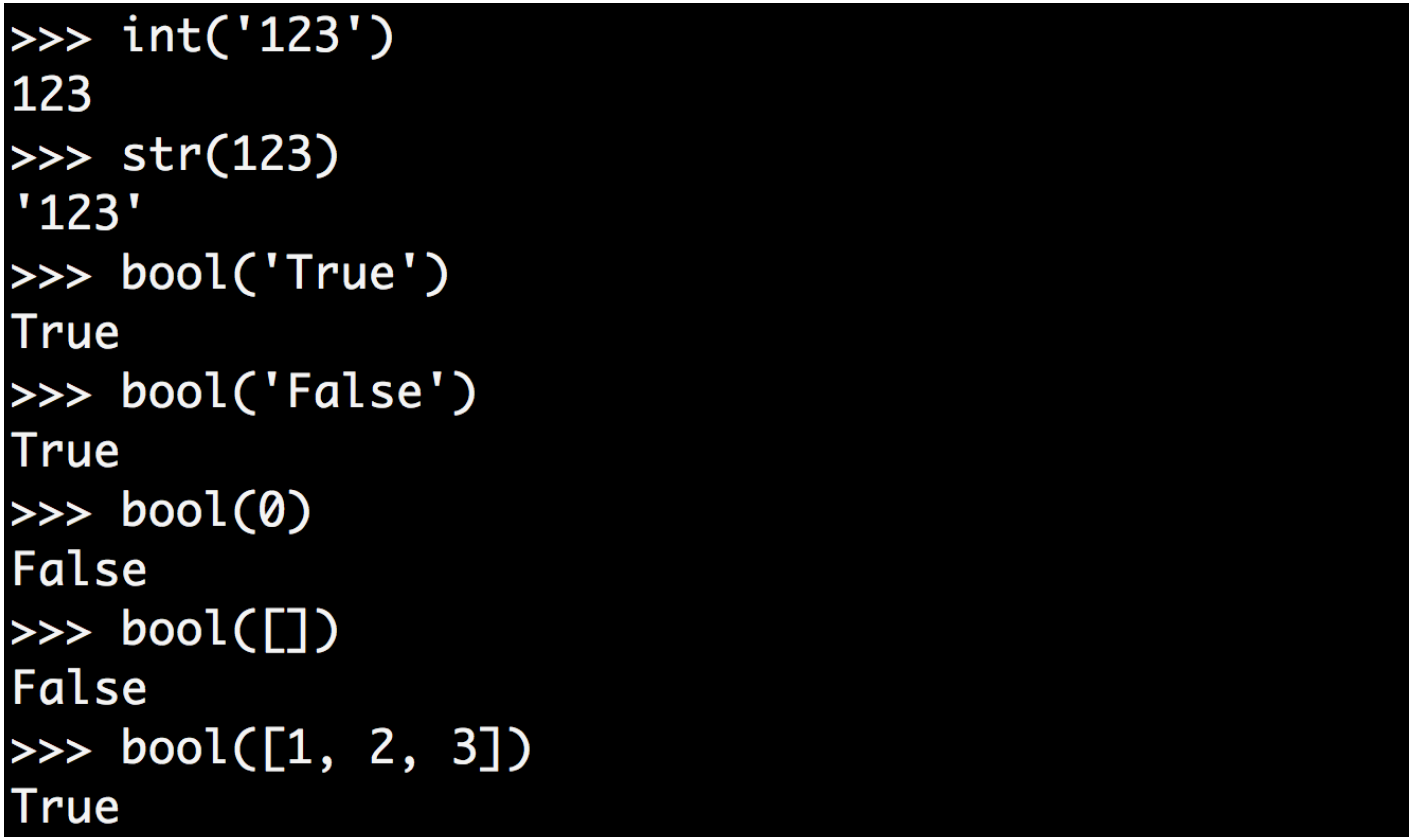
Python Data Types

- Integers (int)
- Floating point numbers (float)
- Strings (str)
- Booleans (bool)

More Python Data Types

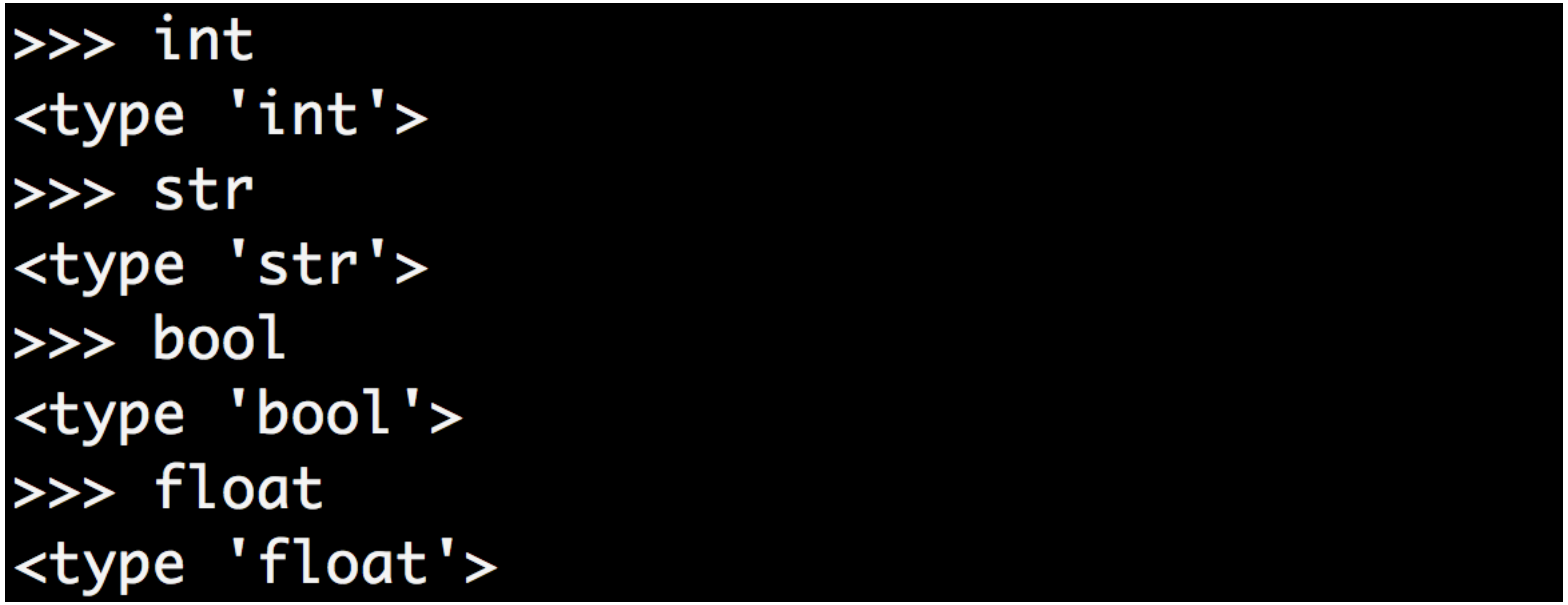
- Tuples (tuple)
- Lists (list)
- X Ranges (xrange)

The Type Conversion Functions



```
>>> int('123')
123
>>> str(123)
'123'
>>> bool('True')
True
>>> bool('False')
True
>>> bool(0)
False
>>> bool([])
False
>>> bool([1, 2, 3])
True
```

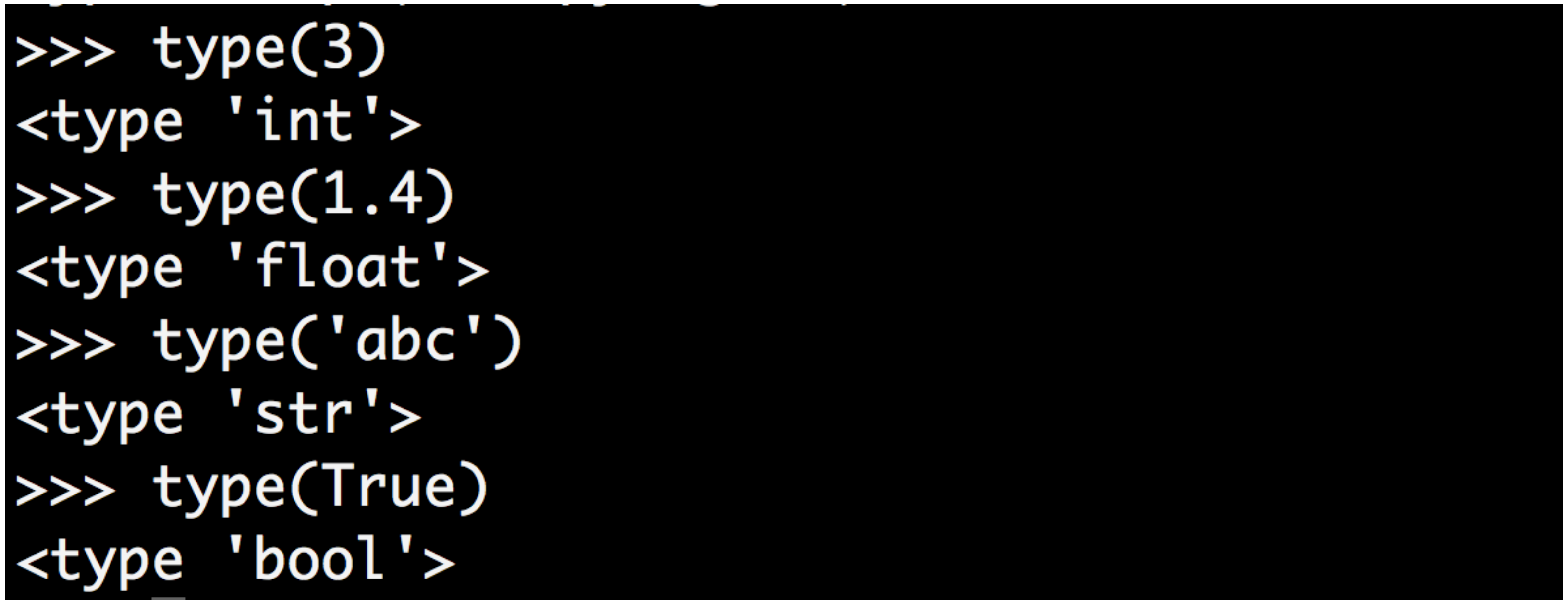
But what are they?



```
>>> int
<type 'int'>
>>> str
<type 'str'>
>>> bool
<type 'bool'>
>>> float
<type 'float'>
```

They are types

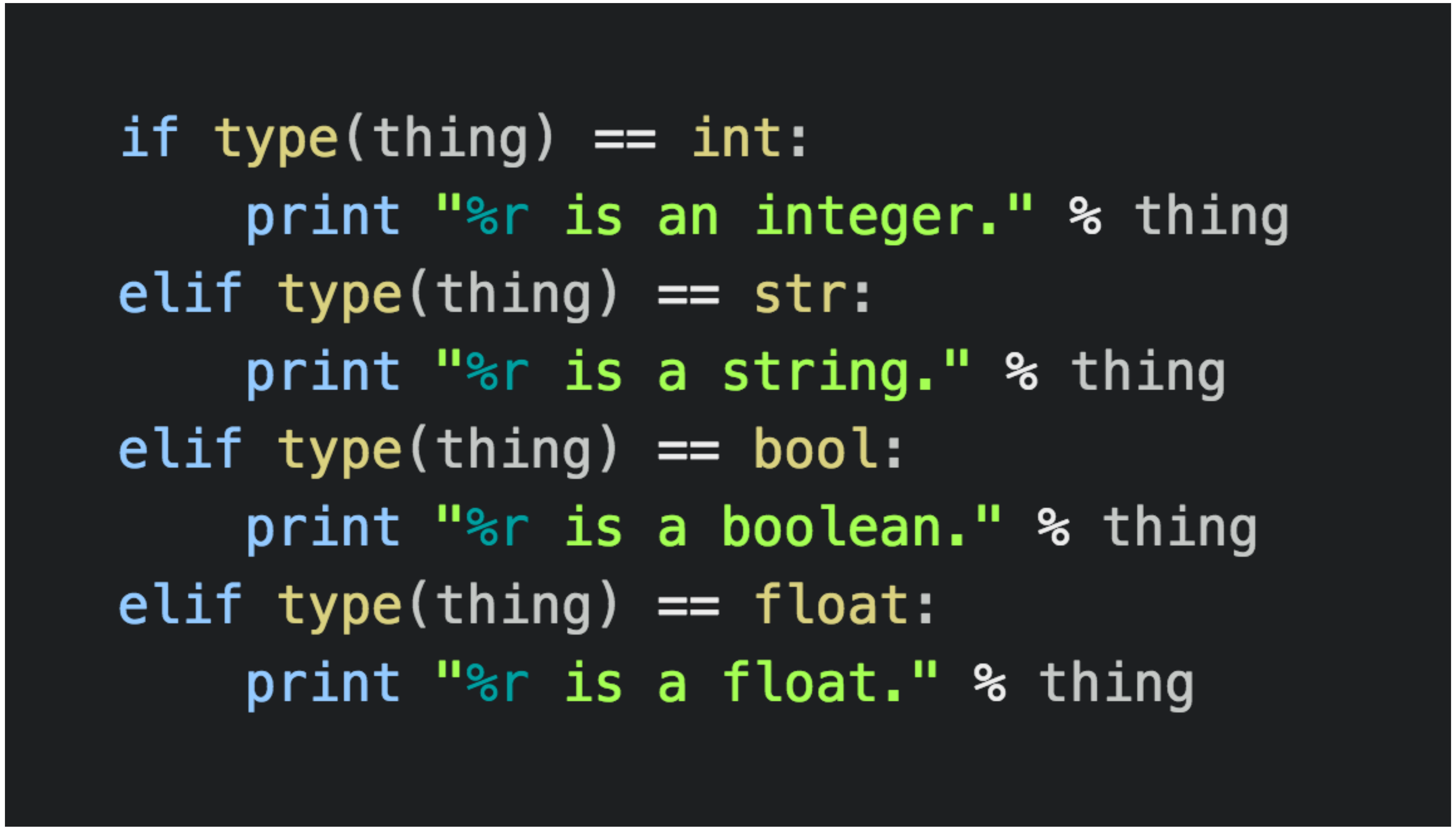
The `type` function



```
>>> type(3)
<type 'int'>
>>> type(1.4)
<type 'float'>
>>> type('abc')
<type 'str'>
>>> type(True)
<type 'bool'>
```

Will tell you what the type of a value is.

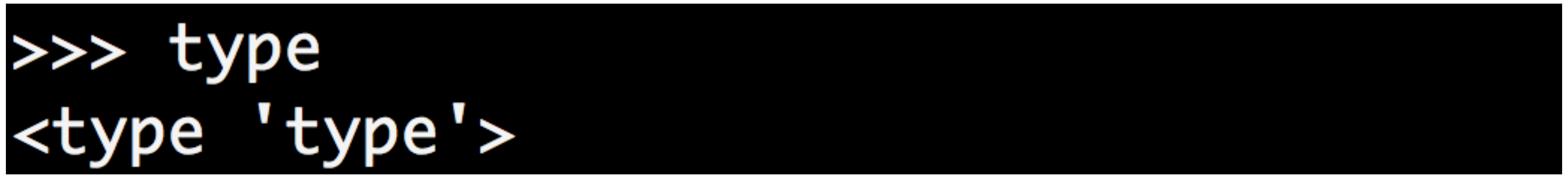
The type function



```
if type(thing) == int:
    print "%r is an integer." % thing
elif type(thing) == str:
    print "%r is a string." % thing
elif type(thing) == bool:
    print "%r is a boolean." % thing
elif type(thing) == float:
    print "%r is a float." % thing
```

You can use if statements to test for the type of a value.

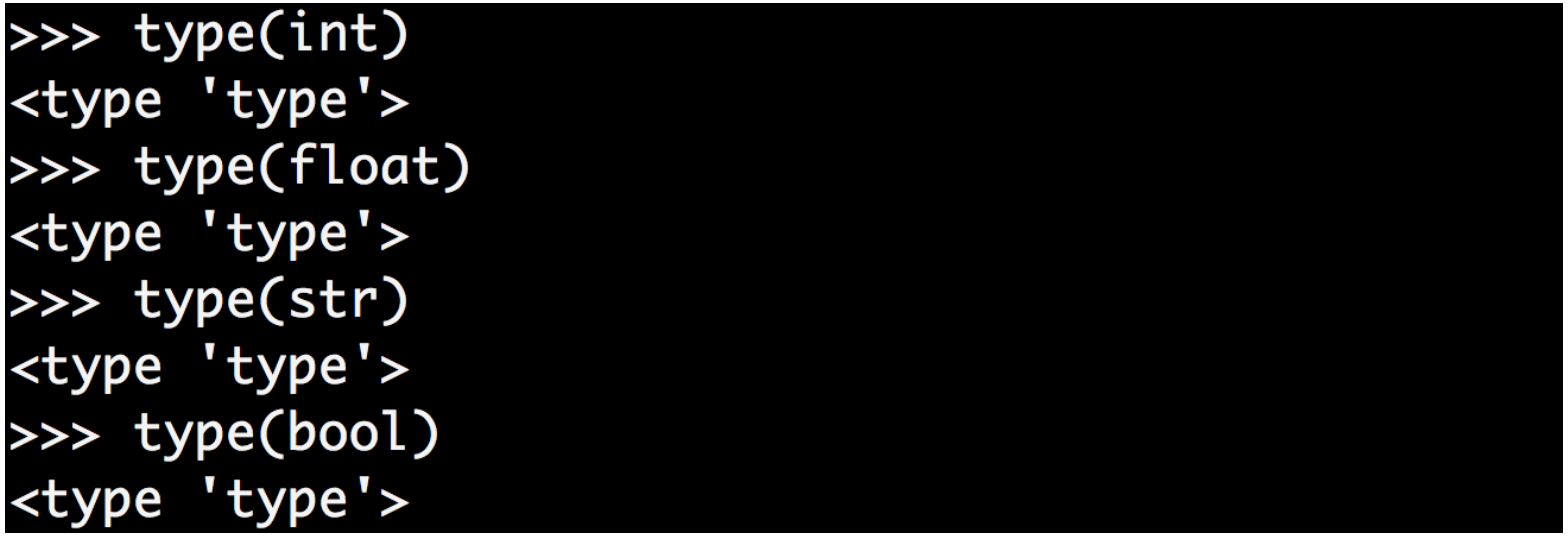
What is `type`?



```
>>> type
<type 'type'>
```

type is a type.

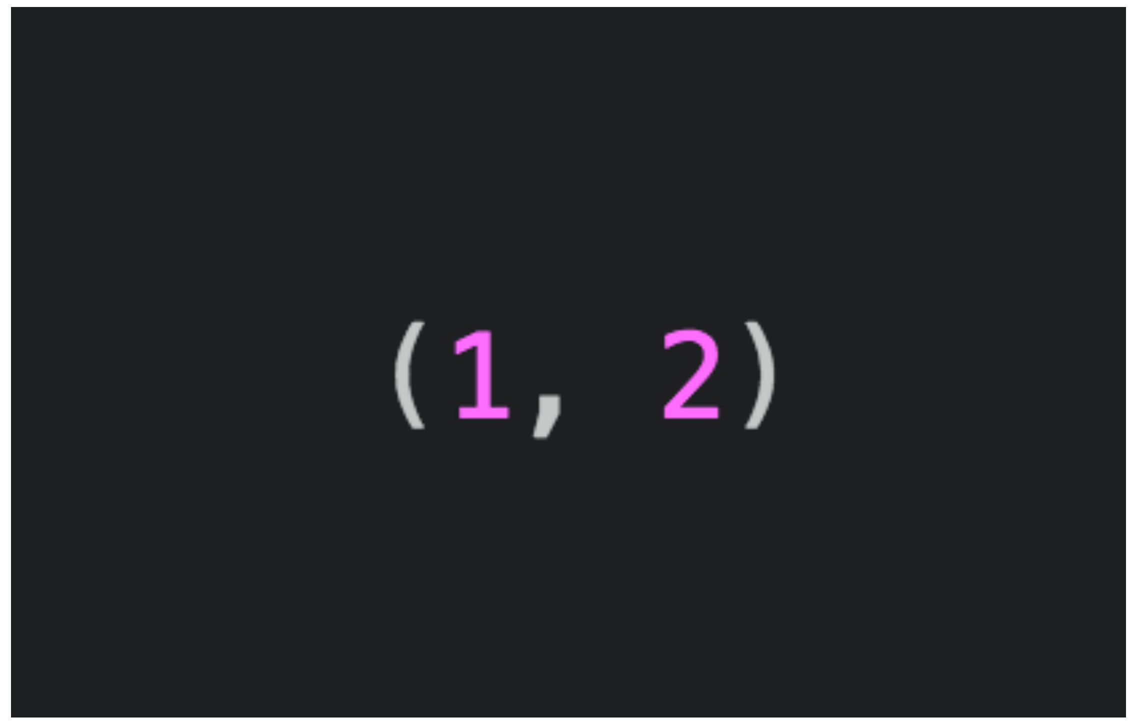
What things are types?



```
>>> type(int)
<type 'type'>
>>> type(float)
<type 'type'>
>>> type(str)
<type 'type'>
>>> type(bool)
<type 'type'>
```

Tuples

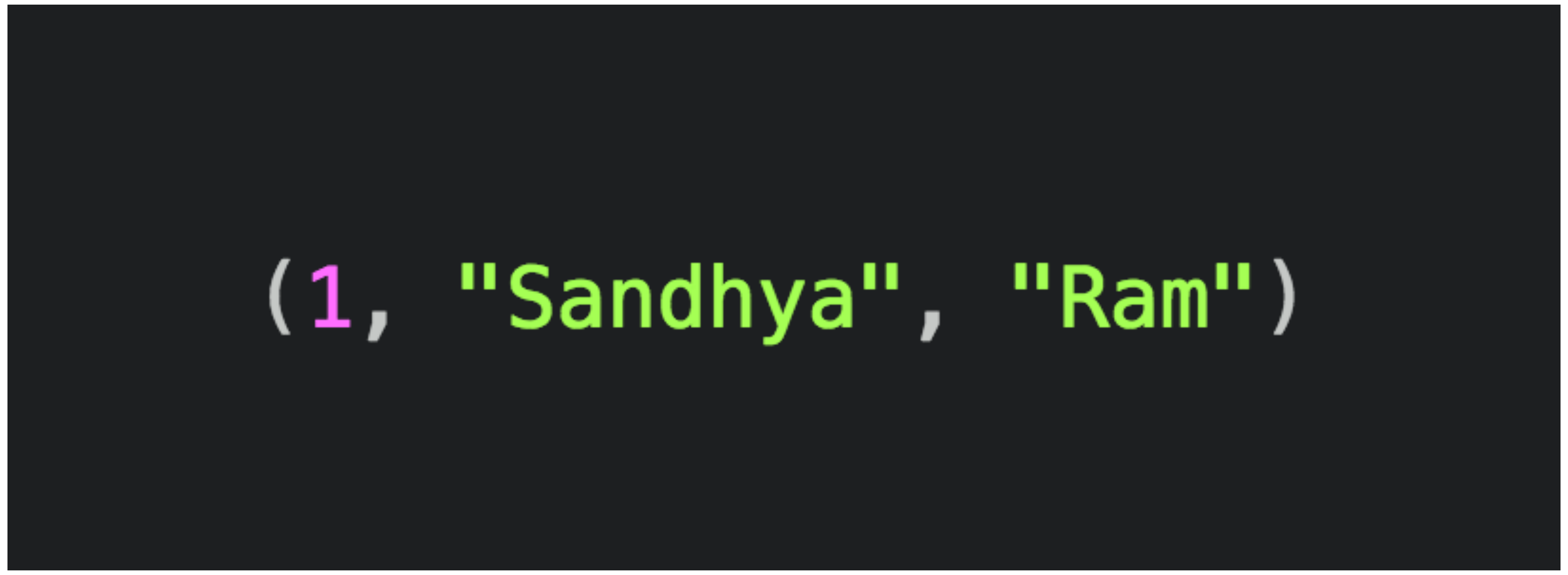
Tuples



```
(1, 2)
```

Tuples group one or more values of any type.

Tuples: Don’t have to be same type



```
(1, "Sandhya", "Ram")
```

Tuples group one or more values of any type.

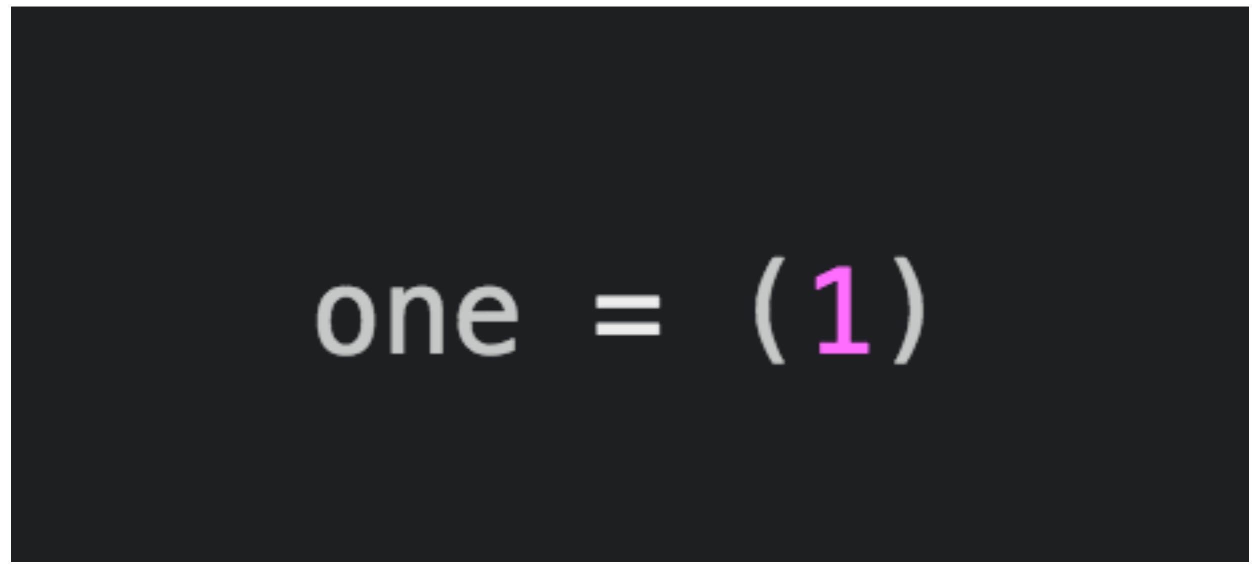
If assigning, parans are optional

```
sandhya = 1, "Sandhya", "Ram"
```

same as:

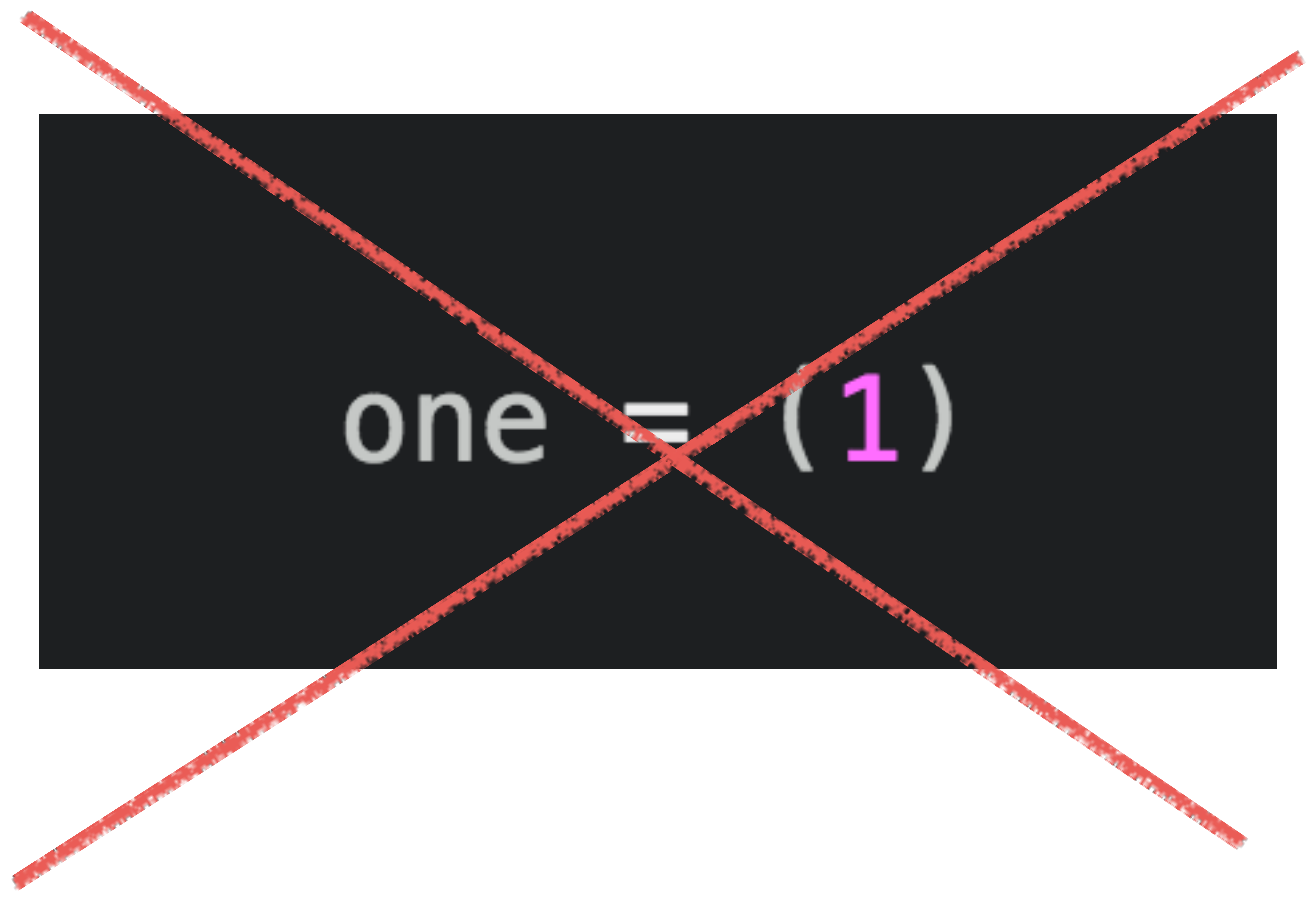
```
sandhya = (1, "Sandhya", "Ram")
```

One-Sized Tuple

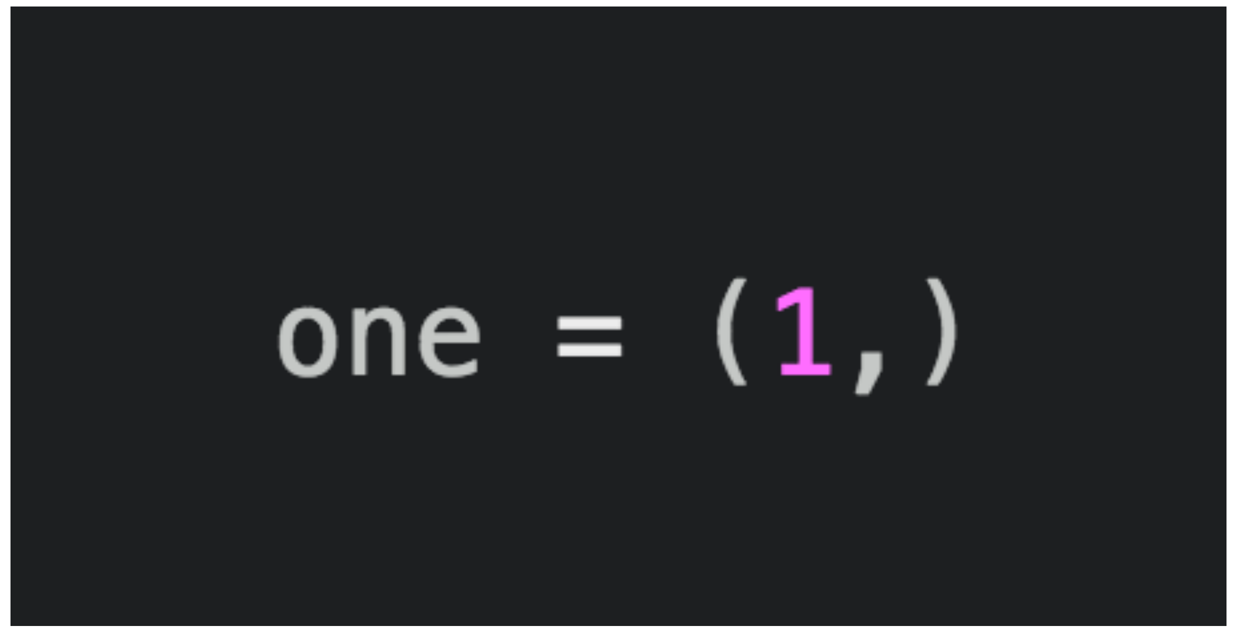


```
one = (1)
```

One-Sized Tuple

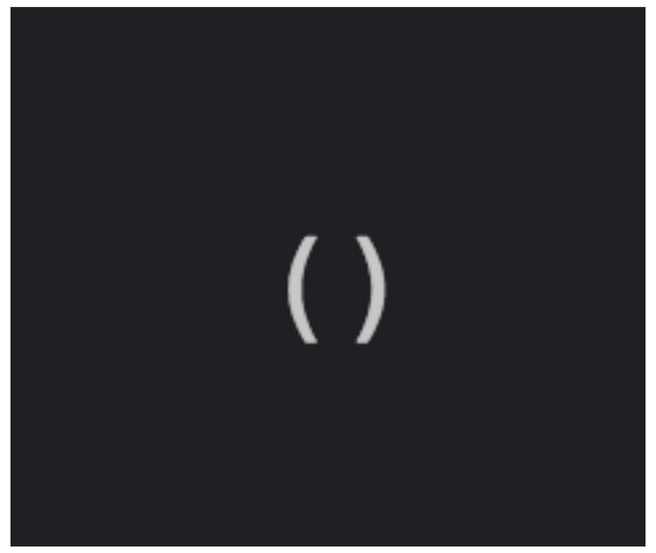


~~`one = (1)`~~



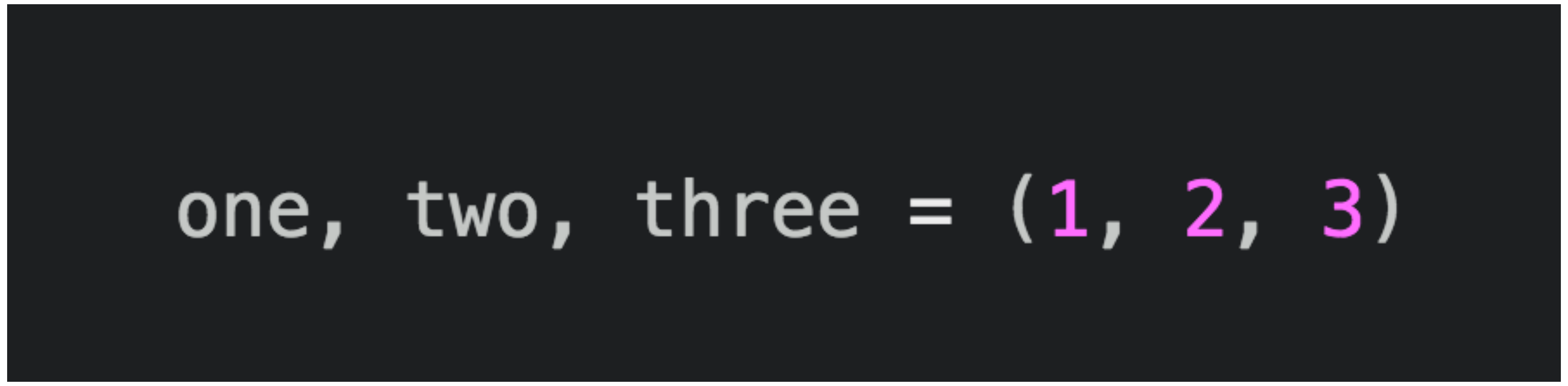
```
one = (1,)
```

Zero-sized Tuple



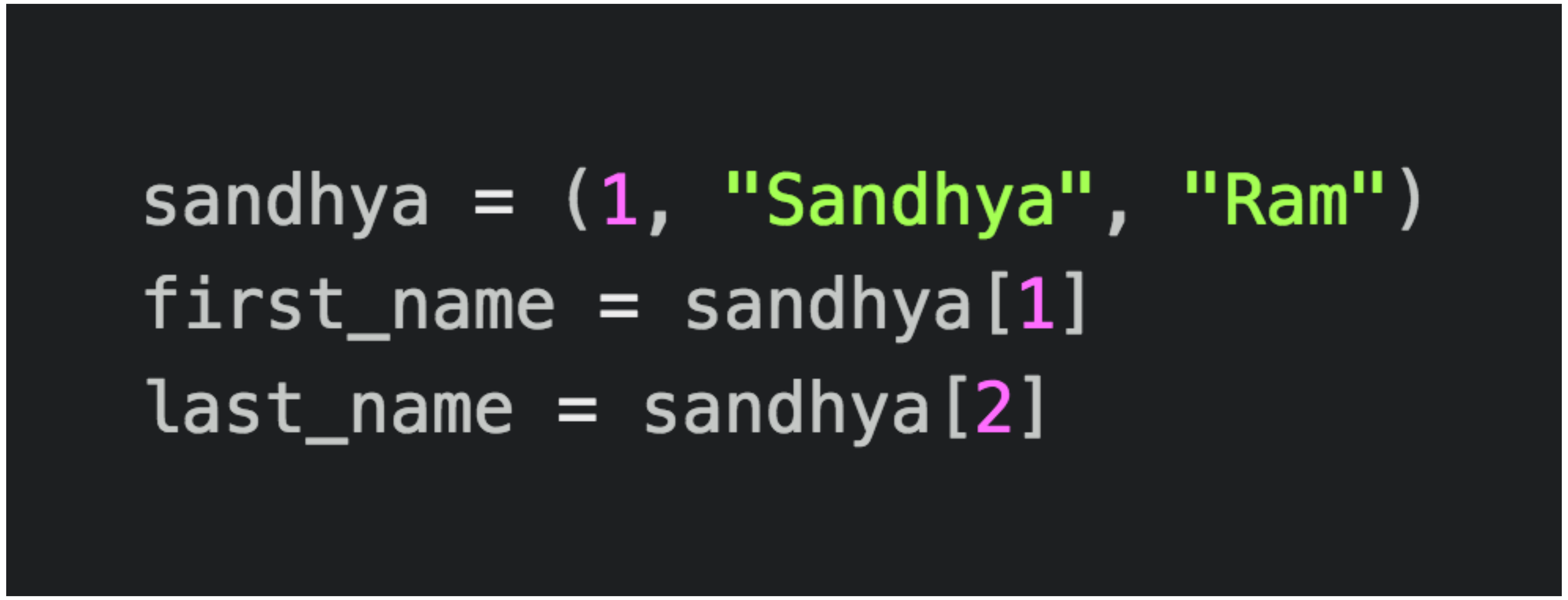
```
()
```

Destructuring Assignment



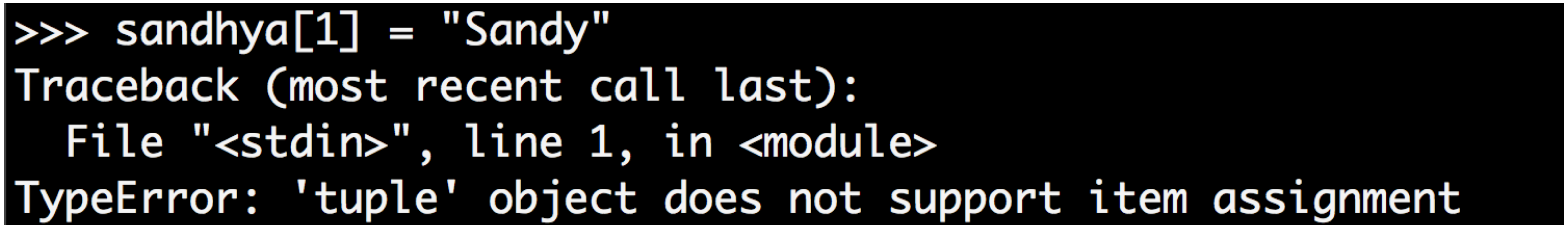
```
one, two, three = (1, 2, 3)
```

Indexing



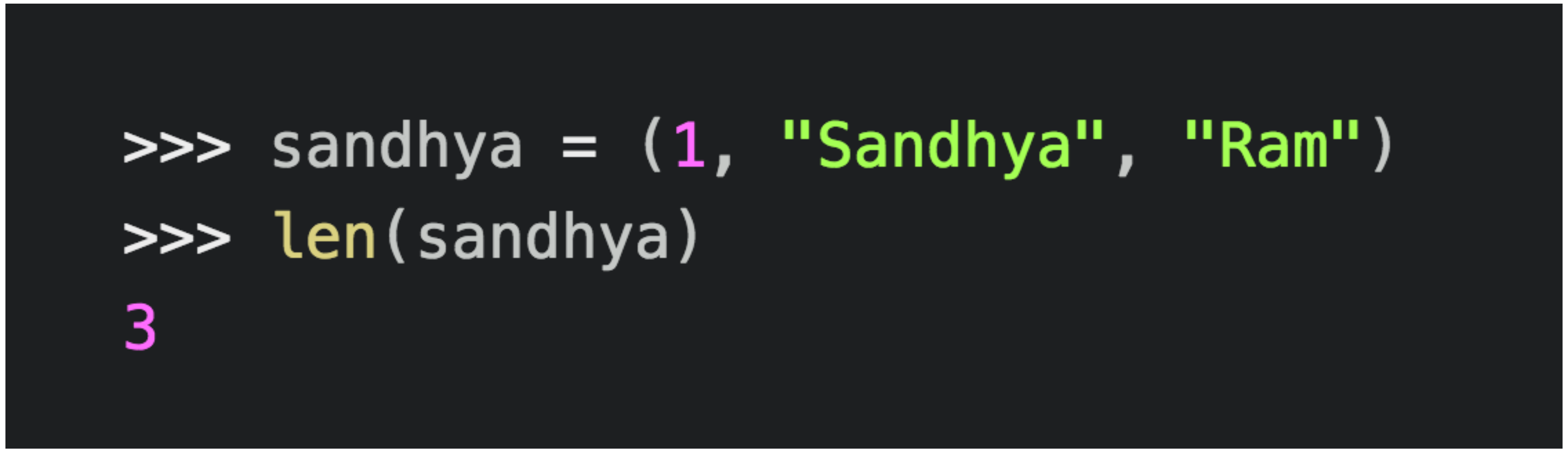
```
sandhya = (1, "Sandhya", "Ram")
first_name = sandhya[1]
last_name = sandhya[2]
```

Tuples are Immutable



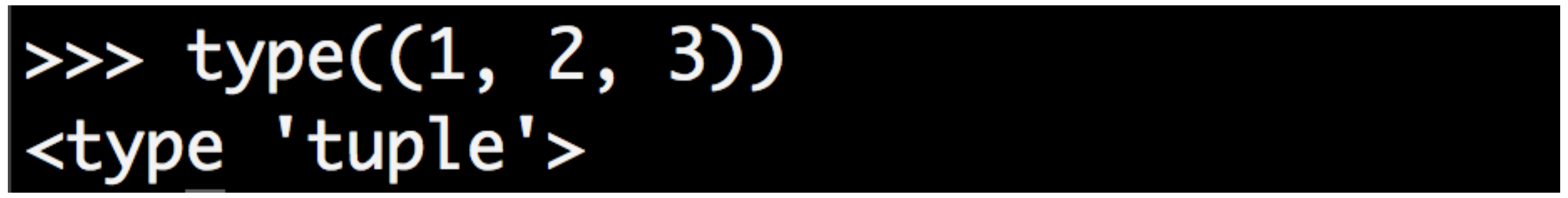
```
>>> sandhya[1] = "Sandy"
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

len() function



```
>>> sandhya = (1, "Sandhya", "Ram")
>>> len(sandhya)
3
```

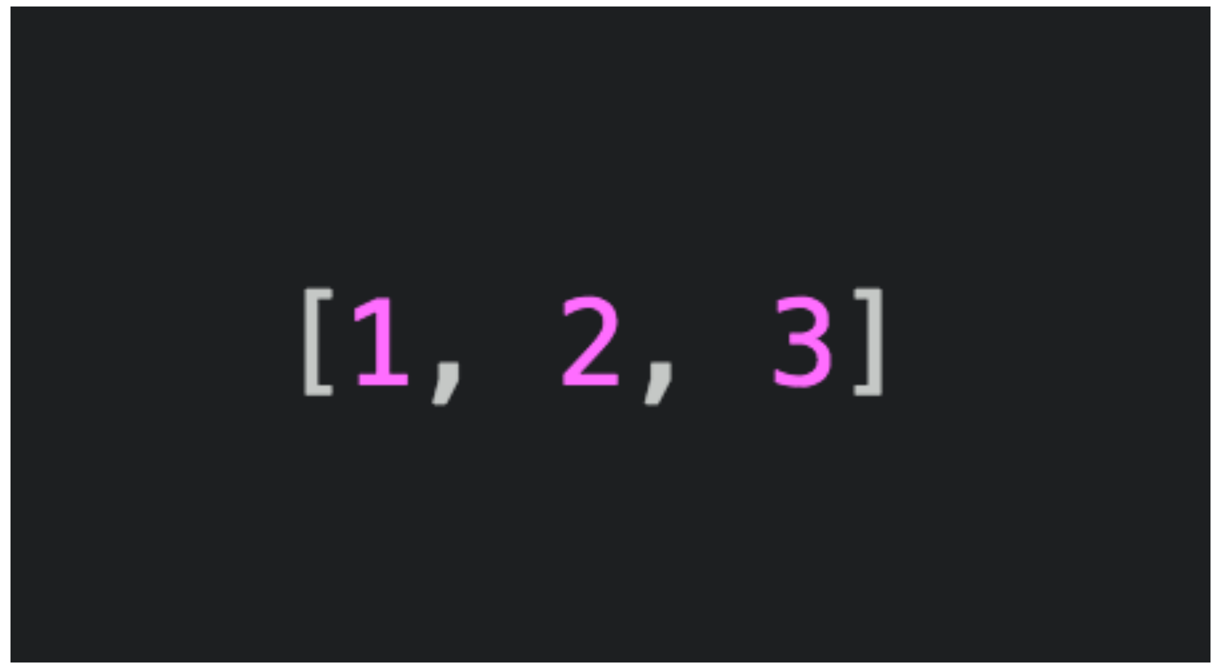
What’s a Tuple?



```
>>> type((1, 2, 3))
<type 'tuple'>
```

Lists

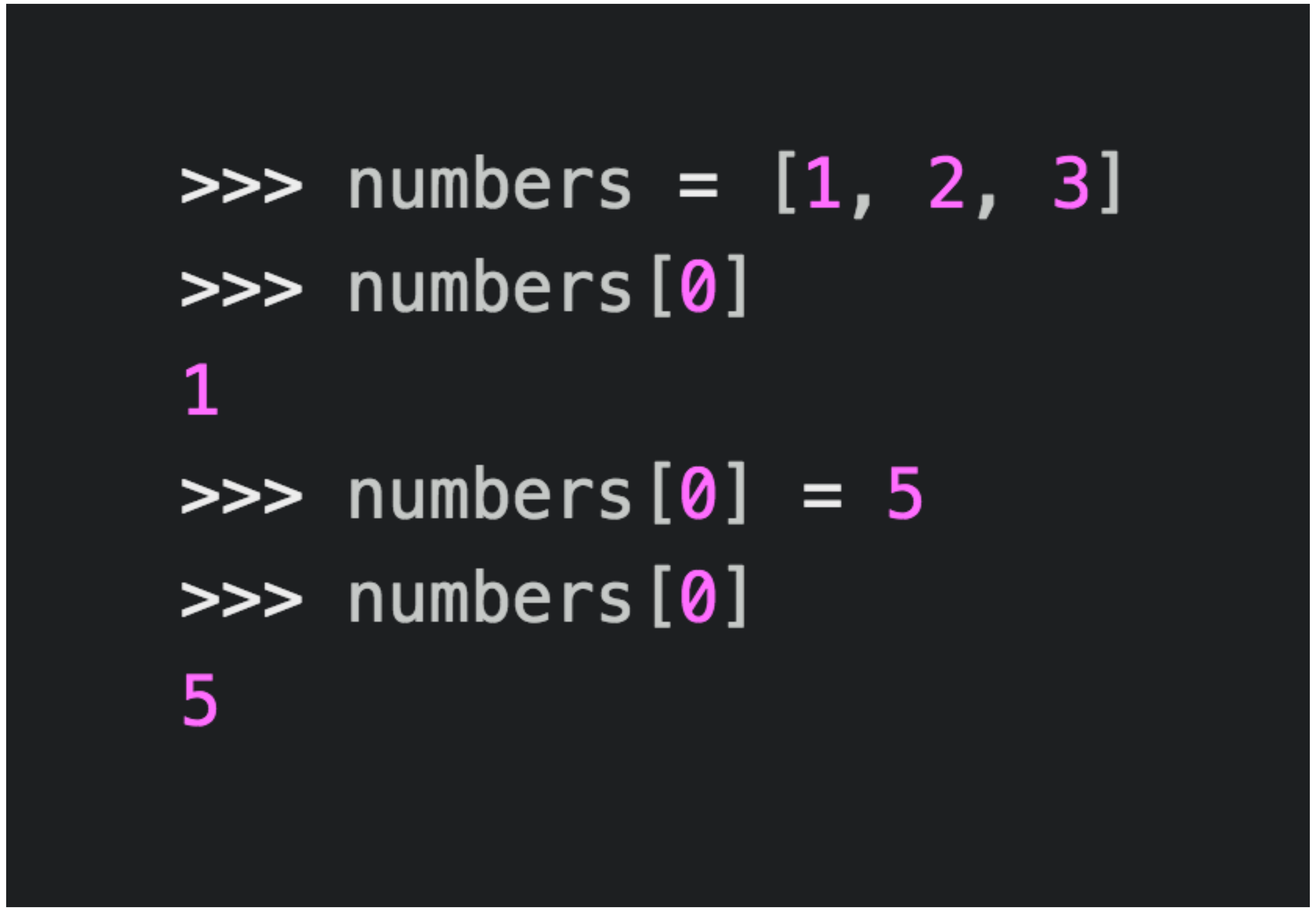
Lists



```
[1, 2, 3]
```

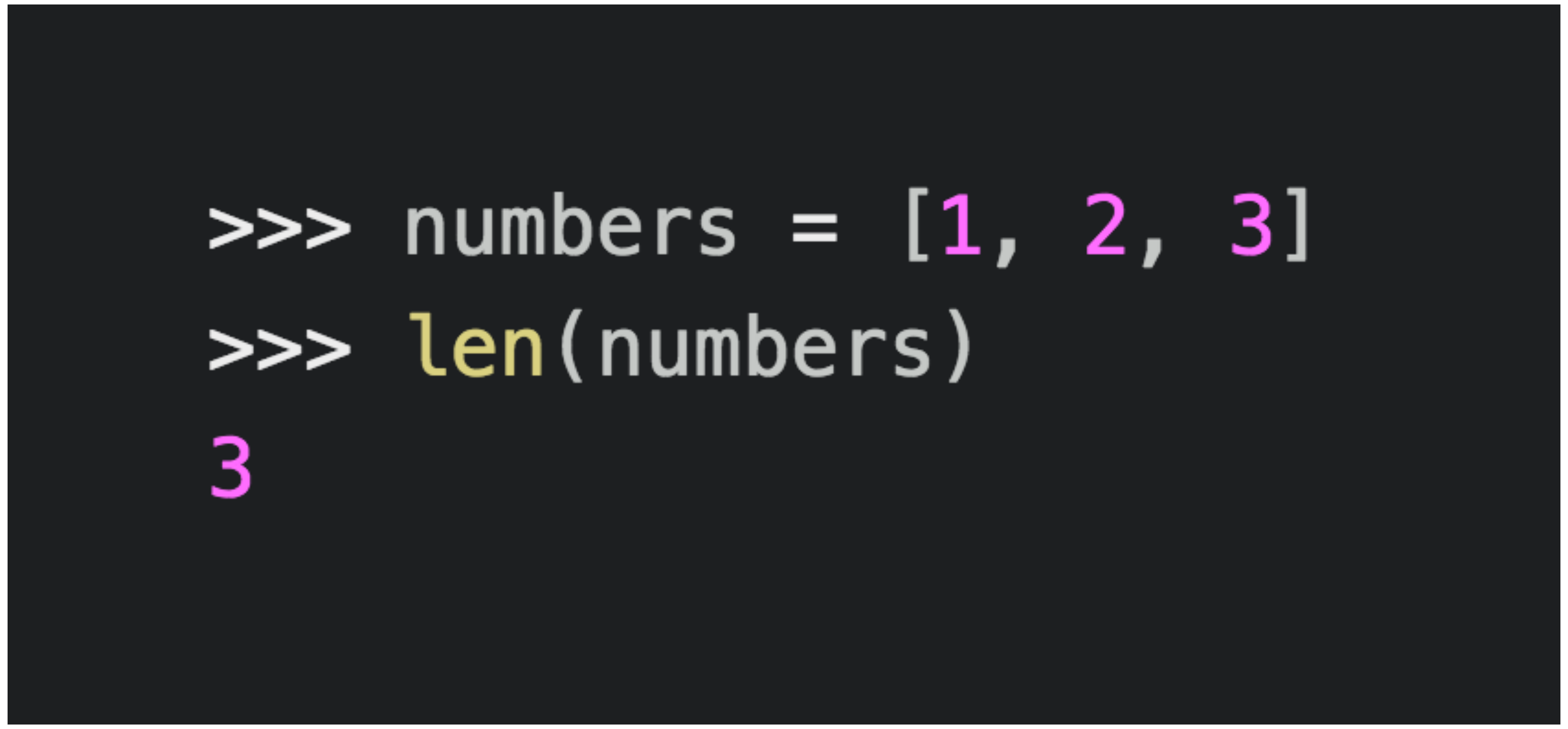
Lists are like tuples but are mutable

Indexing and mutability



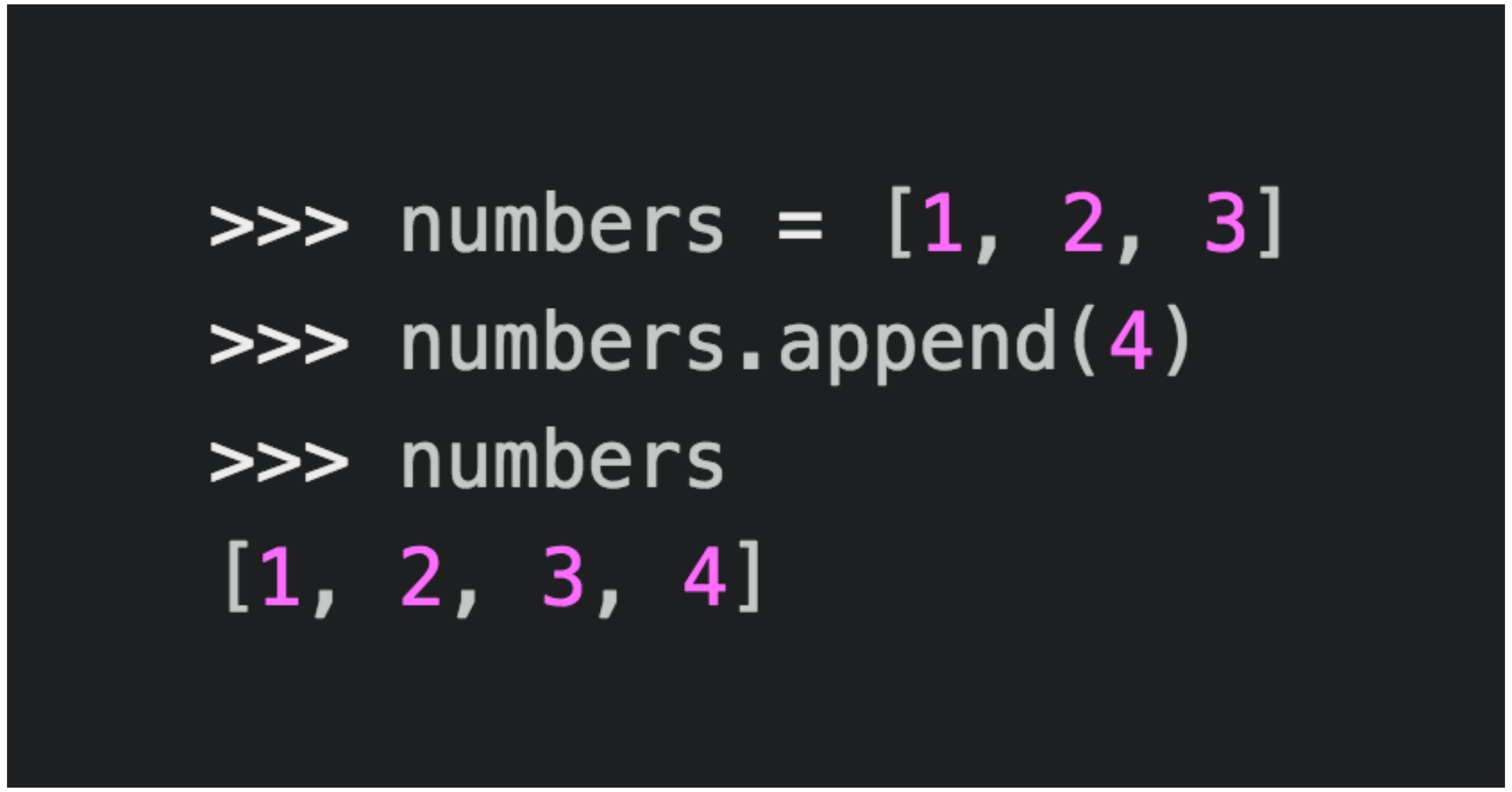
```
>>> numbers = [1, 2, 3]
>>> numbers[0]
1
>>> numbers[0] = 5
>>> numbers[0]
5
```

len() function



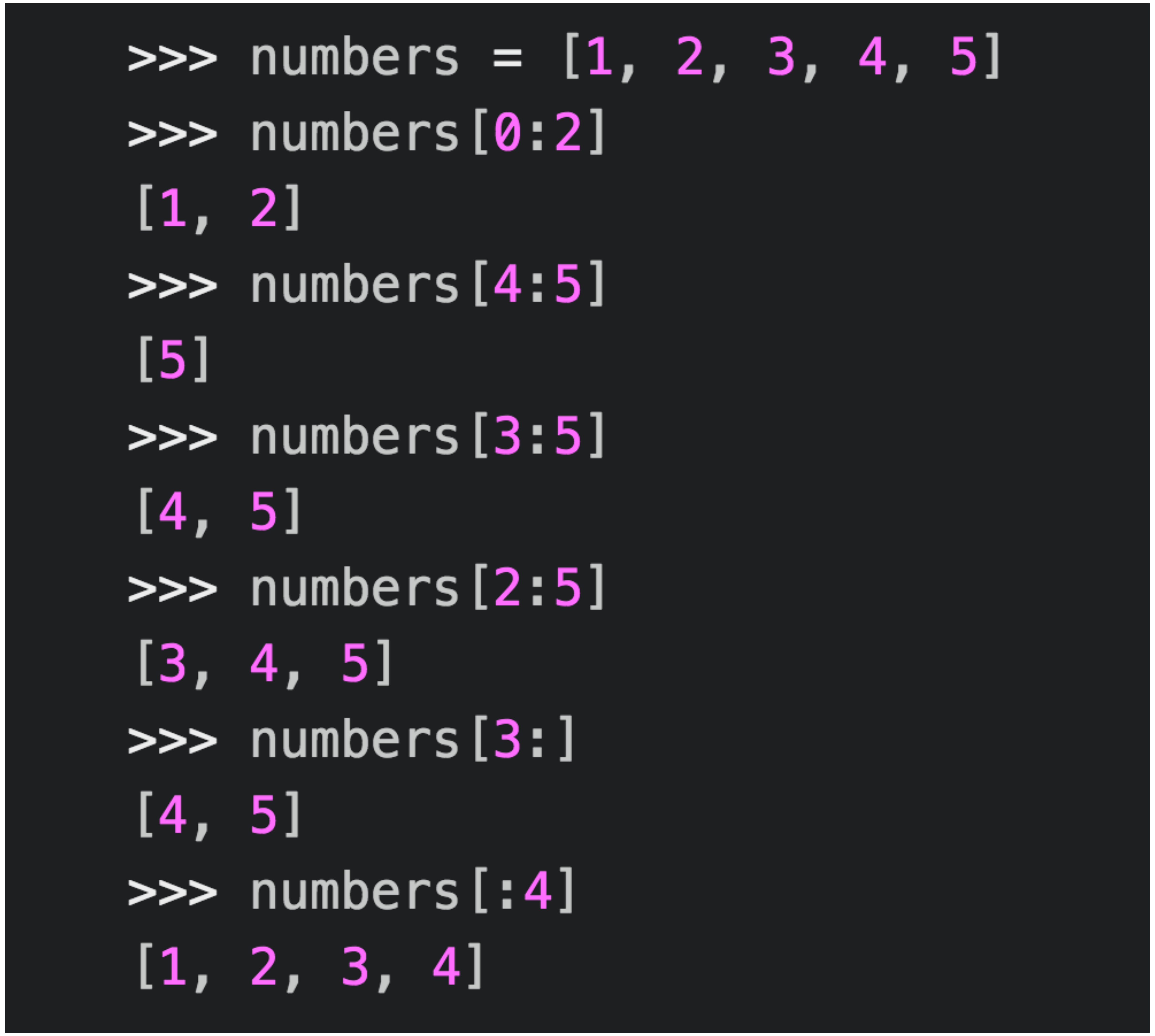
```
>>> numbers = [1, 2, 3]
>>> len(numbers)
3
```

append method



```
>>> numbers = [1, 2, 3]
>>> numbers.append(4)
>>> numbers
[1, 2, 3, 4]
```

List Slicing

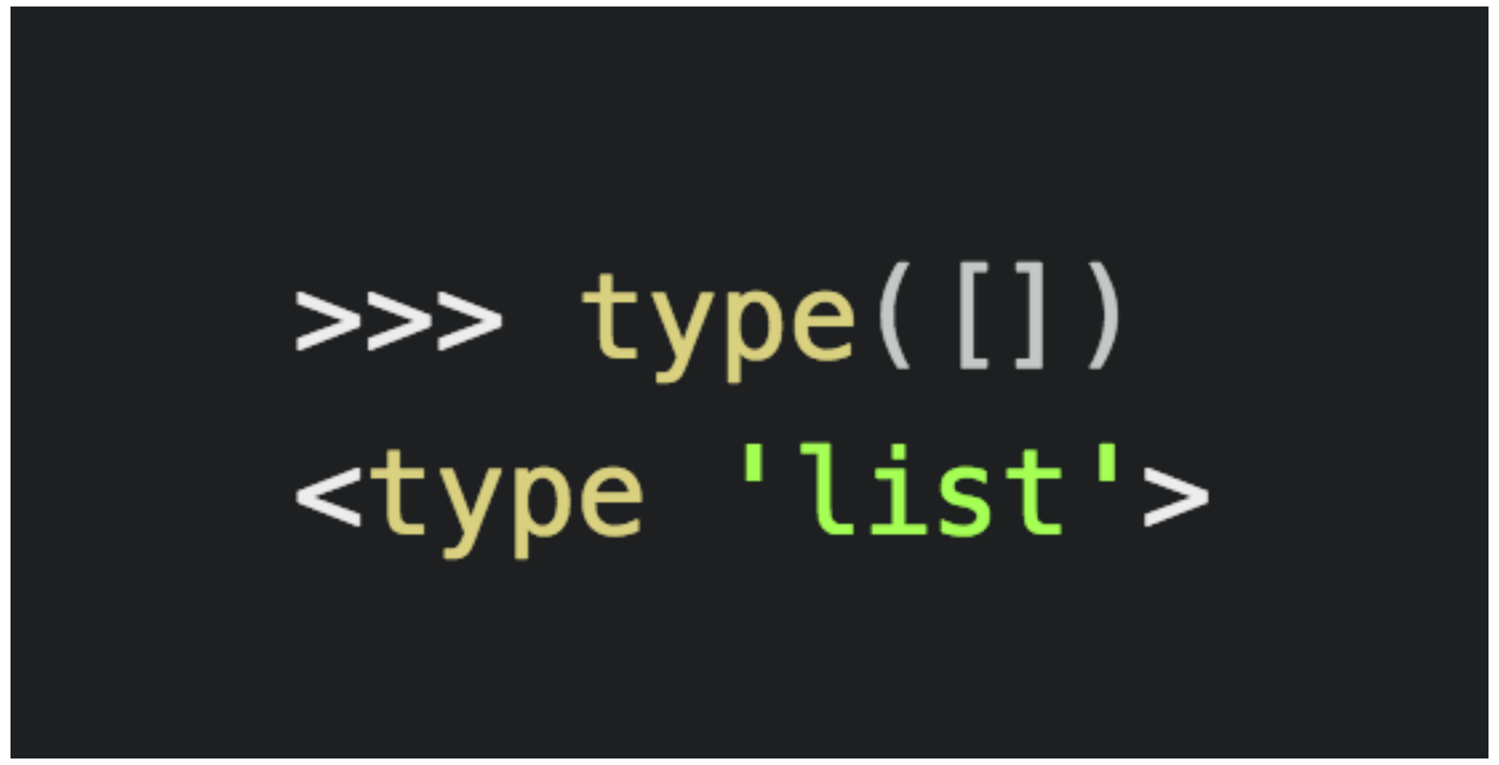


```
>>> numbers = [1, 2, 3, 4, 5]
>>> numbers[0:2]
[1, 2]
>>> numbers[4:5]
[5]
>>> numbers[3:5]
[4, 5]
>>> numbers[2:5]
[3, 4, 5]
>>> numbers[3:]
[4, 5]
>>> numbers[:4]
[1, 2, 3, 4]
```

Insert, Extend, Pop, Sort, More

- lst.insert(3, item) - inserts item at specified location
- lst.extend([3, 4, 5]) - appends all items in given list to this list
- removed_item = lst.pop() - removes an item from the end of the list
- list.index(item) - returns the index of the item in the list
- list.sort() - sorts the list
- list.copy() - returns a new copy of the list

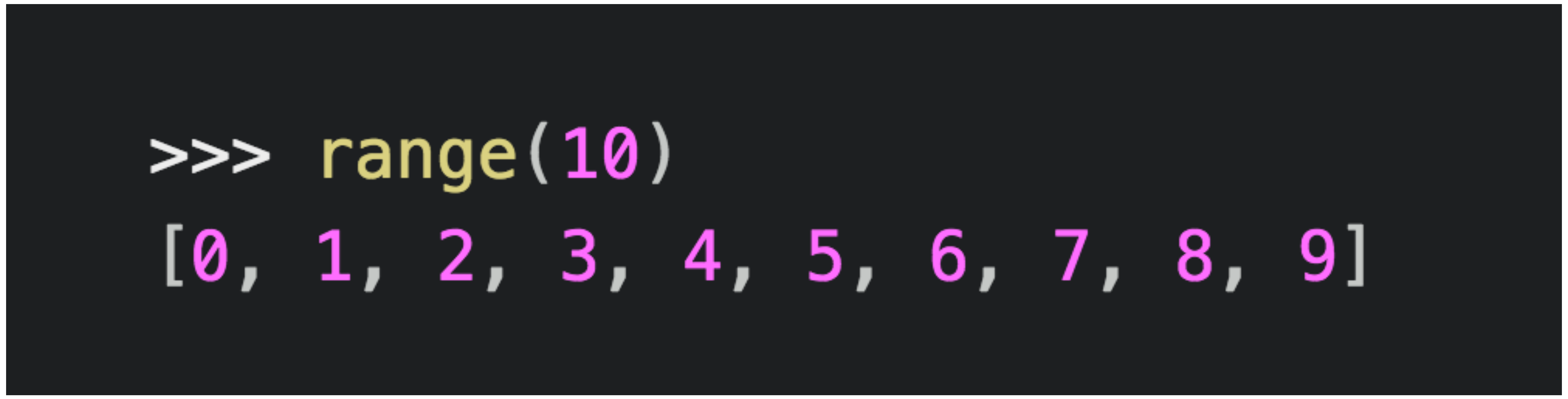
Type of a List



```
>>> type([])
<type 'list'>
```

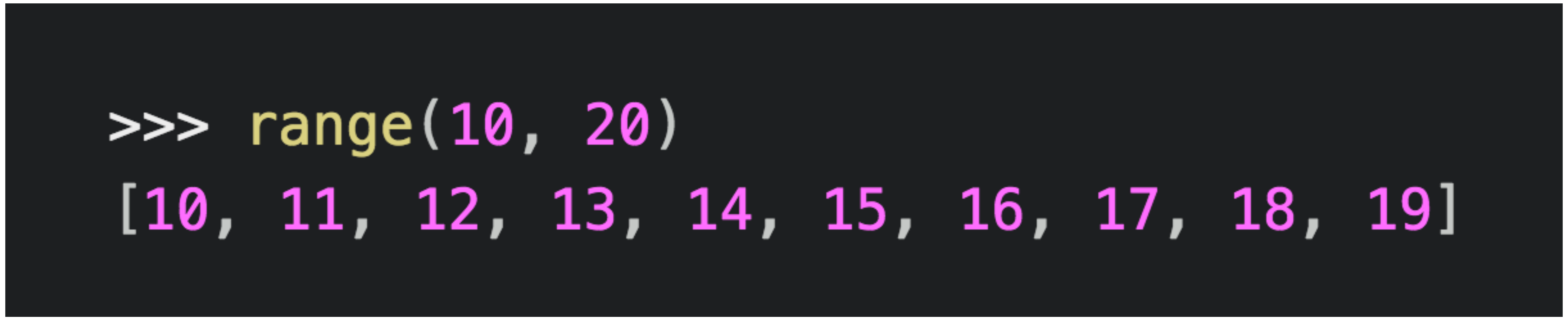
Range

Range



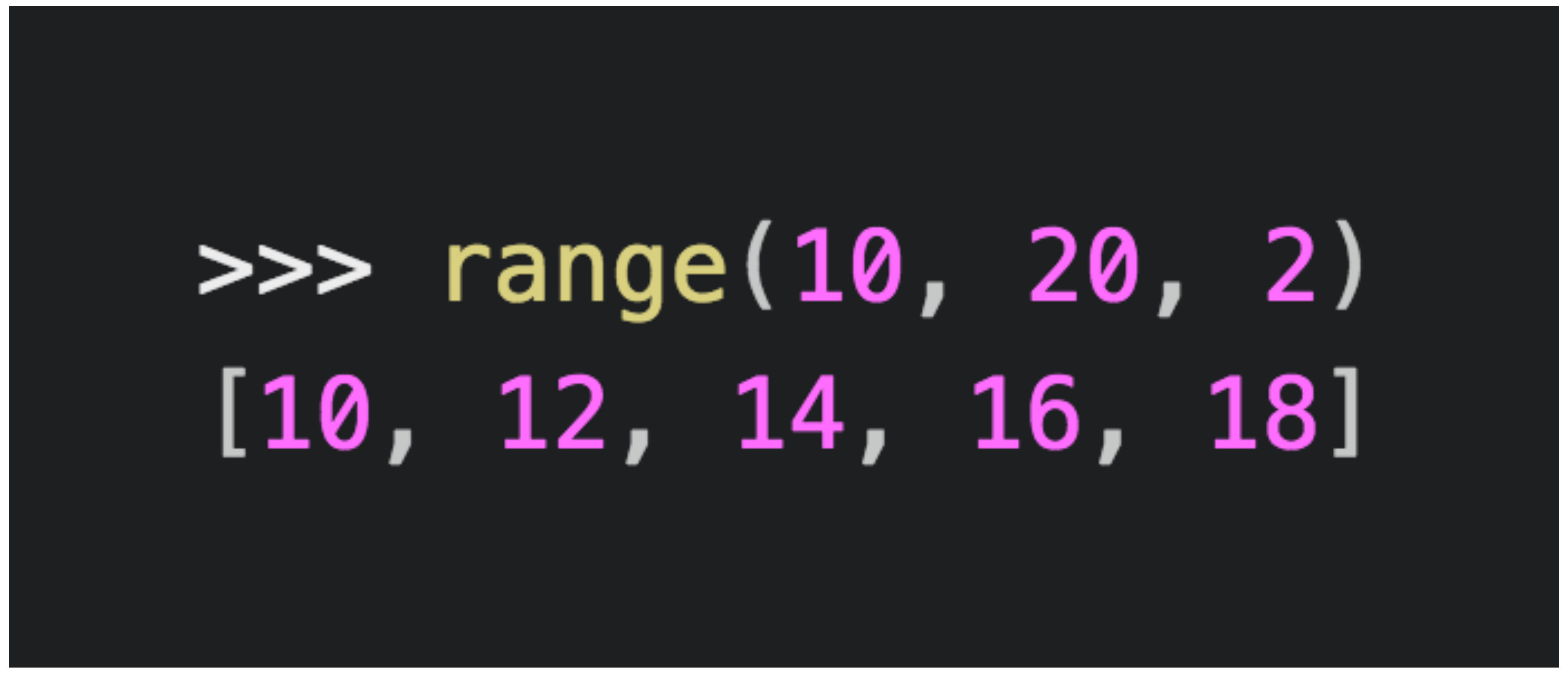
```
>>> range(10)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Range with Offset



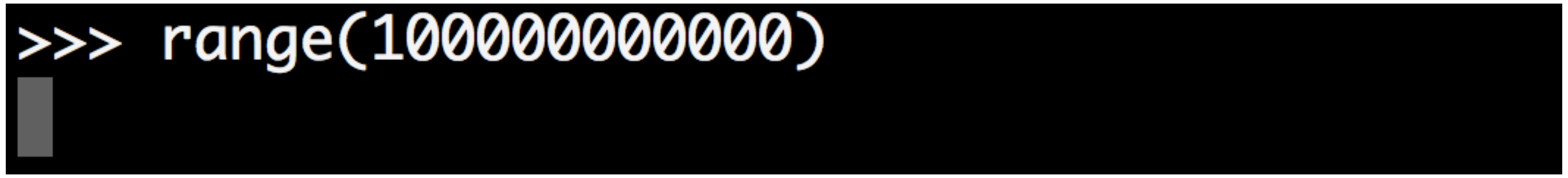
```
>>> range(10, 20)
[10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
```

Range with Skipping



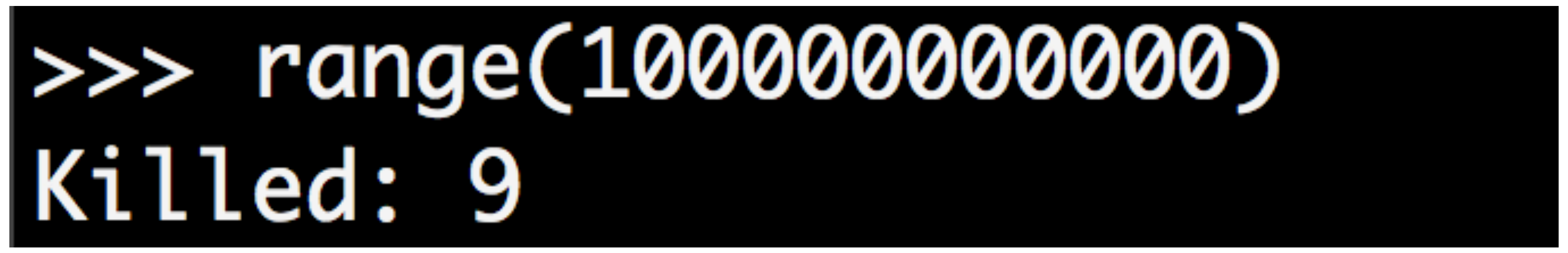
```
>>> range(10, 20, 2)
[10, 12, 14, 16, 18]
```

Big Range?



```
>>> range(100000000000)
```

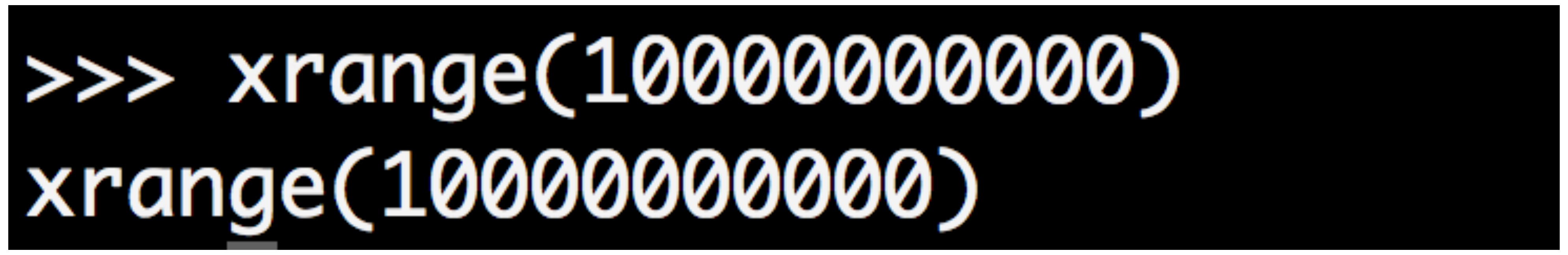
Result after waiting for a minute:



```
>>> range(100000000000)
Killed: 9
```

A range creates a list. If the list is large,
Python will need to allocate a lot of memory and also
loop many times to create all those numbers.

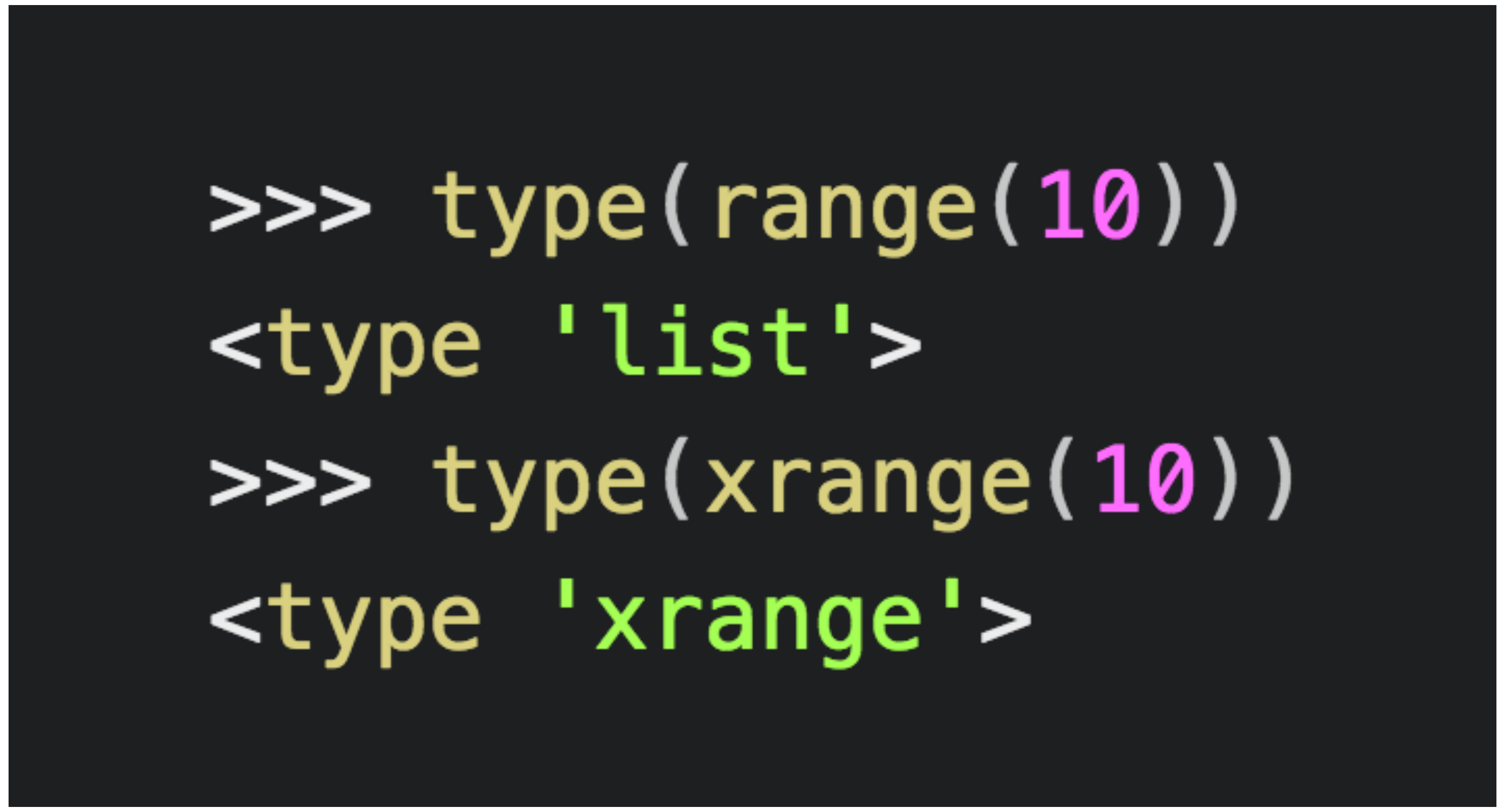
X Range



```
>>> xrange(10000000000)
xrange(10000000000)
```

X range creates a lazy list, which doesn't actually store the range of numbers it represents, but still works like the same list.

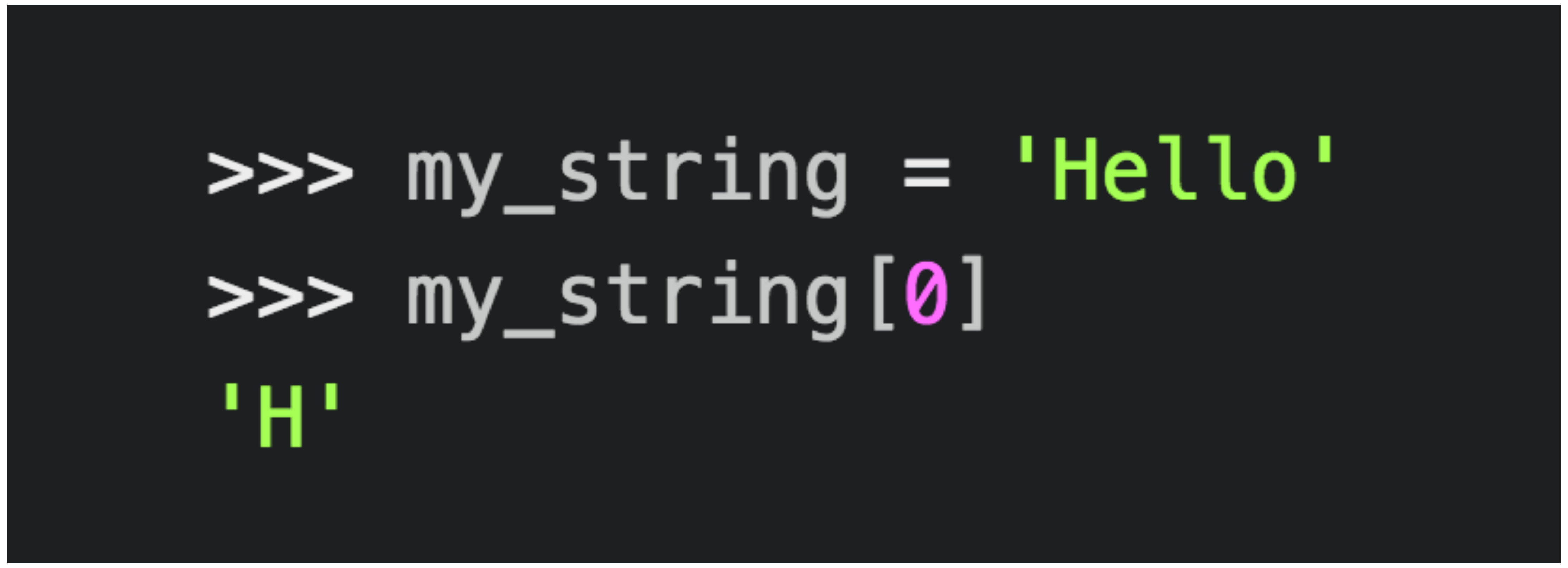
Type of Range vs X Range



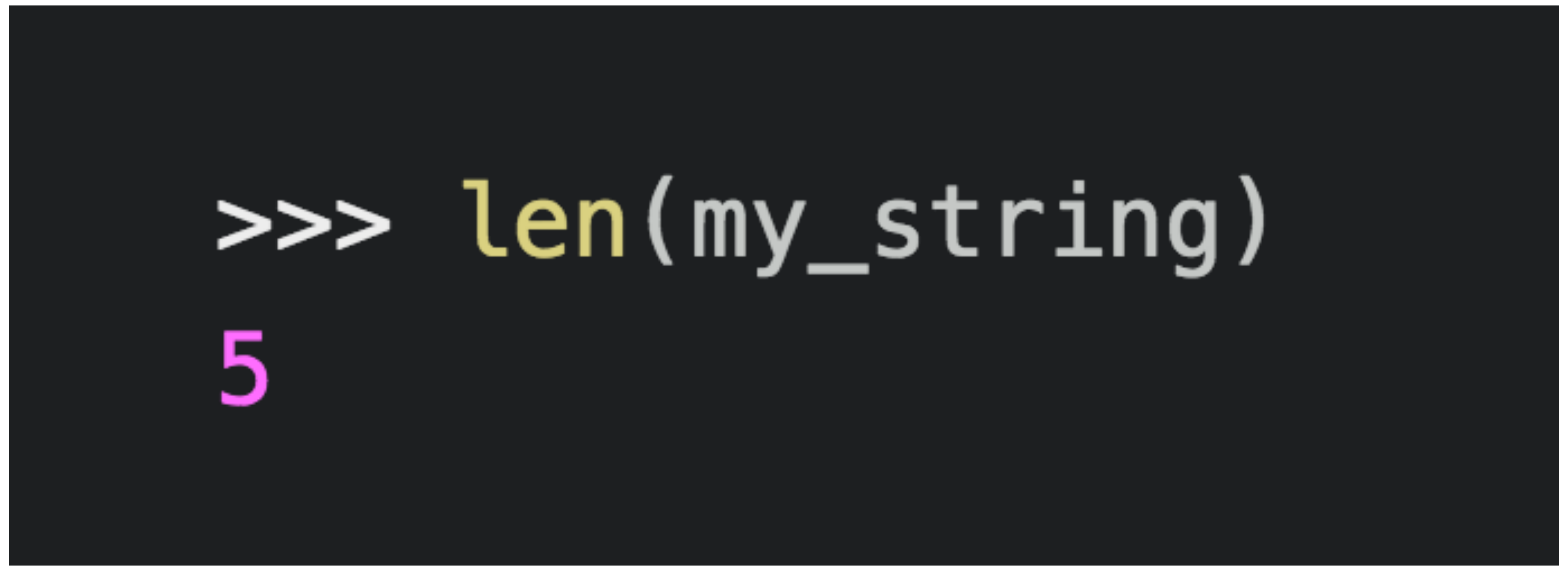
```
>>> type(range(10))
<type 'list'>
>>> type(xrange(10))
<type 'xrange'>
```

More about Strings

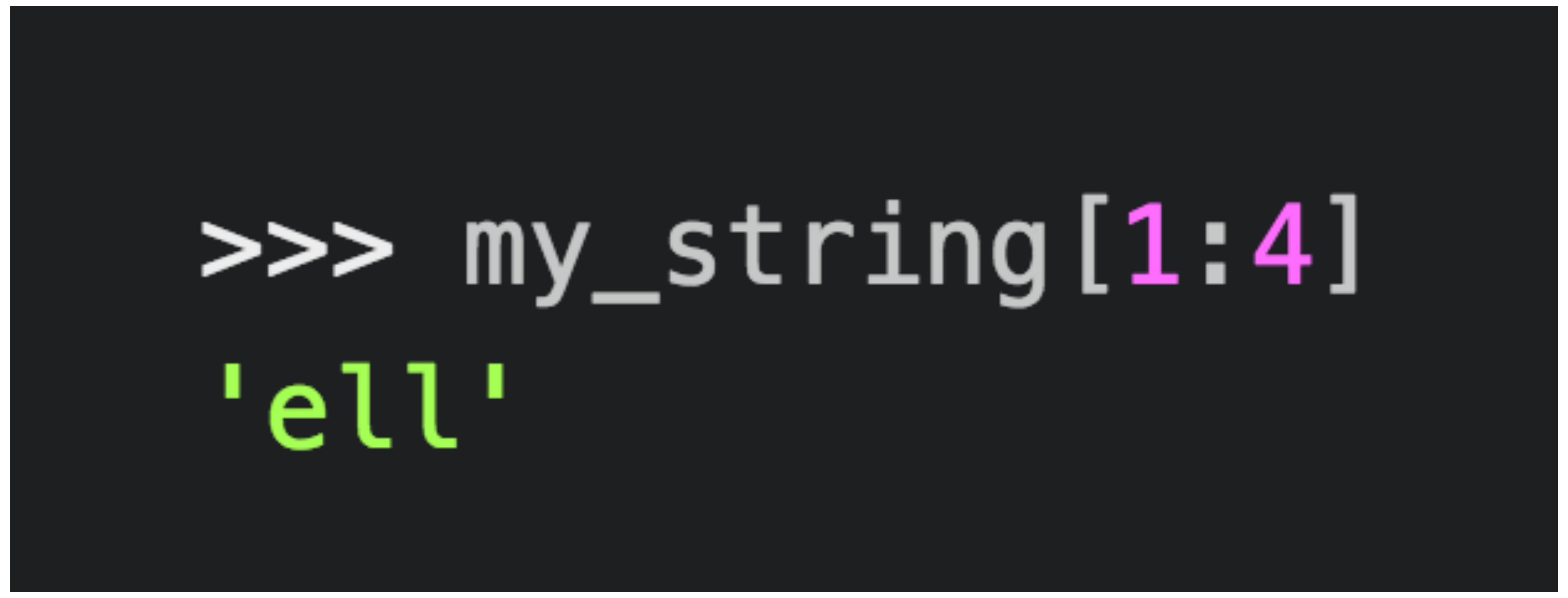
Strings are just like Lists!



```
>>> my_string = 'Hello'
>>> my_string[0]
'H'
```



```
>>> len(my_string)
5
```



```
>>> my_string[1:4]
'ell'
```

ASCII Codes

- Each ASCII character - each character you can type on your keyboard - excluding unicode characters, has a numeric value associated with it called the ASCII code
- Lower case a has an ASCII code of 97
- Upper case z has an ASCII code of 90
- `ord(‘a’)` gives you 97
- `ord(‘z’)` gives you 90

Sequences

Things that are sequences

- strings
- lists
- tuples
- xranges

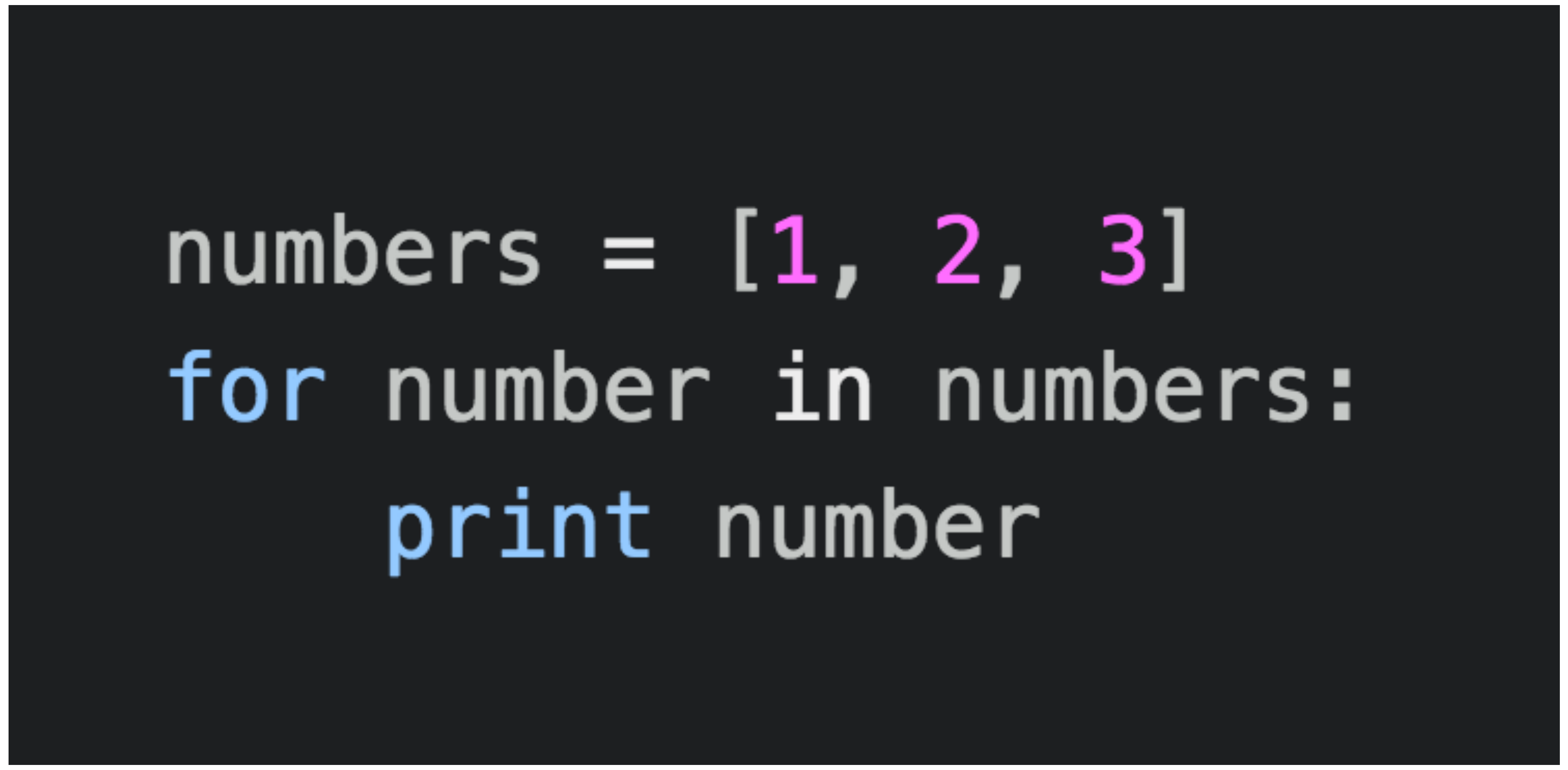
Sequence operations

- s + t - concatenate sequences s and t
- s * n - concatenate s with itself n times
- s[i] - retrieve the item at the i-th index of sequence s
- s[i:j] - retrieve a slice of the sequence s
- len(s) - get the length of the sequence s
- s.index(x) - find the index of the item x within sequence s

For loop

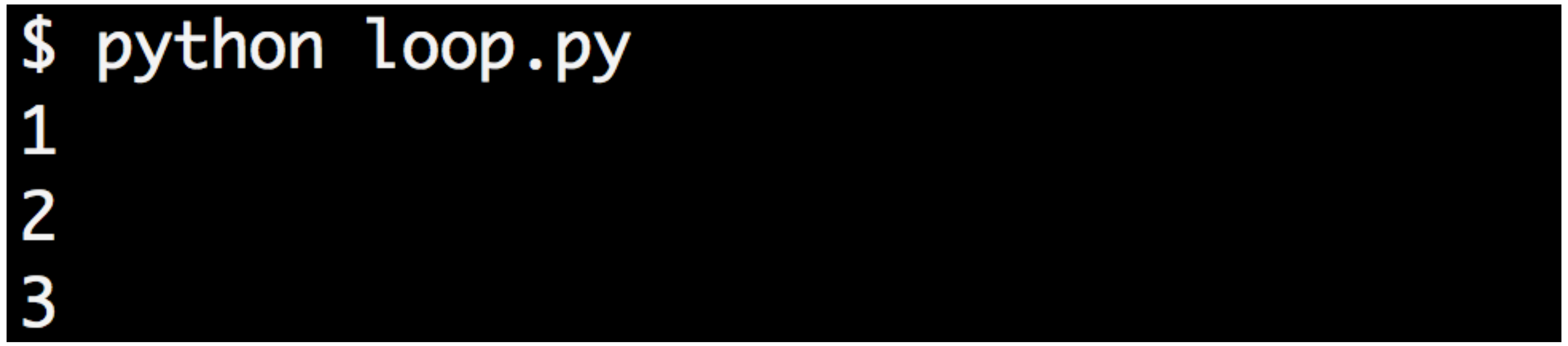
For loops loop over sequences

For loop



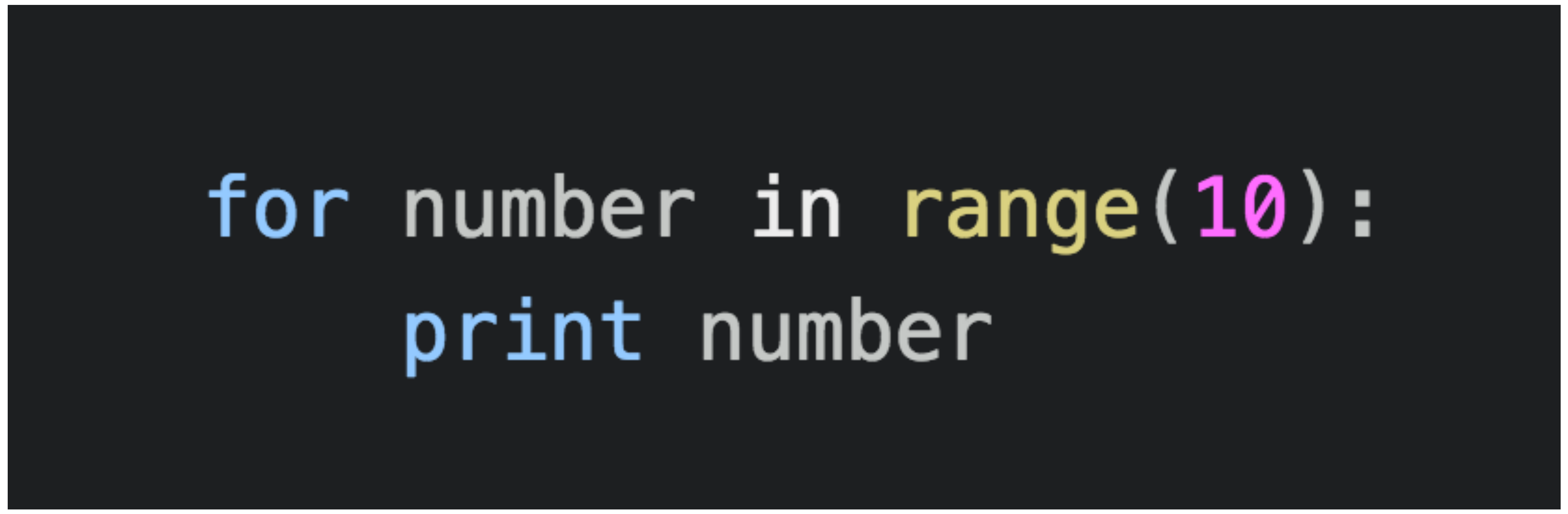
```python
numbers = [1, 2, 3]
for number in numbers:
    print number
```

Prints:



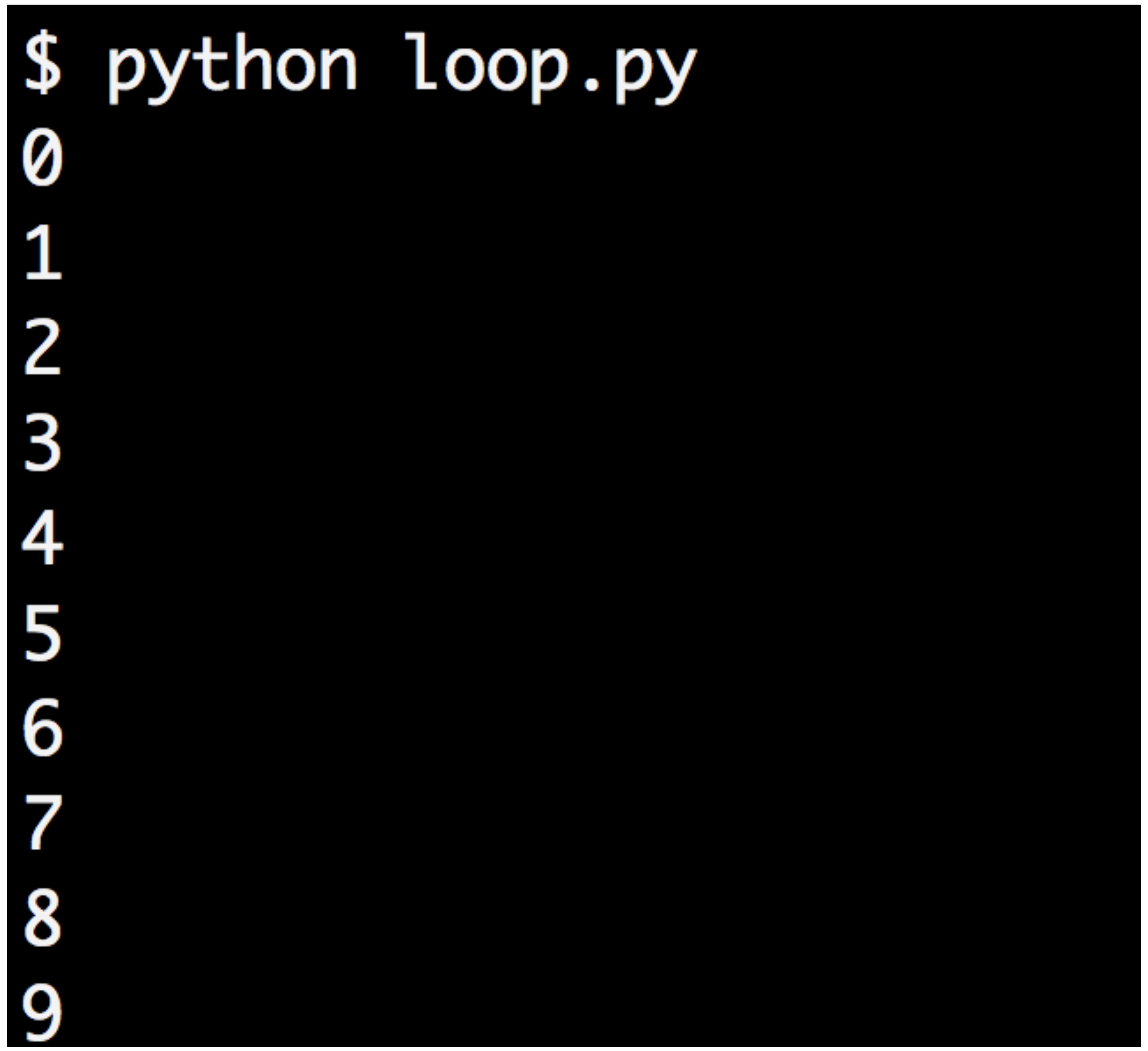
```
$ python loop.py
1
2
3
```

For loop with range or xrange



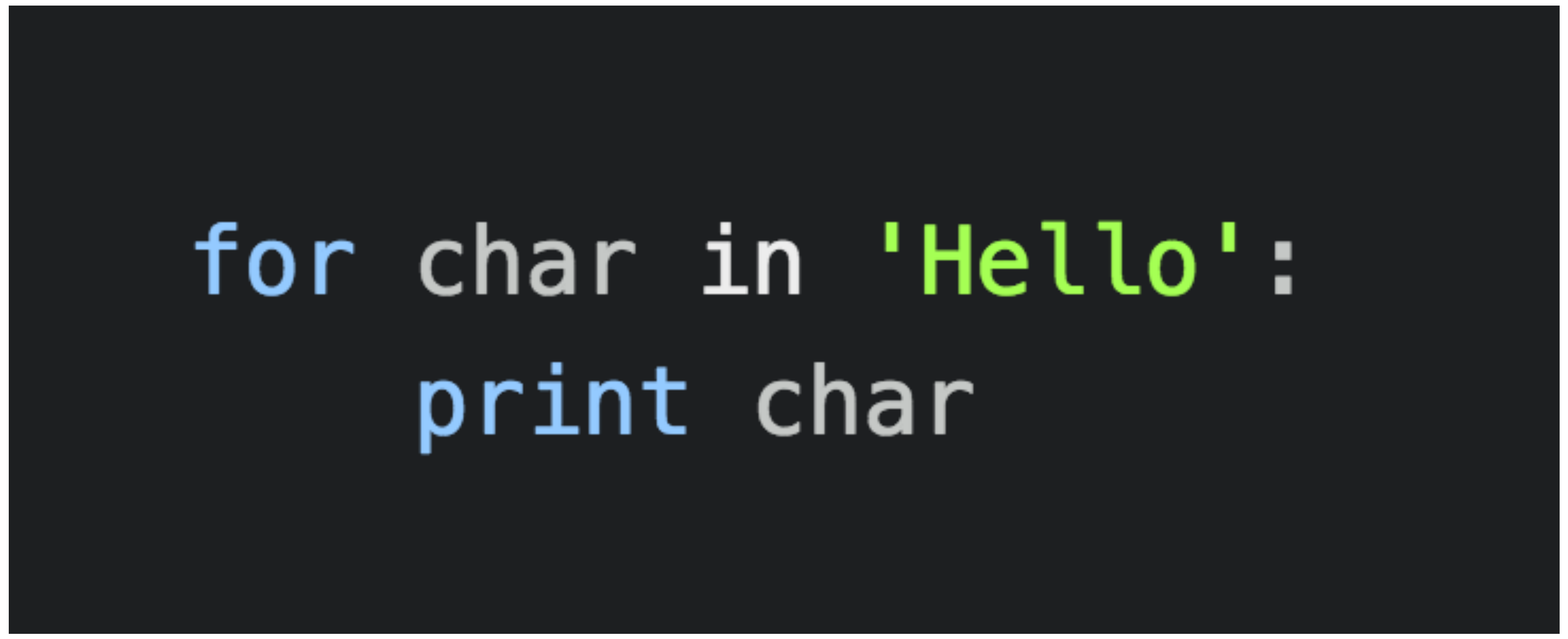
```python
for number in range(10):
    print number
```

Outputs:



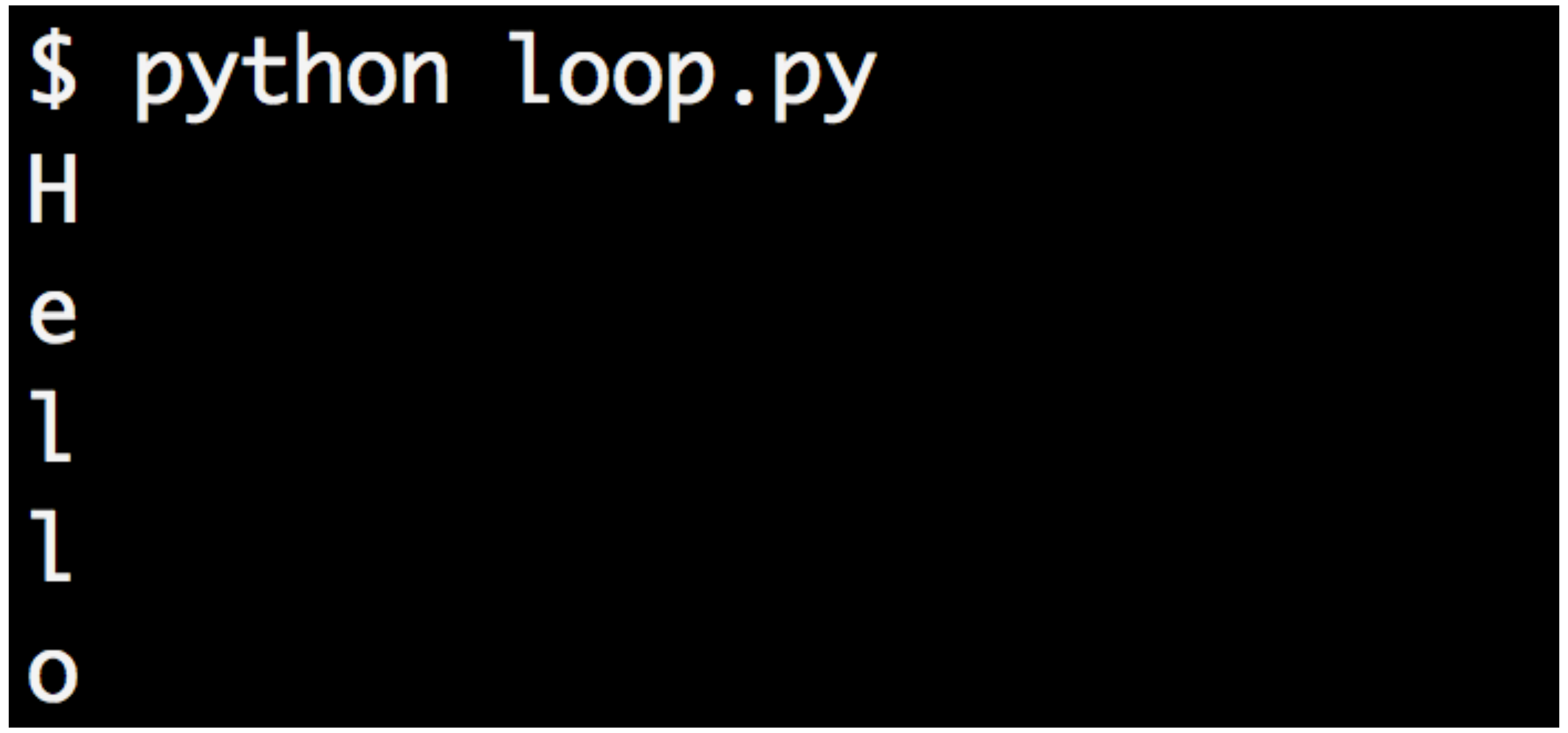
```
$ python loop.py
0
1
2
3
4
5
6
7
8
9
```

Loop through strings



```python
for char in 'Hello':
    print char
```

outputs:



```
$ python loop.py
H
e
l
l
o
```