Lab No. 1
Language Processing and Python

Preparing Development Environment for Learning NLP

- Install Python
- After installation go to CMD and install following Python libraries using pip command.
    - NumPy
    - Pandas
    - Matplotlib
    - Seaborn
    - Scikit-learn or SKLearn
    - Jupyter
    - NLTK
    - Download all NLTK resources

Basics of Python

- Fastest growing language
  - Number of developers
  - Number of Libraries
  - Number of companies
  - Area where it is used for implementation
    - Image Classification
    - NLP
    - Text Classification
    - Video Classification
    - Spatial data mining
- General purpose language
  - ML
  - Software development
  - Web development
  - GUI

Basics of Python

- Interpreted language
- High level language
- It supports procedural and Object Oriented programming
- Why it is famous?
  - Not new
    - Java 1995
    - Python 1989
  - Java was used for enterprise application and it was so popular
  - Java time-consuming in writing code
  - Scientists need easy language for data analysis and ML
- Several companies use python now:
  - Google, Yahoo, YouTube, Dropbox, NASA

Basics of Python

- **Why named as python?**
- Author of Python (Guido van Rossum) was good fan of British comedy movie

Basics of Python

- Very easy, can be coded by non programmers easily
- Python helps those who don’t know how to code
- Even great for children to learn
- Different versions
  - Python 1.0 (1994)
  - Python 2.0 (2000)
  - Python 3.0 (2008)
- No backward compatibility with previous version
- That’s why 2.0 still running in market and will have a support till 2020

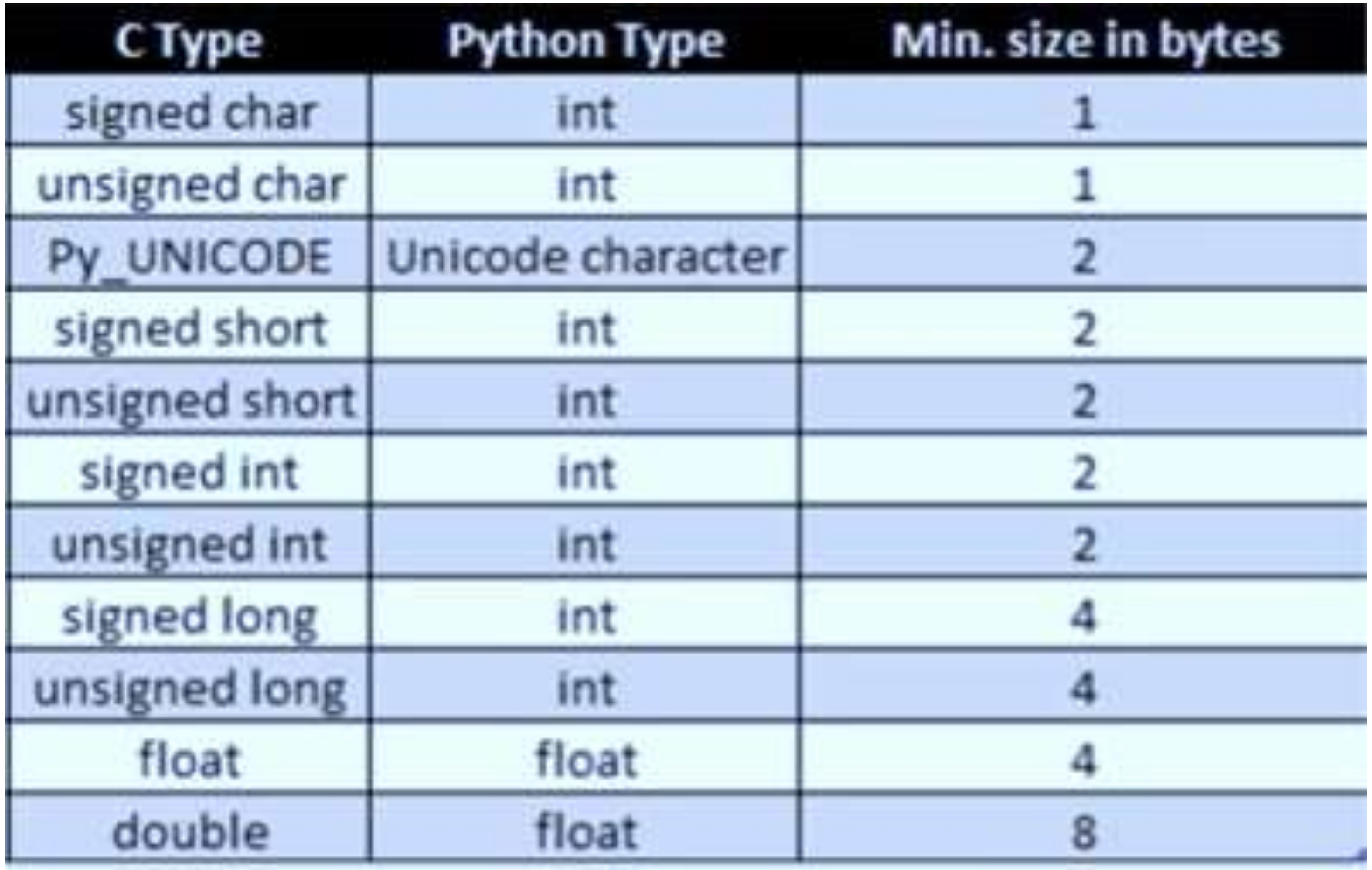
Basics of Python- Installation

- Need python interpreter to run the python code
- Do we have python interpreter in windows?
  - No
- To install latest version (Python 3.7.2)
  - https://www.python.org/downloads/
- To some thing big, you also need IDE for python
  - PyCharm (developed by Jet Brains: who developed Intellij)
- Other Python IDEs include:
  - Spyder, Eclipse + PyDev, IDLE, Atom, Jupyter Notebook
  - Jupyter is not IDE. Jupyter allows for programming in over 40 languages, including Python

Basics of Python- Installation

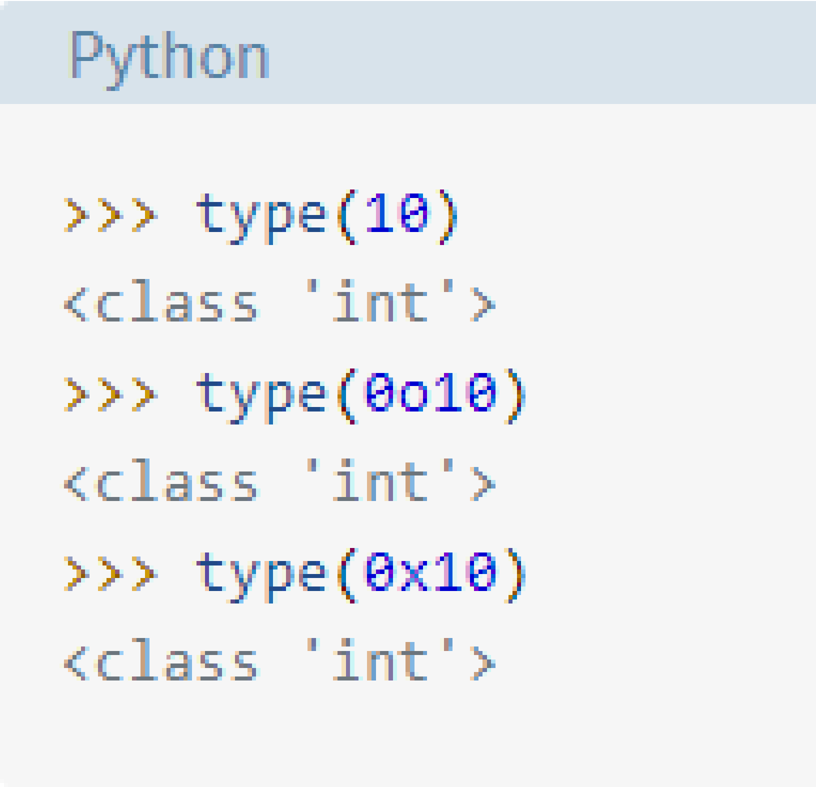
- Open cmd and type python
- If it is not working then set following paths in system variables
- MyPC --> Properties --> Advanced System Settings --> System Variables --> Path --> Edit (Add here new paths)
- Path1 = C:\Users\YourUser\AppData\Local\Programs\Python\Python35-32
- Path2 = C:\Users\YourUser\AppData\Local\Programs\Python\Python35-32\Scripts
- Restart CMD and type python
- It should work now
- Also check the pip --version

Basics of Python- Basic Datatypes



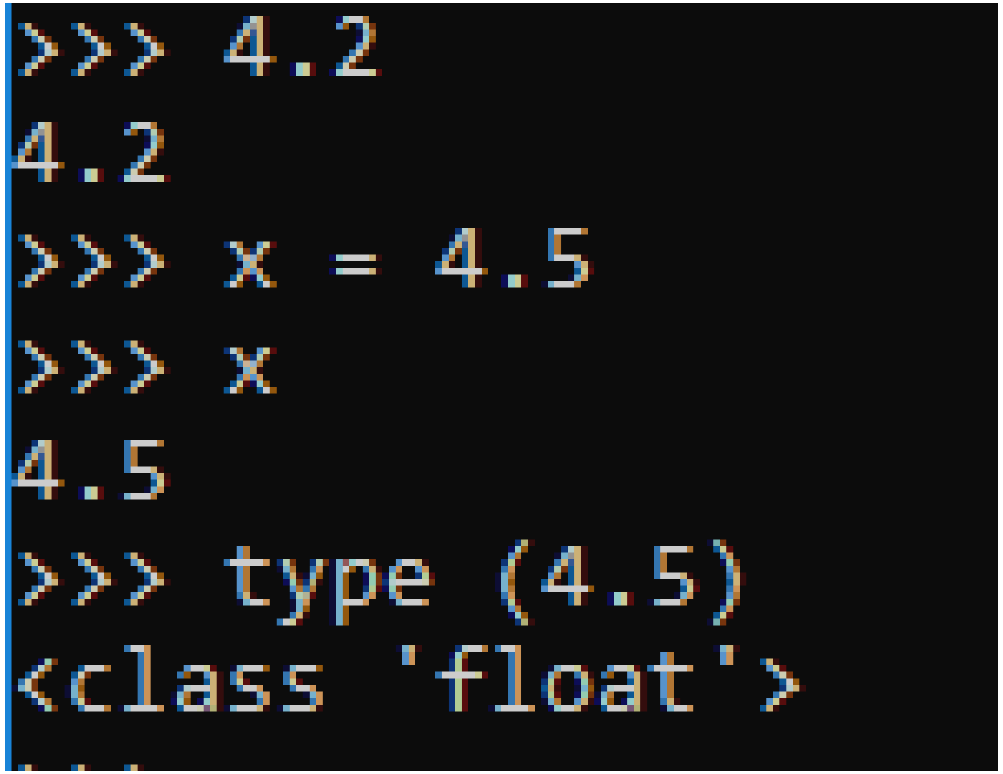
C Type	Python Type	Min. size in bytes
signed char	int	1
unsigned char	int	1
Py_UNICODE	Unicode character	2
signed short	int	2
unsigned short	int	2
signed int	int	2
unsigned int	int	2
signed long	int	4
unsigned long	int	4
float	float	4
double	float	8

Basics of Python- Integers & Floats



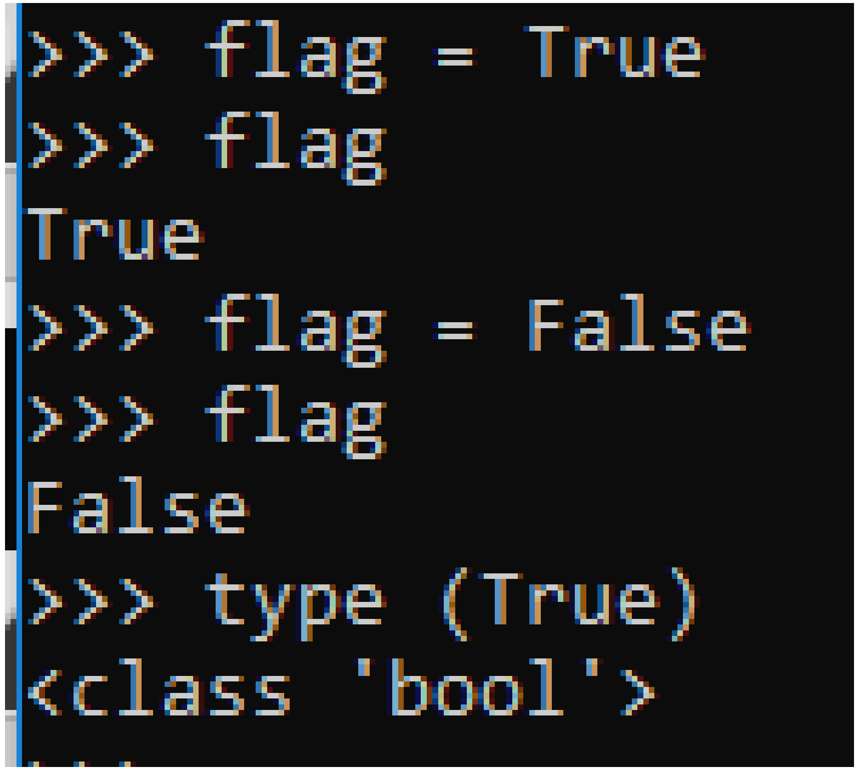
Python

```
>>> type(10)
<class 'int'>
>>> type(0o10)
<class 'int'>
>>> type(0x10)
<class 'int'>
```

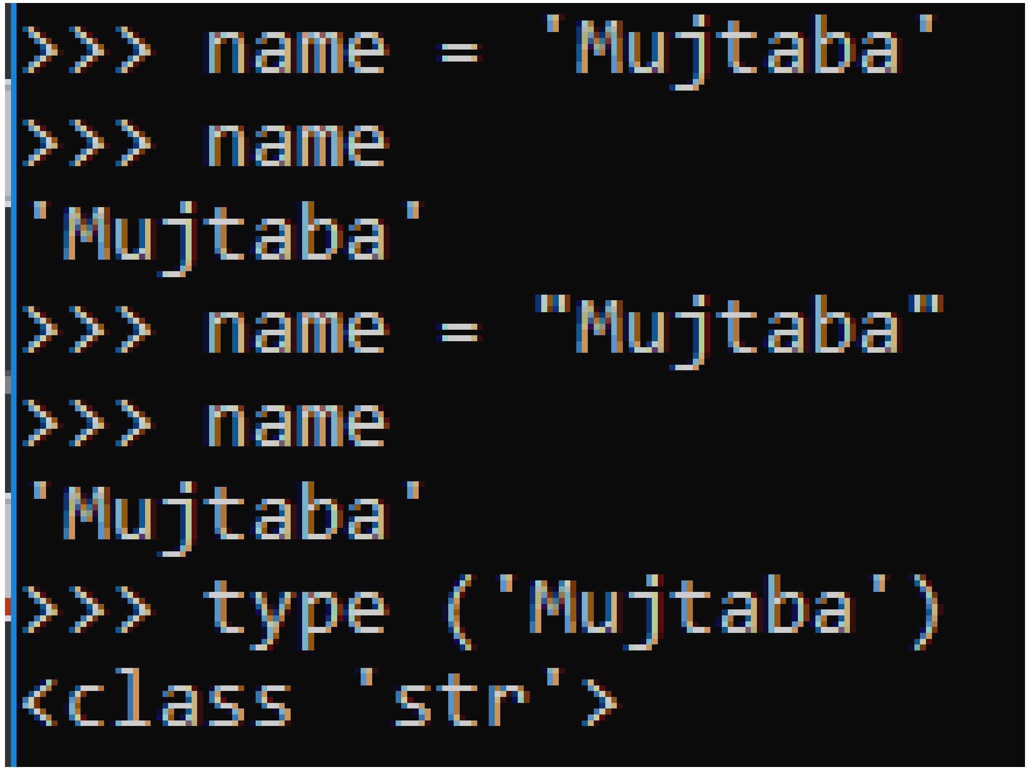


```
>>> 4.2
4.2
>>> x = 4.5
>>> x
4.5
>>> type (4.5)
<class 'float'>
```

Basics of Python- Boolean & String

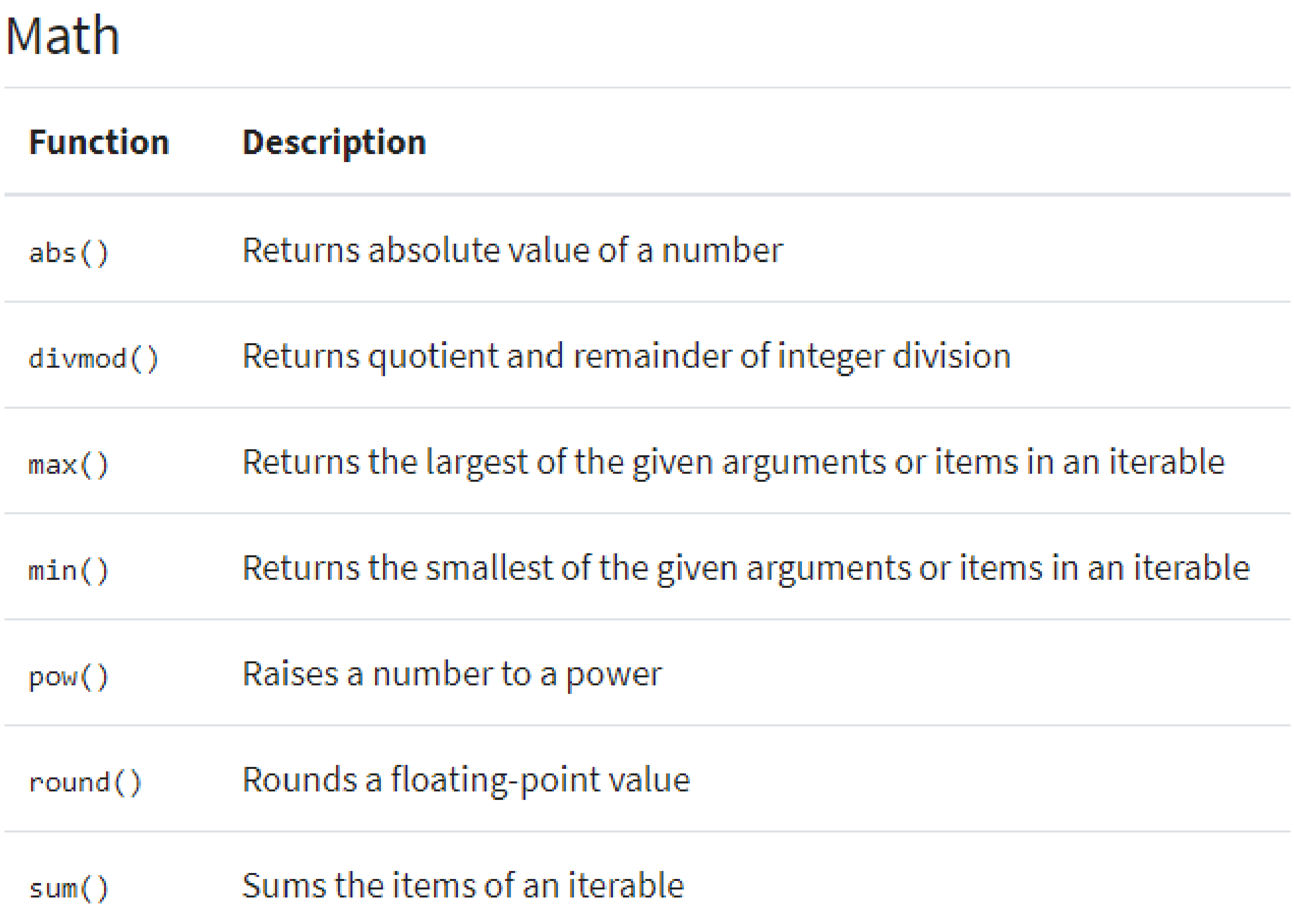


```
>>> flag = True
>>> flag
True
>>> flag = False
>>> flag
False
>>> type (True)
<class 'bool'>
```



```
>>> name = 'Mujtaba'
>>> name
'Mujtaba'
>>> name = "Mujtaba"
>>> name
'Mujtaba'
>>> type ('Mujtaba')
<class 'str'>
```

Basics of Python- Built-in Functions



Math

Function	Description
`abs()`	Returns absolute value of a number
`divmod()`	Returns quotient and remainder of integer division
`max()`	Returns the largest of the given arguments or items in an iterable
`min()`	Returns the smallest of the given arguments or items in an iterable
`pow()`	Raises a number to a power
`round()`	Rounds a floating-point value
`sum()`	Sums the items of an iterable

Basics of Python- User Input

```
x = input ("Enter first value : ")
num1 = int (x)

y = input ("Enter second value : ")
num2 = int (y)

result = num1 + num2

print (result)
```

Math Function

Import math

X = math.sqrt (25)

Print (math.floor (3.5))

Import math as m (Alias)

X = m.sqrt (25)

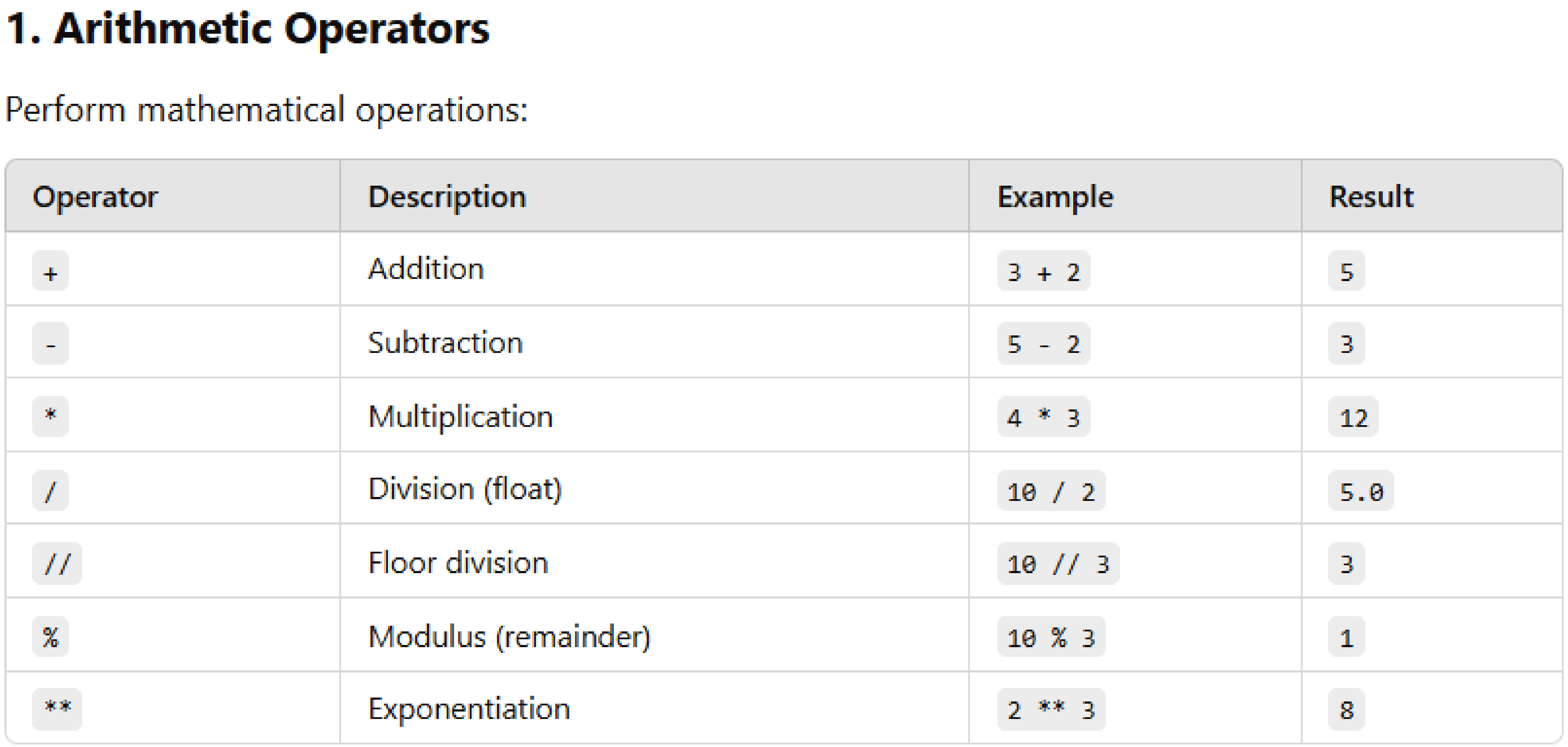
From math import sqrt, pow

Help (‘math’)

User input

```
x = input ("Enter first value : ")
num1 = int (x)
y = input ("Enter second value : ")
num2 = int (y)
result = num1 + num2
print (result)
```

Operators in Python



1. Arithmetic Operators

Perform mathematical operations:

Operator	Description	Example	Result
`+`	Addition	`3 + 2`	`5`
`-`	Subtraction	`5 - 2`	`3`
`*`	Multiplication	`4 * 3`	`12`
`/`	Division (float)	`10 / 2`	`5.0`
`//`	Floor division	`10 // 3`	`3`
`%`	Modulus (remainder)	`10 % 3`	`1`
`**`	Exponentiation	`2 ** 3`	`8`

2. Comparison (Relational) Operators

Compare two values and return a boolean result:

Operator	Description	Example	Result
`==`	Equal to	`5 == 5`	`True`
`!=`	Not equal to	`5 != 3`	`True`
`>`	Greater than	`5 > 3`	`True`
`<`	Less than	`3 < 5`	`True`
`>=`	Greater than or equal to	`5 >= 3`	`True`
`<=`	Less than or equal to	`3 <= 5`	`True`

3. Logical Operators

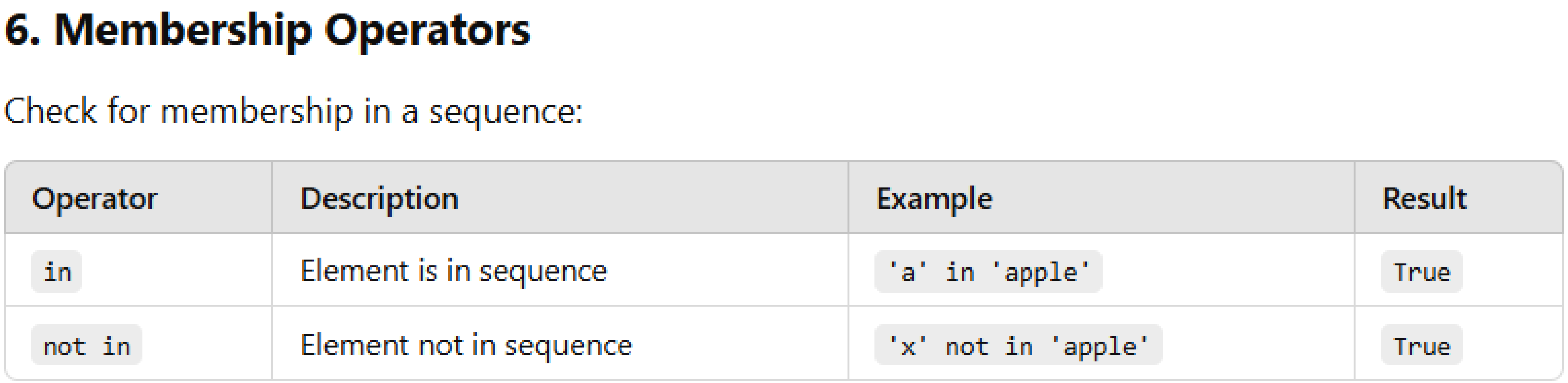
Used to combine conditional statements:

Operator	Description	Example	Result
`and`	Logical AND	`True and False`	`False`
`or`	Logical OR	`True or False`	`True`
`not`	Logical NOT	`not True`	`False`

4. Assignment Operators

Used to assign values to variables:

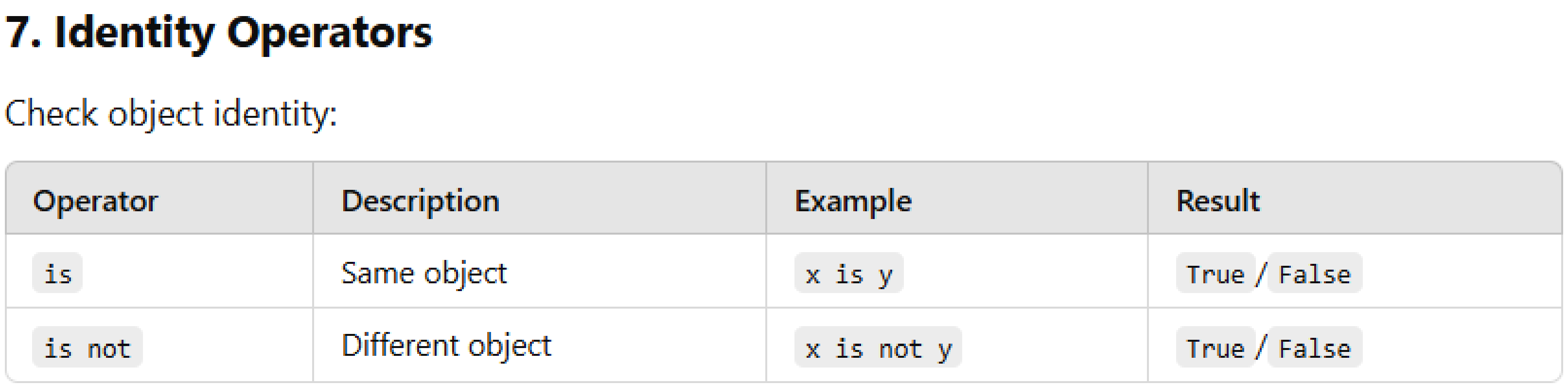
Operator	Description	Example	Equivalent To
`=`	Assign	`x = 5`	-
`+=`	Add and assign	`x += 3`	`x = x + 3`
`-=`	Subtract and assign	`x -= 2`	`x = x - 2`
`*=`	Multiply and assign	`x *= 3`	`x = x * 3`
`/=`	Divide and assign	`x /= 2`	`x = x / 2`
`//=`	Floor divide and assign	`x //= 3`	`x = x // 3`
`%=`	Modulus and assign	`x %= 3`	`x = x % 3`
`**=`	Exponent and assign	`x **= 2`	`x = x ** 2`



6. Membership Operators

Check for membership in a sequence:

Operator	Description	Example	Result
`in`	Element is in sequence	`'a' in 'apple'`	`True`
`not in`	Element not in sequence	`'x' not in 'apple'`	`True`



7. Identity Operators

Check object identity:

Operator	Description	Example	Result
`is`	Same object	`x is y`	`True` / `False`
`is not`	Different object	`x is not y`	`True` / `False`

Conditions- Simple If else

```
x = 7
r = x % 2
if r == 0:
print ("Even")
else:
print ("Odd")
```

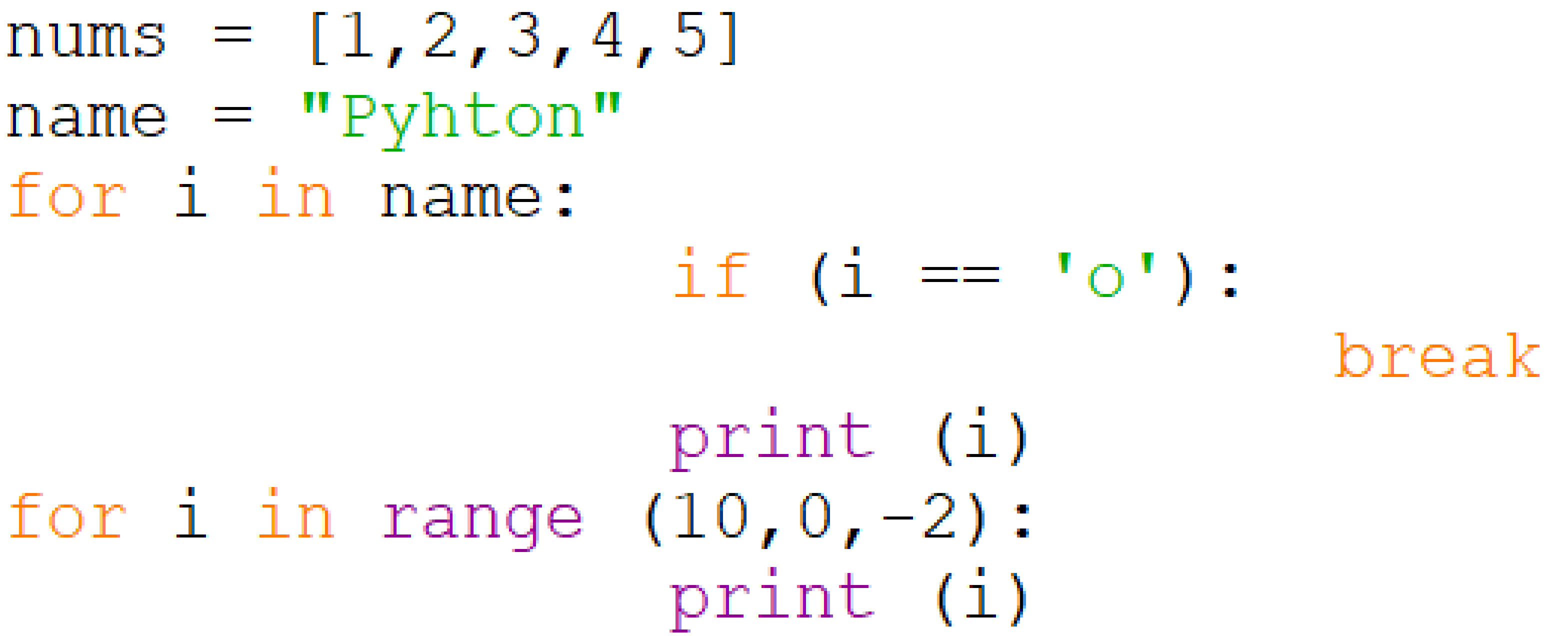
Conditions- ElIf

```
x = 1
if x == 1:
print ("One")
elif x == 2:
print ("Two")
elif x == 3:
print ("Three")
else:
print ("No match")
```

Conditions- Nested If

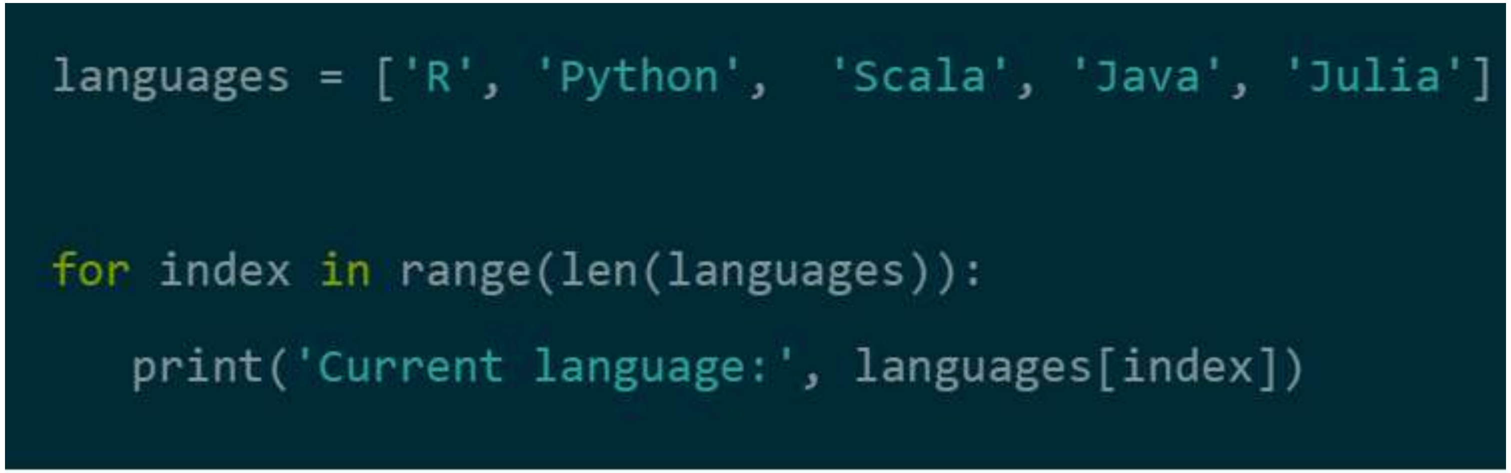
```
x = 8
r = x % 2
if r == 0:
print ("Even")
if x > 10:
print ("Great")
else:
print ("Not great")
else:
print ("Odd")
print ("Bye")
```

for Loop

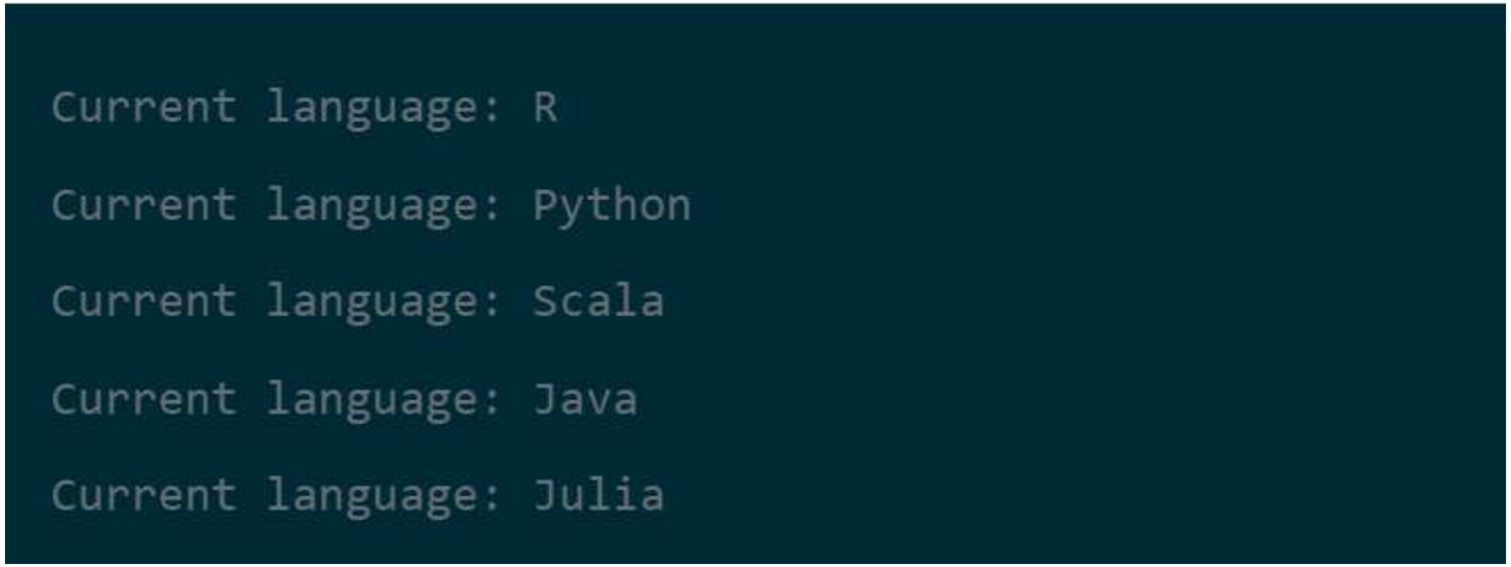


```
nums = [1,2,3,4,5]
name = "Pyhton"
for i in name:
        if (i == 'o'):
                break
        print (i)
for i in range (10,0,-2):
        print (i)
```

for Loop

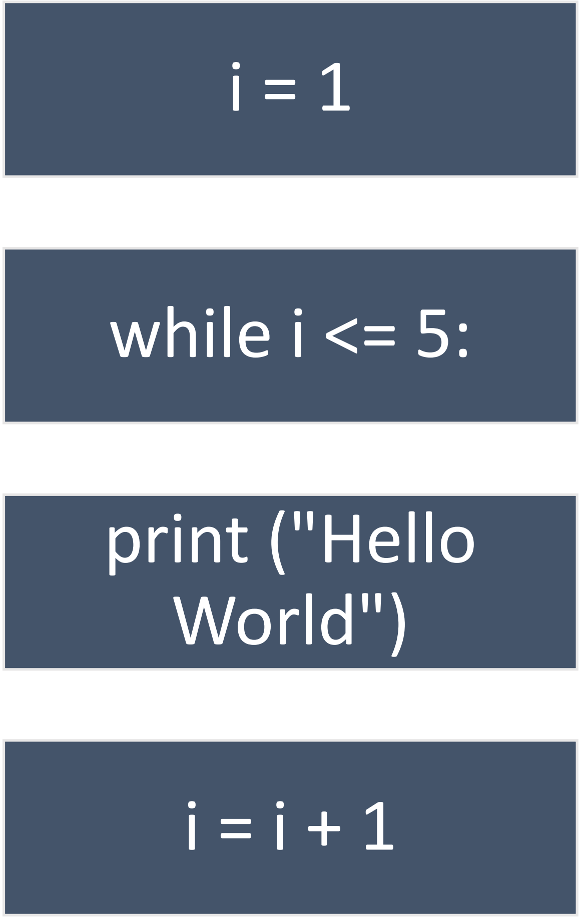


```
languages = ['R', 'Python',  'Scala', 'Java', 'Julia']


for index in range(len(languages)):
    print('Current language:', languages[index])
```



```
Current language: R
Current language: Python
Current language: Scala
Current language: Java
Current language: Julia
```

While Loop

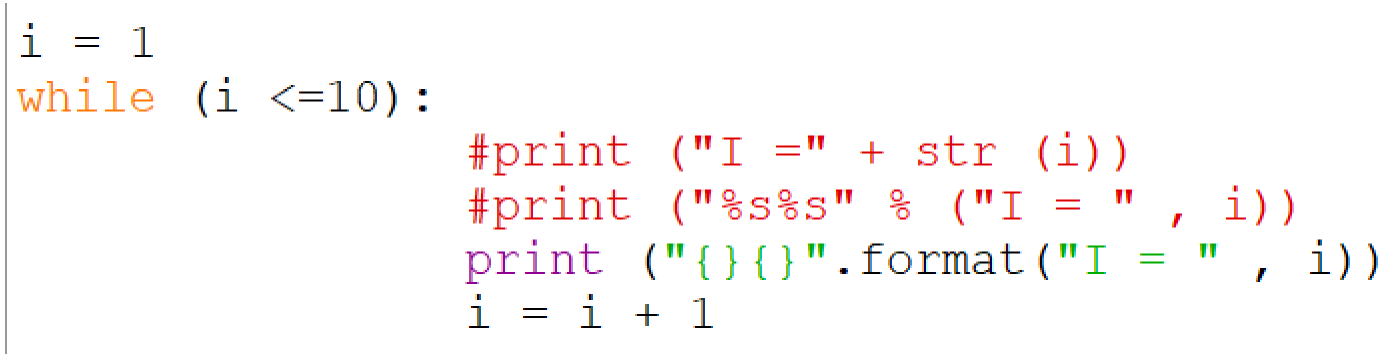


i = 1

while i <= 5:

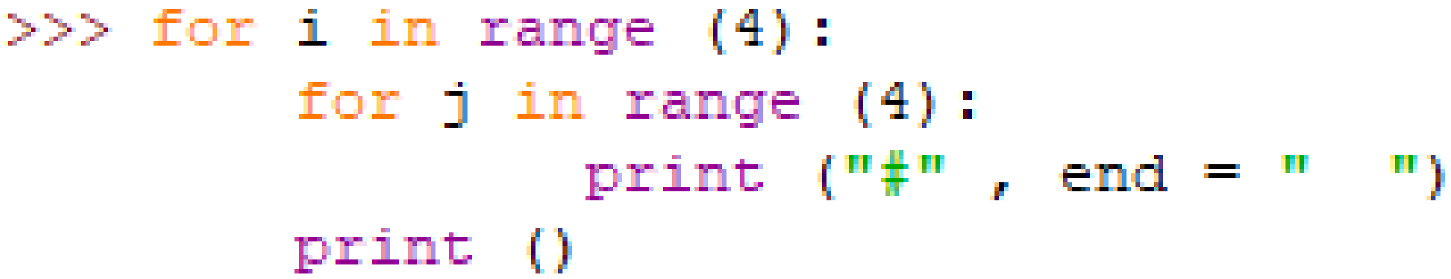
print ("Hello World")

i = i + 1

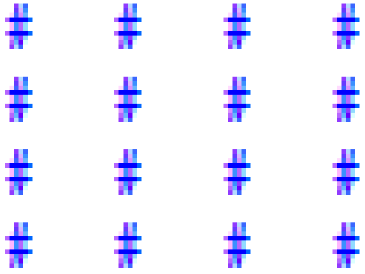


```
i = 1
while (i <=10):
            #print ("I =" + str (i))
            #print ("%s%s" % ("I = " , i))
            print ("{}{}".format("I = " , i))
            i = i + 1
```

Nested Loop



```
>>> for i in range (4):
        for j in range (4):
                print ("#" , end = "  ")
        print ()
```



```
#   #   #   #
#   #   #   #
#   #   #   #
#   #   #   #
```

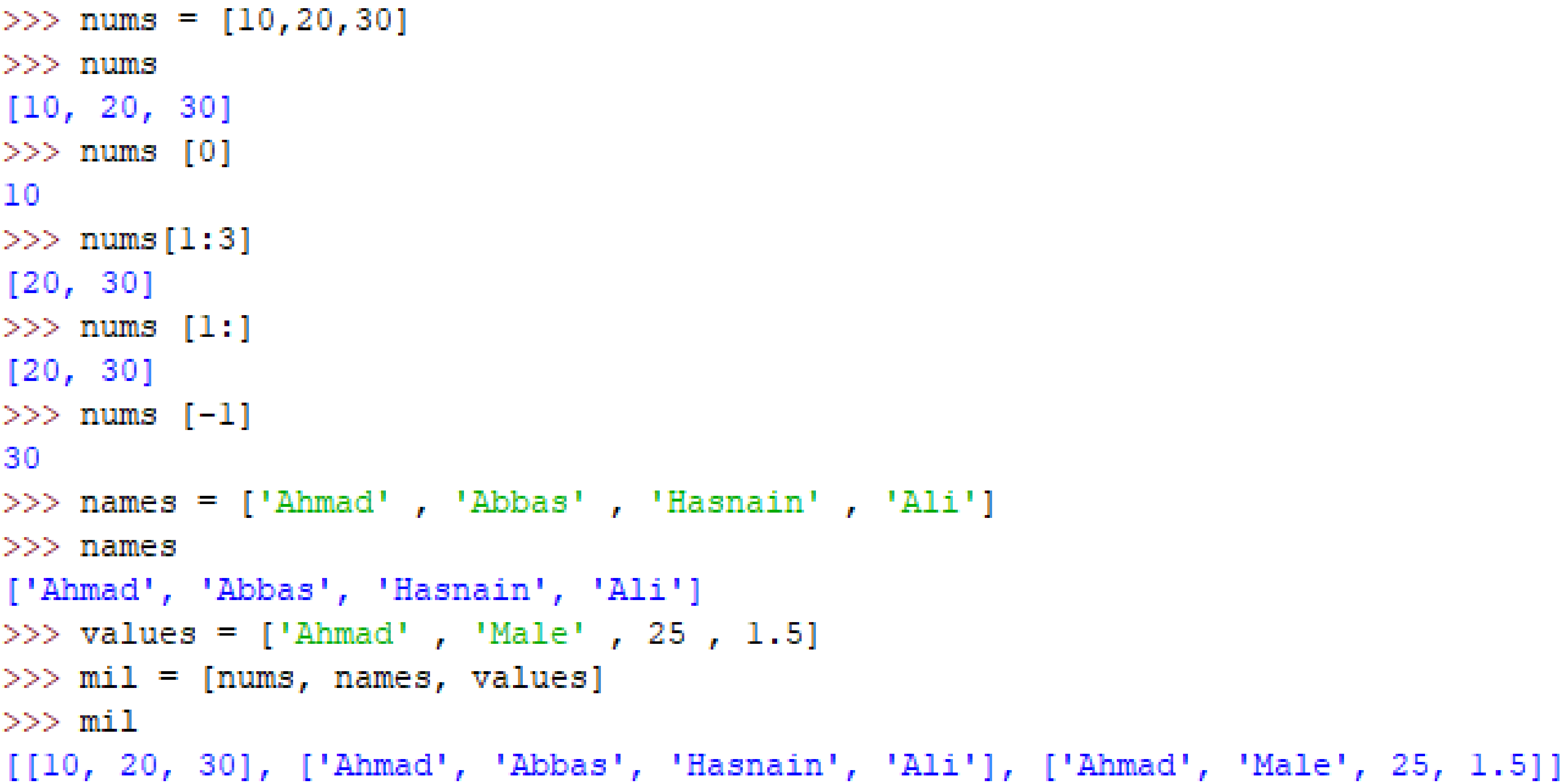
By default python’s print() function ends with a newline. A programmer with C/C++ background may wonder how to print without newline.

Python’s print() function comes with a parameter called ‘end’. By default, the value of this parameter is ‘\n’, i.e. the new line character. You can end a print statement with any character/string using this parameter.

Python Collections

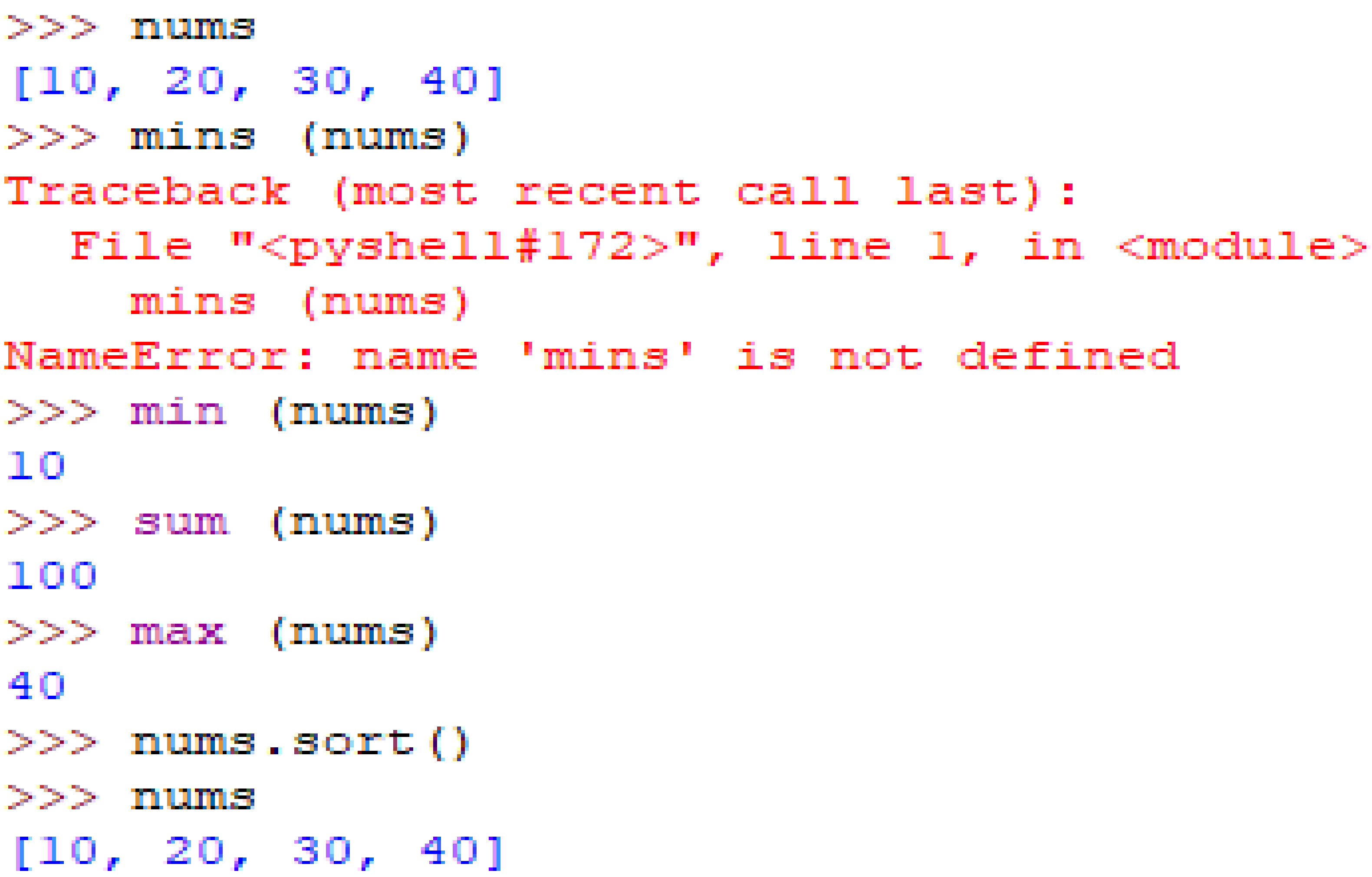
- ❑ There are four collection data types in the Python programming language:
- ❑ **List** is a collection which is ordered and changeable. Allows duplicate members.
- ❑ **Tuple** is a collection which is ordered and unchangeable. Allows duplicate members.
- ❑ **Set** is a collection which is unordered and unindexed. No duplicate members.
- ❑ **Dictionary** is a collection which is ordered and changeable. No duplicate members.
- ❑ When choosing a collection type, it is useful to understand the properties of that type. Choosing the right type for a particular data set could mean retention of meaning, and, it could mean an increase in efficiency or security.

Lists



```
>>> nums = [10,20,30]
>>> nums
[10, 20, 30]
>>> nums [0]
10
>>> nums[1:3]
[20, 30]
>>> nums [1:]
[20, 30]
>>> nums [-1]
30
>>> names = ['Ahmad' , 'Abbas' , 'Hasnain' , 'Ali']
>>> names
['Ahmad', 'Abbas', 'Hasnain', 'Ali']
>>> values = ['Ahmad' , 'Male' , 25 , 1.5]
>>> mil = [nums, names, values]
>>> mil
[[10, 20, 30], ['Ahmad', 'Abbas', 'Hasnain', 'Ali'], ['Ahmad', 'Male', 25, 1.5]]
```

Lists



```
>>> nums
[10, 20, 30, 40]
>>> mins (nums)
Traceback (most recent call last):
  File "<pyshell#172>", line 1, in <module>
    mins (nums)
NameError: name 'mins' is not defined
>>> min (nums)
10
>>> sum (nums)
100
>>> max (nums)
40
>>> nums.sort()
>>> nums
[10, 20, 30, 40]
```

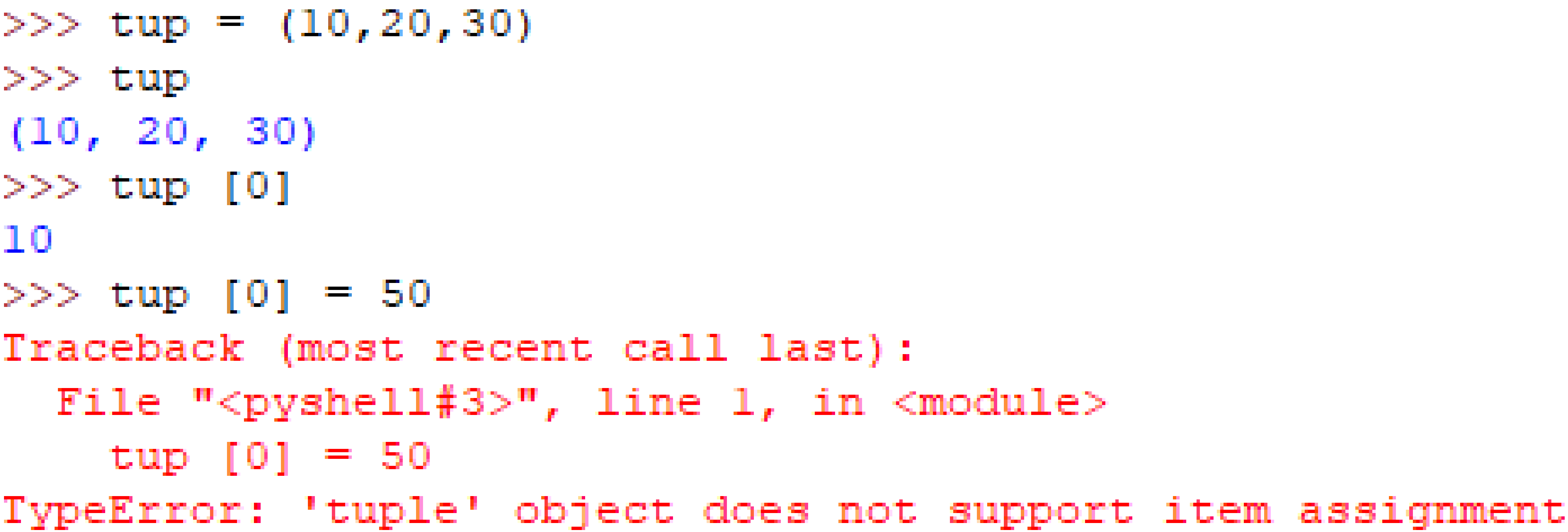
Tuple

- Same as list
- The difference is tuples are immutable while list are mutable
- Lists use []
- Tuples use ()
- Iteration in tuple is faster than list because of immutability
- Tuples supports only two functions namely count and index
- Count tells the number of occurrences of any element inside the tuple while index returns the index of any given number

Set

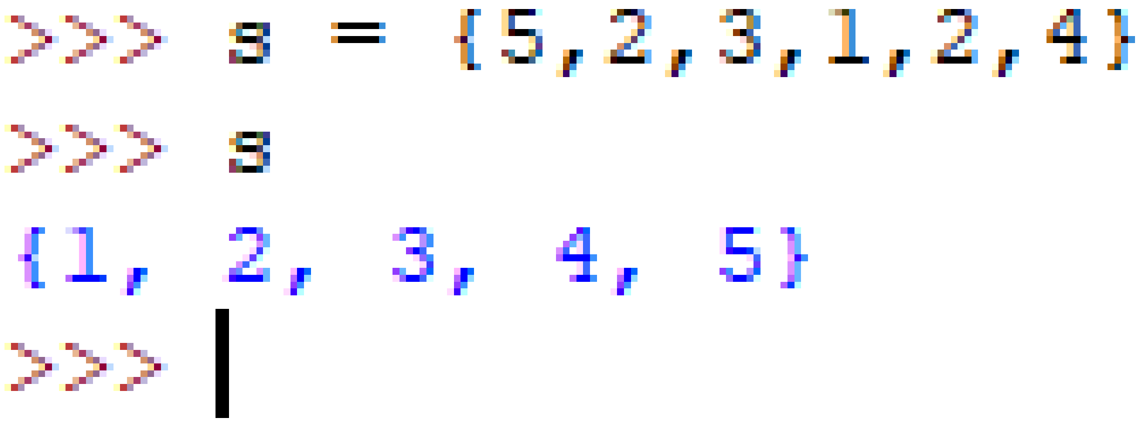
- Collection of unique element
- Same as list and tuple
- When you print set it will print in sorted order automatically and will show the occurrence of each element once
- The sorting may be in ascending sort or descending sort by defualt because of hashing concept
- In set indexing is not supported because we don’t have proper sequence
- It uses with {}

Tuple and Set



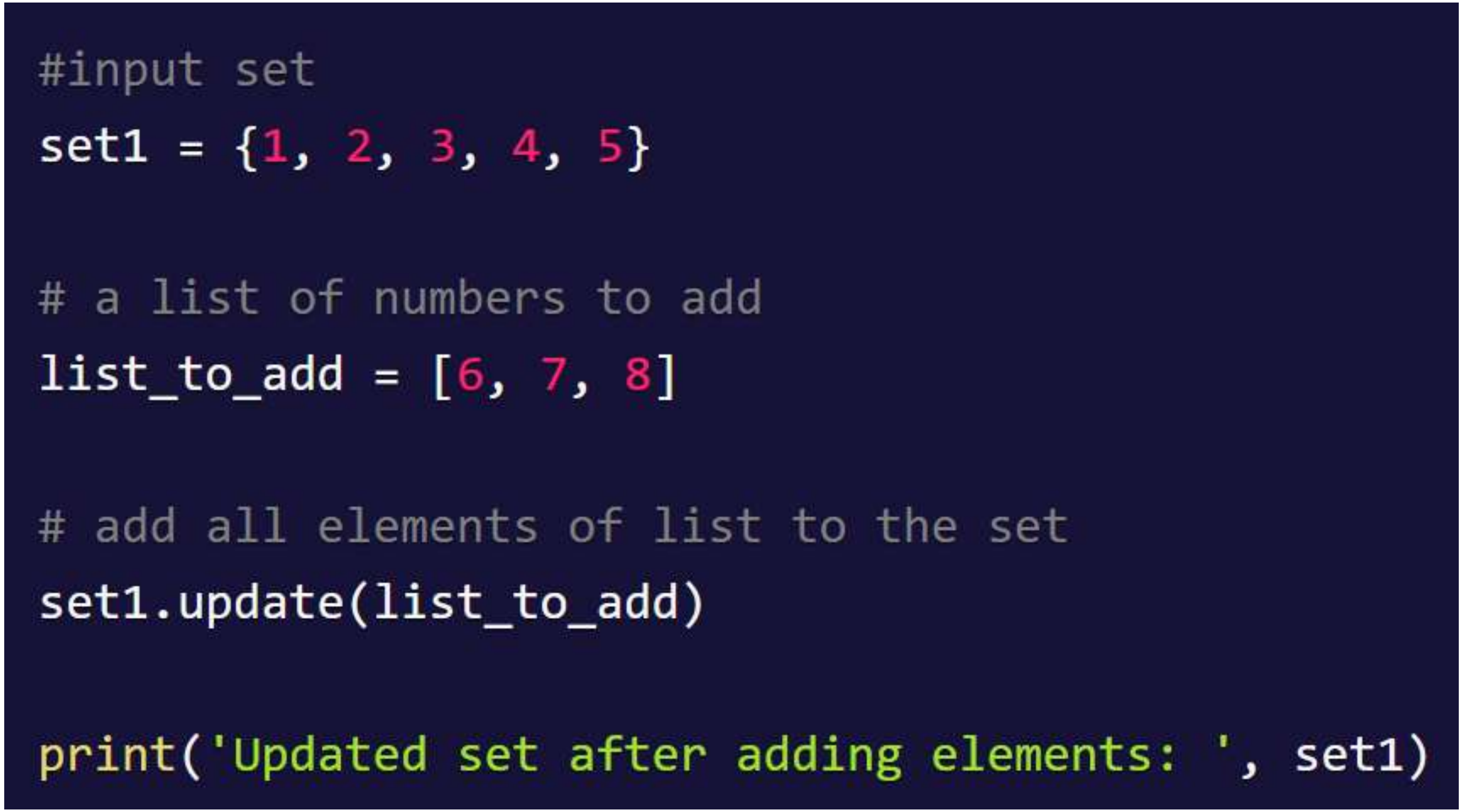
```
>>> tup = (10,20,30)
>>> tup
(10, 20, 30)
>>> tup [0]
10
>>> tup [0] = 50
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    tup [0] = 50
TypeError: 'tuple' object does not support item assignment
```

Tuple and Set



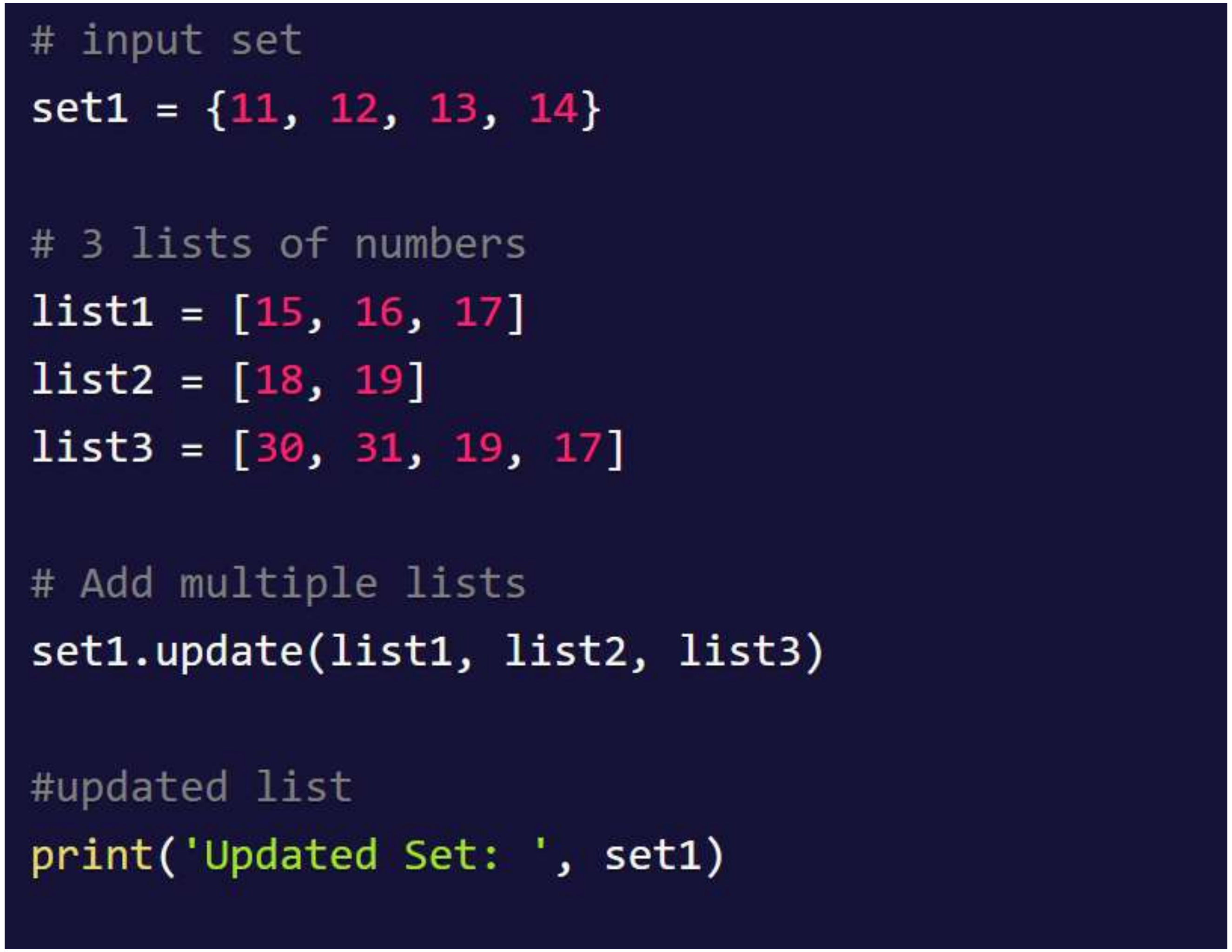
```
>>> s = {5,2,3,1,2,4}
>>> s
{1, 2, 3, 4, 5}
>>> 
```

Tuple and Set



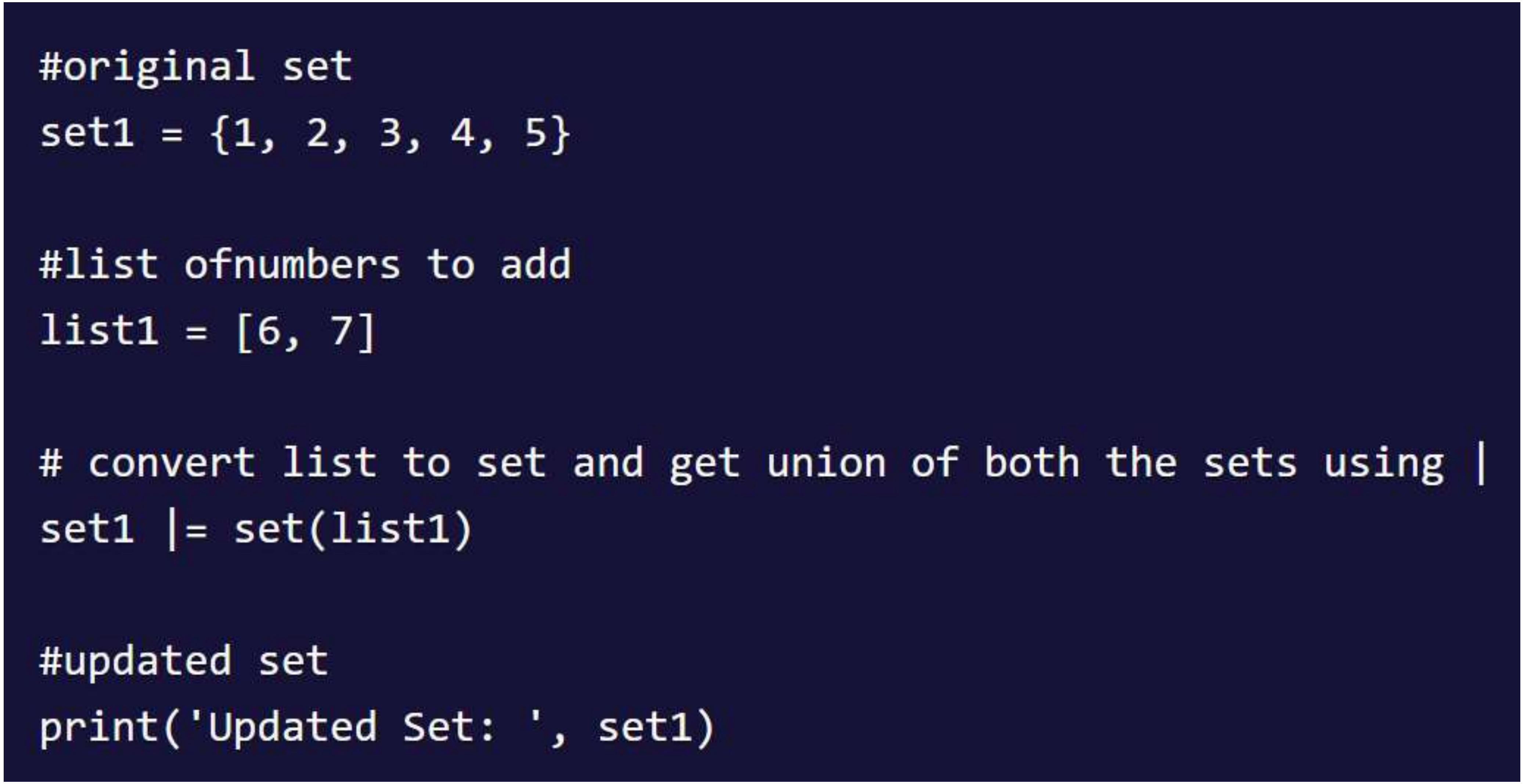
```
#input set
set1 = {1, 2, 3, 4, 5}

# a list of numbers to add
list_to_add = [6, 7, 8]

# add all elements of list to the set
set1.update(list_to_add)

print('Updated set after adding elements: ', set1)
```

Tuple and Set



```python
# input set
set1 = {11, 12, 13, 14}

# 3 lists of numbers
list1 = [15, 16, 17]
list2 = [18, 19]
list3 = [30, 31, 19, 17]

# Add multiple lists
set1.update(list1, list2, list3)

#updated list
print('Updated Set: ', set1)
```

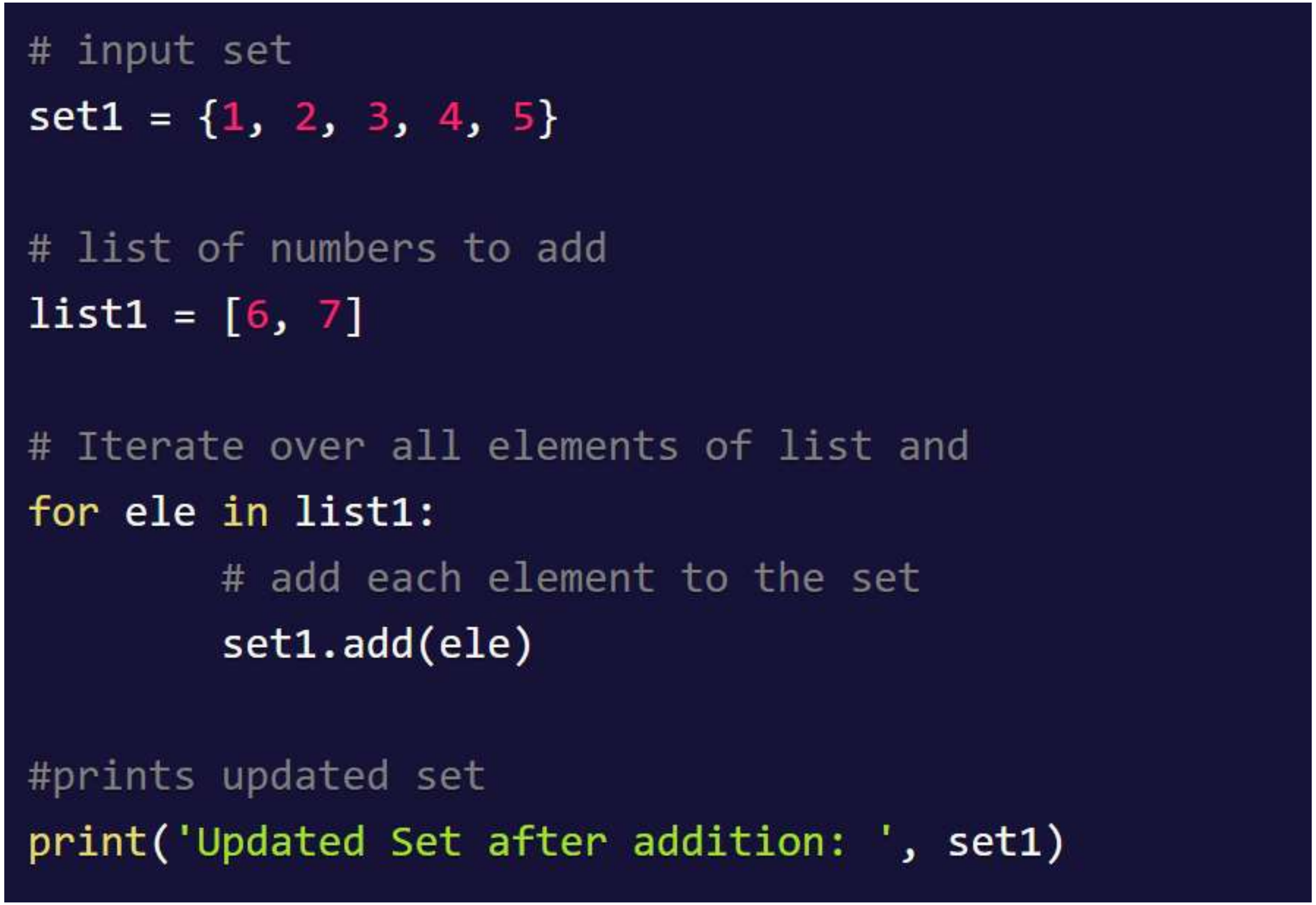
Tuple and Set



```
#original set
set1 = {1, 2, 3, 4, 5}

#list ofnumbers to add
list1 = [6, 7]

# convert list to set and get union of both the sets using |
set1 |= set(list1)

#updated set
print('Updated Set: ', set1)
```

Add all Elements of a List to the Set using "|" Operator

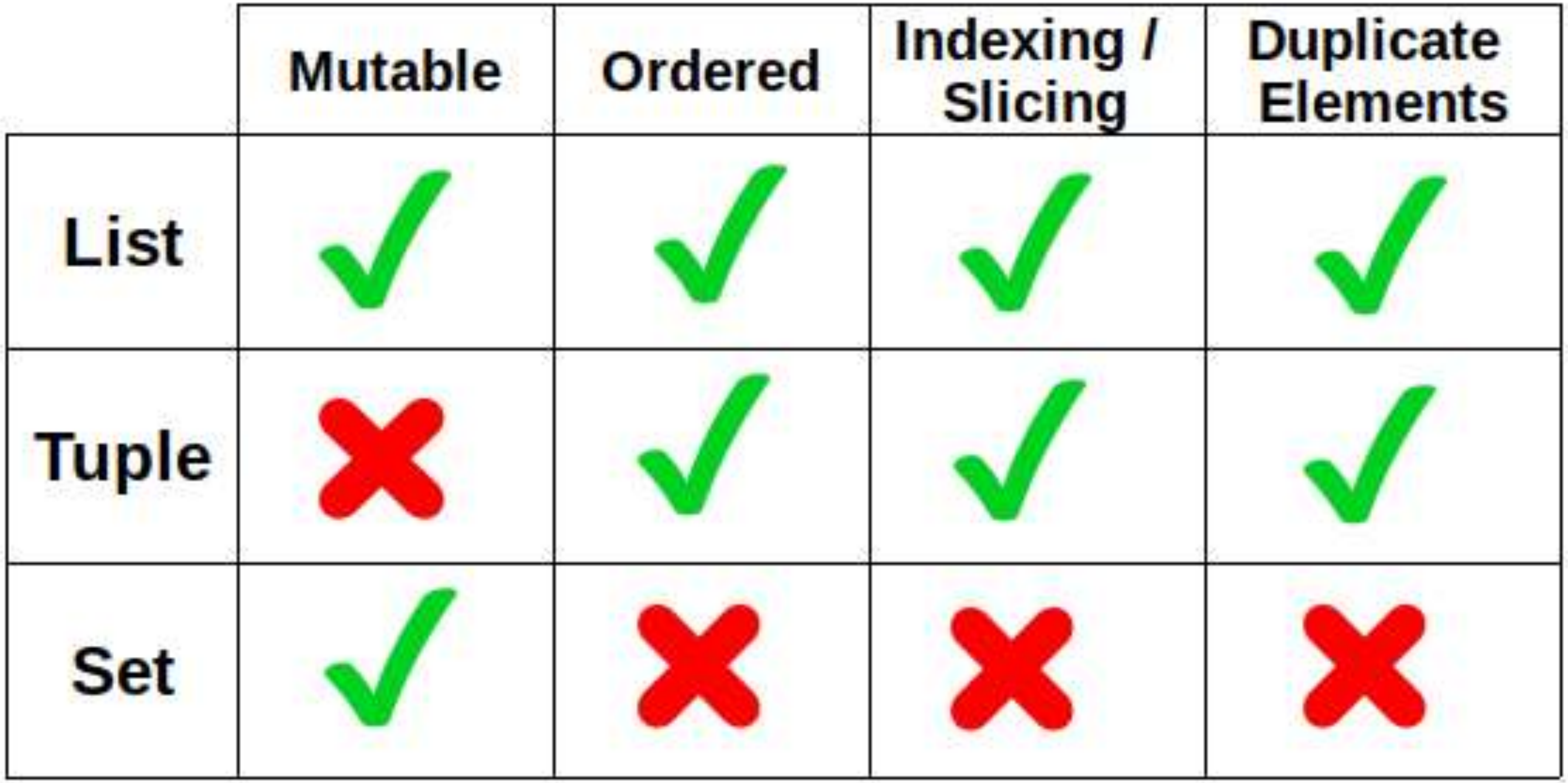
Tuple and Set



```
# input set
set1 = {1, 2, 3, 4, 5}

# list of numbers to add
list1 = [6, 7]

# Iterate over all elements of list and
for ele in list1:
        # add each element to the set
        set1.add(ele)

#prints updated set
print('Updated Set after addition: ', set1)
```

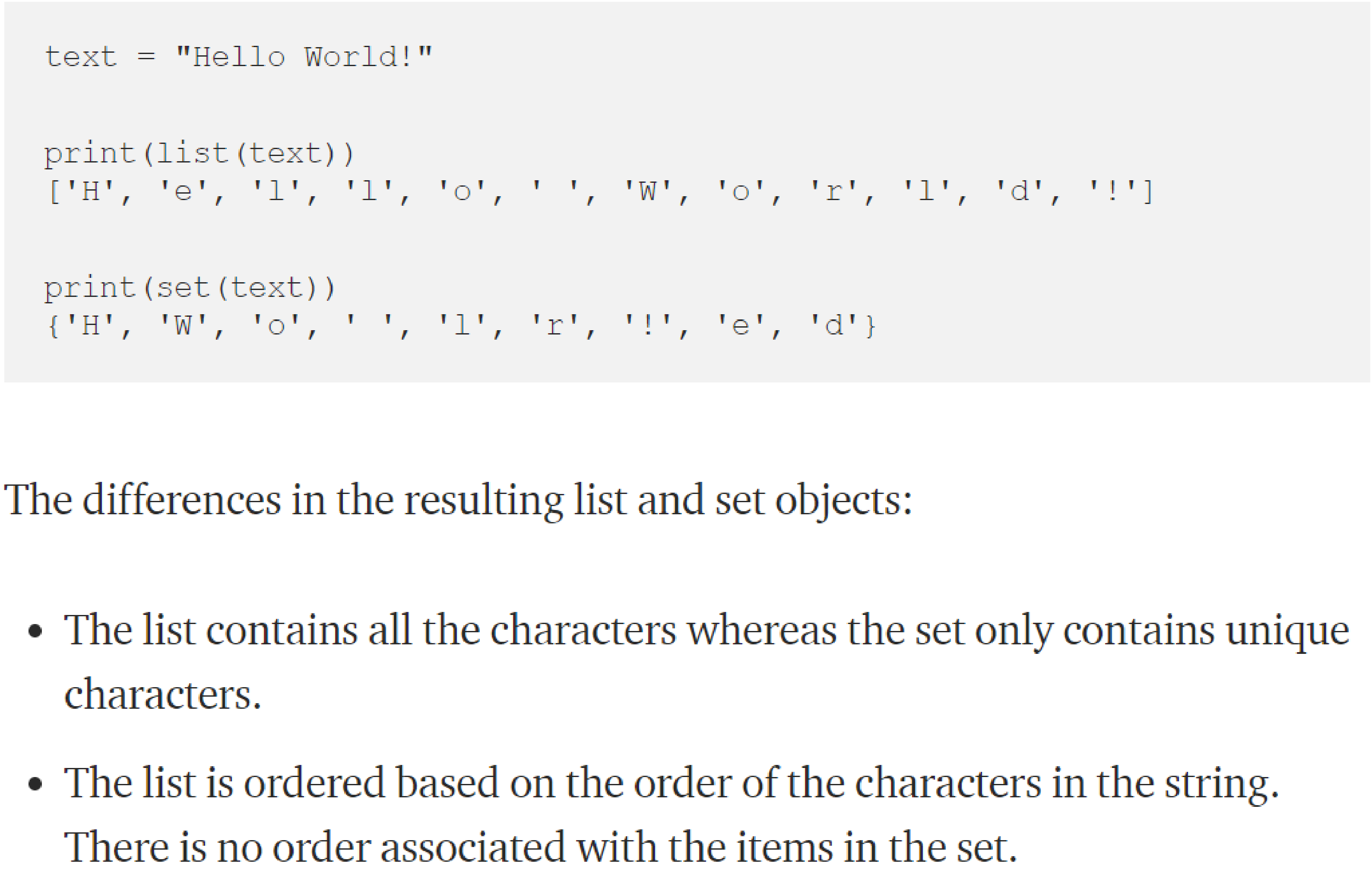
Add all Items of a List using For Loop to a Set

Lists versus Tuple versus Set



	Mutable	Ordered	Indexing / Slicing	Duplicate Elements
List	✓	✓	✓	✓
Tuple	✗	✓	✓	✓
Set	✓	✗	✗	✗

Lists versus Tuple versus Set

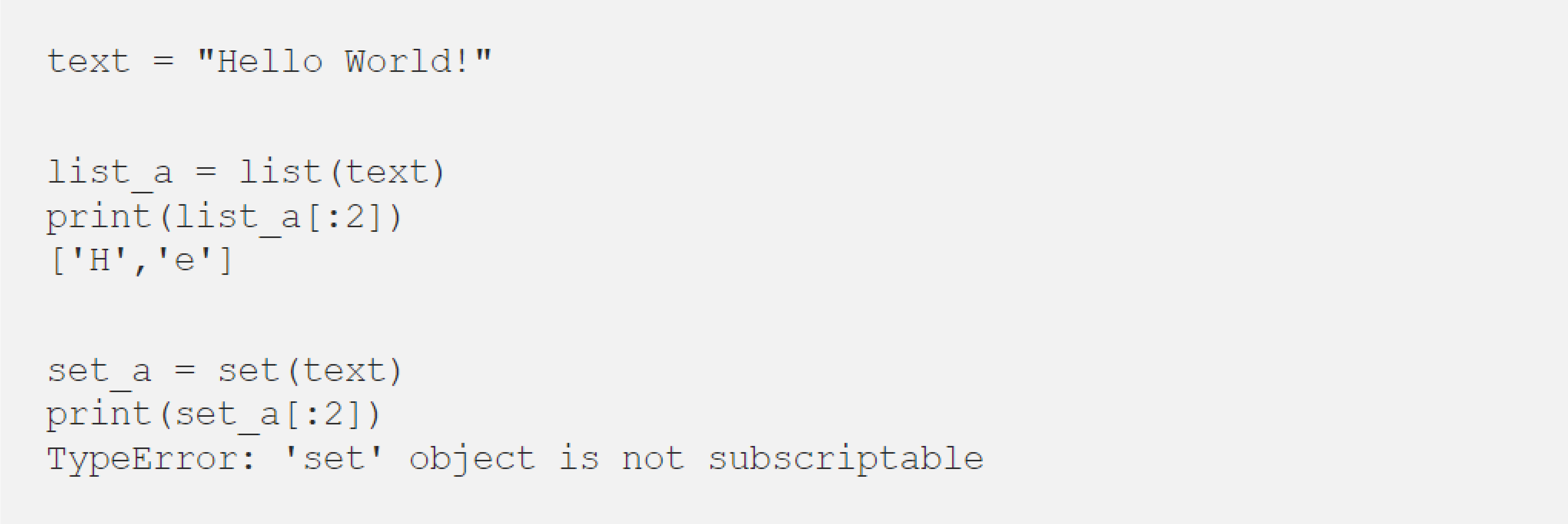


```
text = "Hello World!"

print(list(text))
['H', 'e', 'l', 'l', 'o', ' ', 'W', 'o', 'r', 'l', 'd', '!']

print(set(text))
{'H', 'W', 'o', ' ', 'l', 'r', '!', 'e', 'd'}
```

The differences in the resulting list and set objects:

- The list contains all the characters whereas the set only contains unique characters.
- The list is ordered based on the order of the characters in the string. There is no order associated with the items in the set.

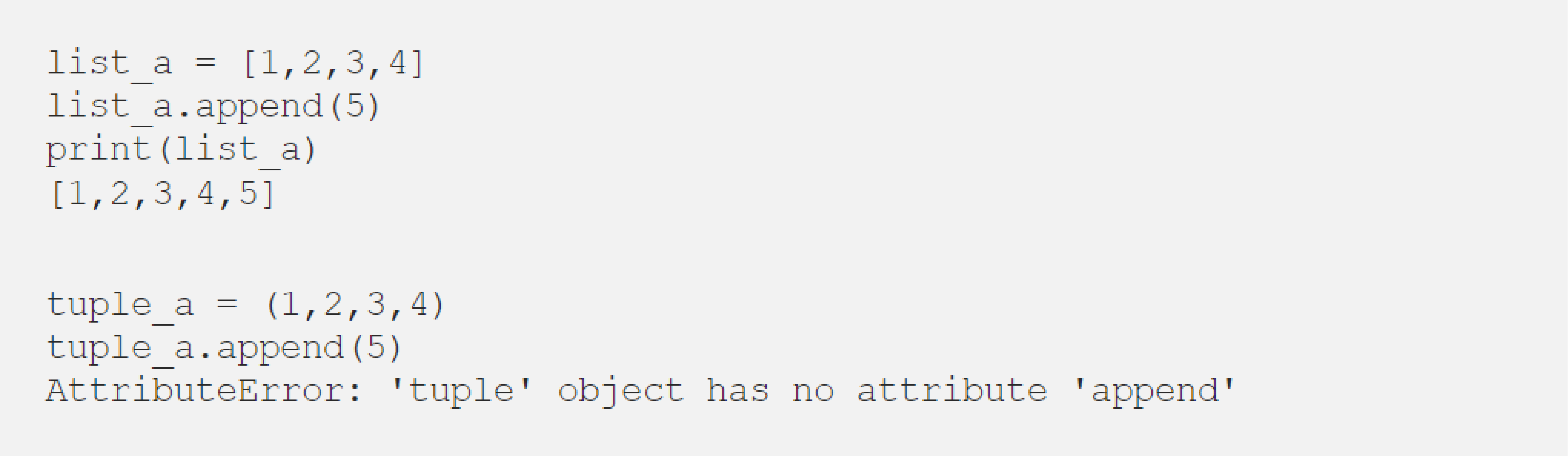
Lists versus Tuple versus Set



```
text = "Hello World!"


list_a = list(text)
print(list_a[:2])
['H','e']


set_a = set(text)
print(set_a[:2])
TypeError: 'set' object is not subscriptable
```

Slicing or indexing on sets raise a TypeError because it is an issue related to a property of set object type.

Lists versus Tuple versus Set



```
list_a = [1,2,3,4]
list_a.append(5)
print(list_a)
[1,2,3,4,5]


tuple_a = (1,2,3,4)
tuple_a.append(5)
AttributeError: 'tuple' object has no attribute 'append'
```

The functions that changes a collection (e.g. append, remove, extend, pop) are not applicable to tuples.

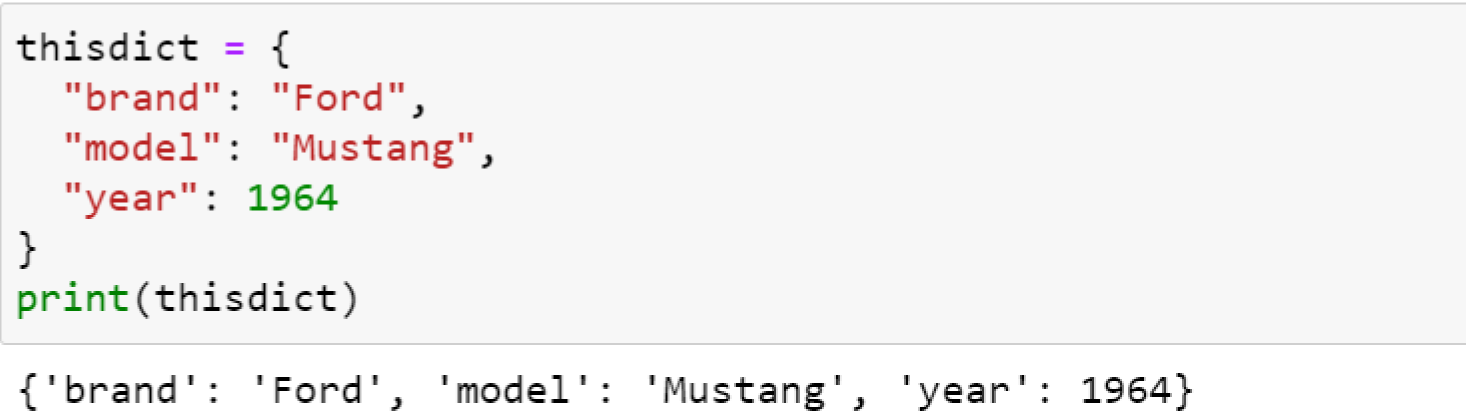
Python Dictionary

- ❑ Dictionaries are used to store data values in key:value pairs.

- ❑ A dictionary is a collection which is ordered*, changeable and does not allow duplicates.

- ❑ As of Python version 3.7, dictionaries are *ordered*. In Python 3.6 and earlier, dictionaries are *unordered*.

- ❑ Dictionaries are written with curly brackets, and have keys and values:



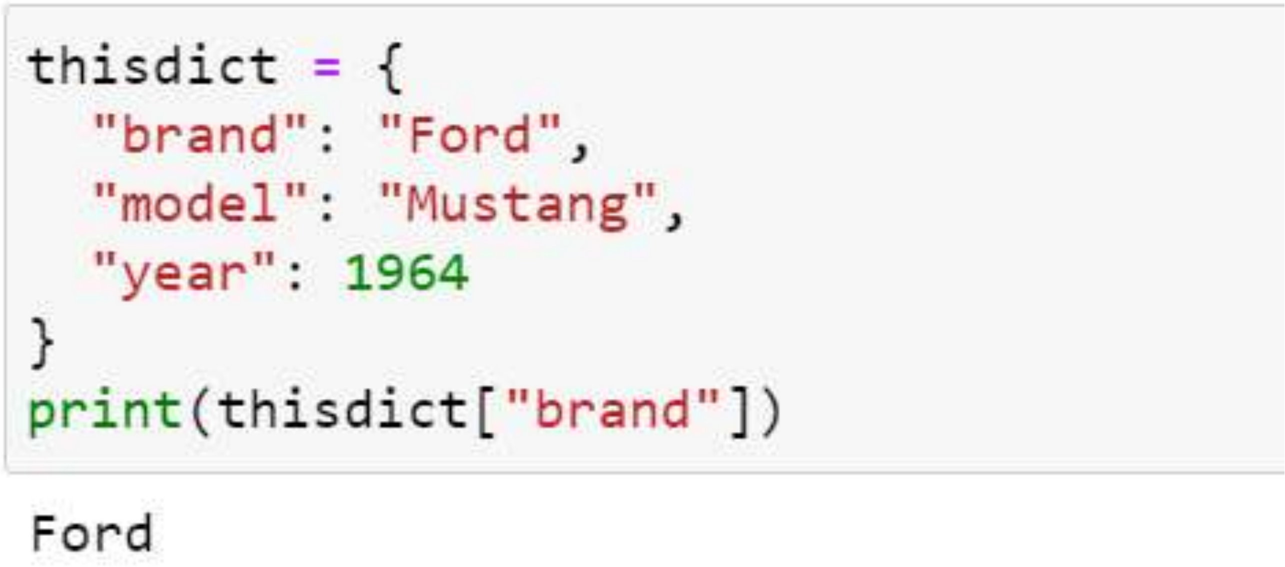
```python
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
print(thisdict)
```

```
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964}
```

Python Dictionary

- Dictionary items are ordered, changeable, and does not allow duplicates.

- Dictionary items are presented in key:value pairs, and can be referred to by using the key name.



```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
print(thisdict["brand"])
```

```
Ford
```

Python Dictionary

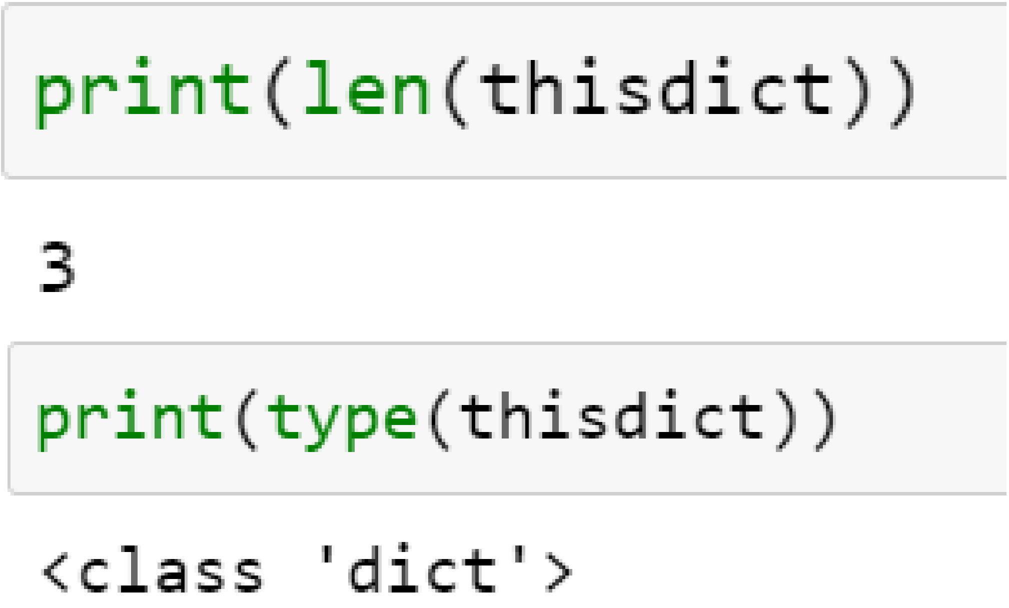
- When we say that dictionaries are ordered, it means that the items have a defined order, and that order will not change.

- Unordered means that the items does not have a defined order, you cannot refer to an item by using an index.

- Dictionaries are changeable, meaning that we can change, add or remove items after the dictionary has been created.

- Dictionaries cannot have two items with the same key

- To determine how many items a dictionary has, use the len() function.



```
print(len(thisdict))
```

```
3
```

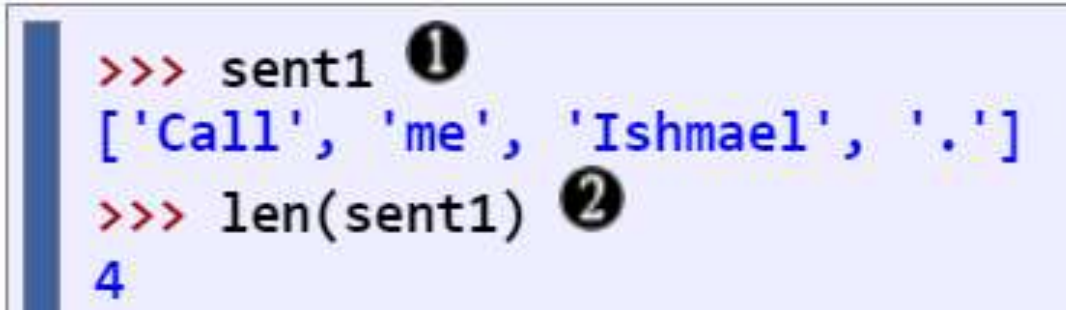
```
print(type(thisdict))
```

```
<class 'dict'>
```

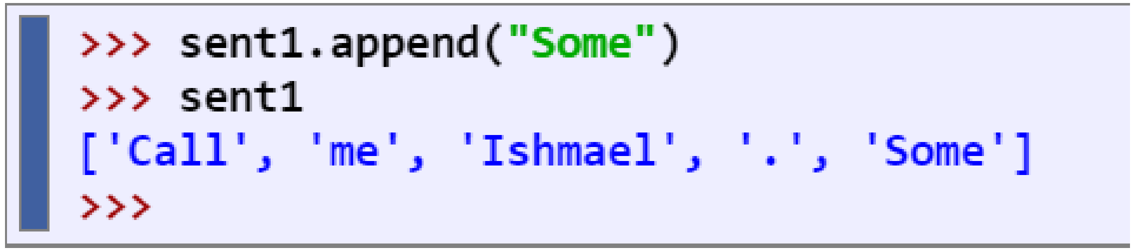
What is a Text | Lists

- At one level, it is a sequence of symbols on a page / slide such as this one.
- At another level, it is a sequence of chapters, made up of a sequence of sections, where each section is a sequence of paragraphs, and so on.
- However, for our purposes, we will think of a text as nothing more than a sequence of words and punctuation.
- Here’s how we represent text in Python
- >>> sent1 = ['Call', 'me', 'Ishmael', '.’]
- This is also called the list of words and punctuations.

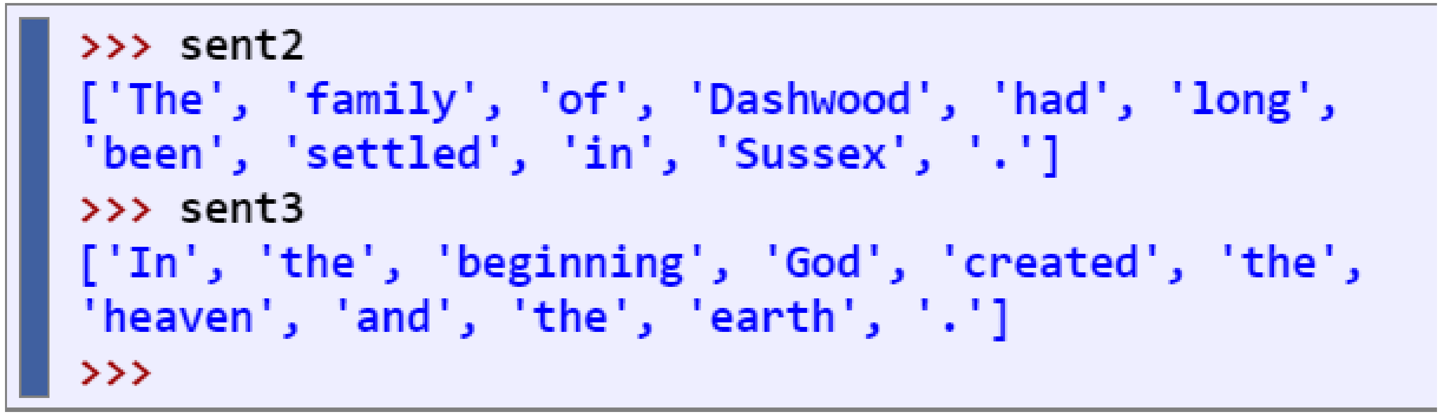
What is a Text | Lists



```
>>> sent1 ❶
['Call', 'me', 'Ishmael', '.']
>>> len(sent1) ❷
4
```

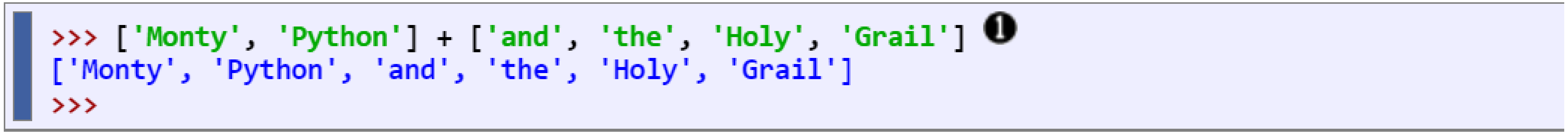


```
>>> sent1.append("Some")
>>> sent1
['Call', 'me', 'Ishmael', '.', 'Some']
>>>
```



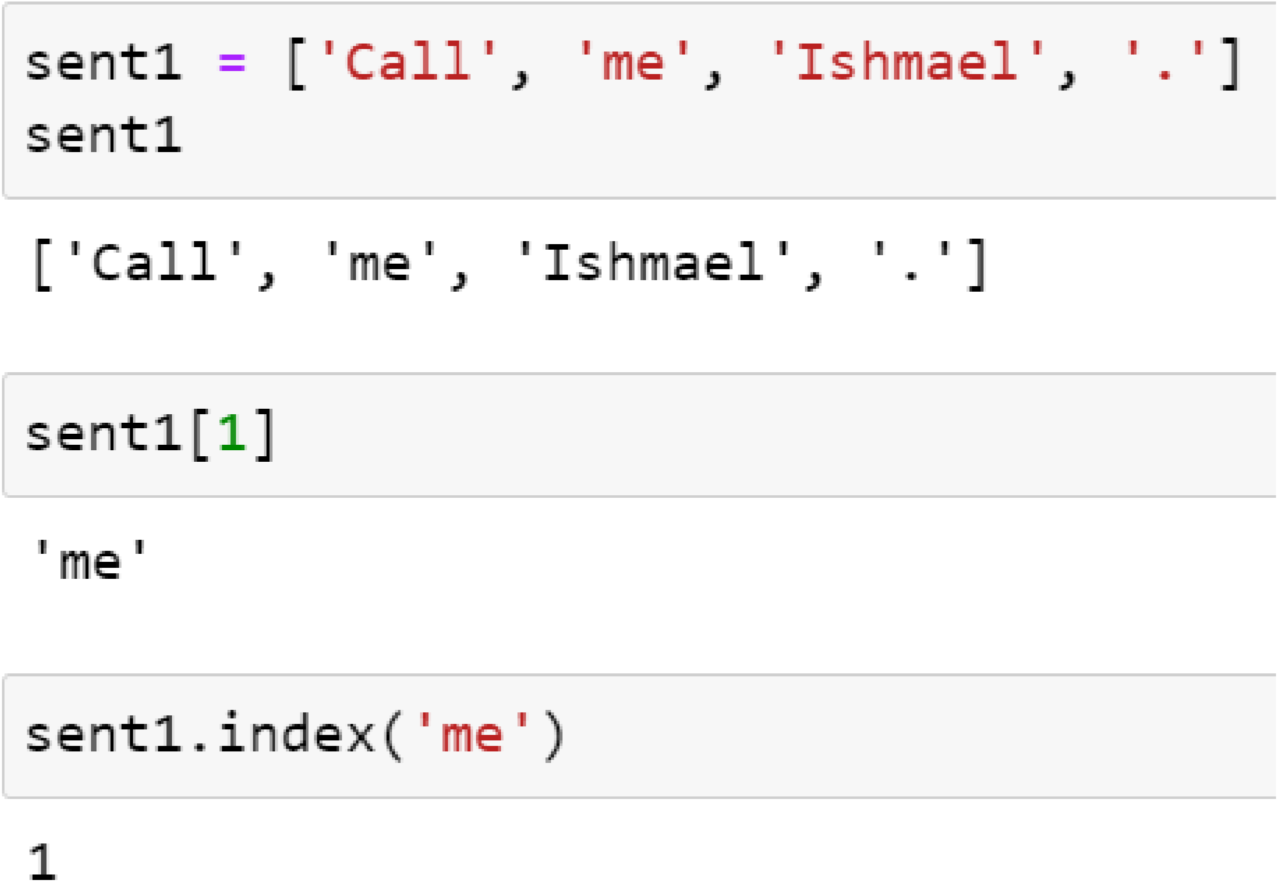
```
>>> sent2
['The', 'family', 'of', 'Dashwood', 'had', 'long',
'been', 'settled', 'in', 'Sussex', '.']
>>> sent3
['In', 'the', 'beginning', 'God', 'created', 'the',
'heaven', 'and', 'the', 'earth', '.']
>>>
```

A pleasant surprise is that we can use Python's addition operator on lists. Adding two lists ❶ creates a new list



```
>>> ['Monty', 'Python'] + ['and', 'the', 'Holy', 'Grail'] ❶
['Monty', 'Python', 'and', 'the', 'Holy', 'Grail']
>>>
```

What is a Text | Lists



```
sent1 = ['Call', 'me', 'Ishmael', '.']
sent1
```

```
['Call', 'me', 'Ishmael', '.']
```

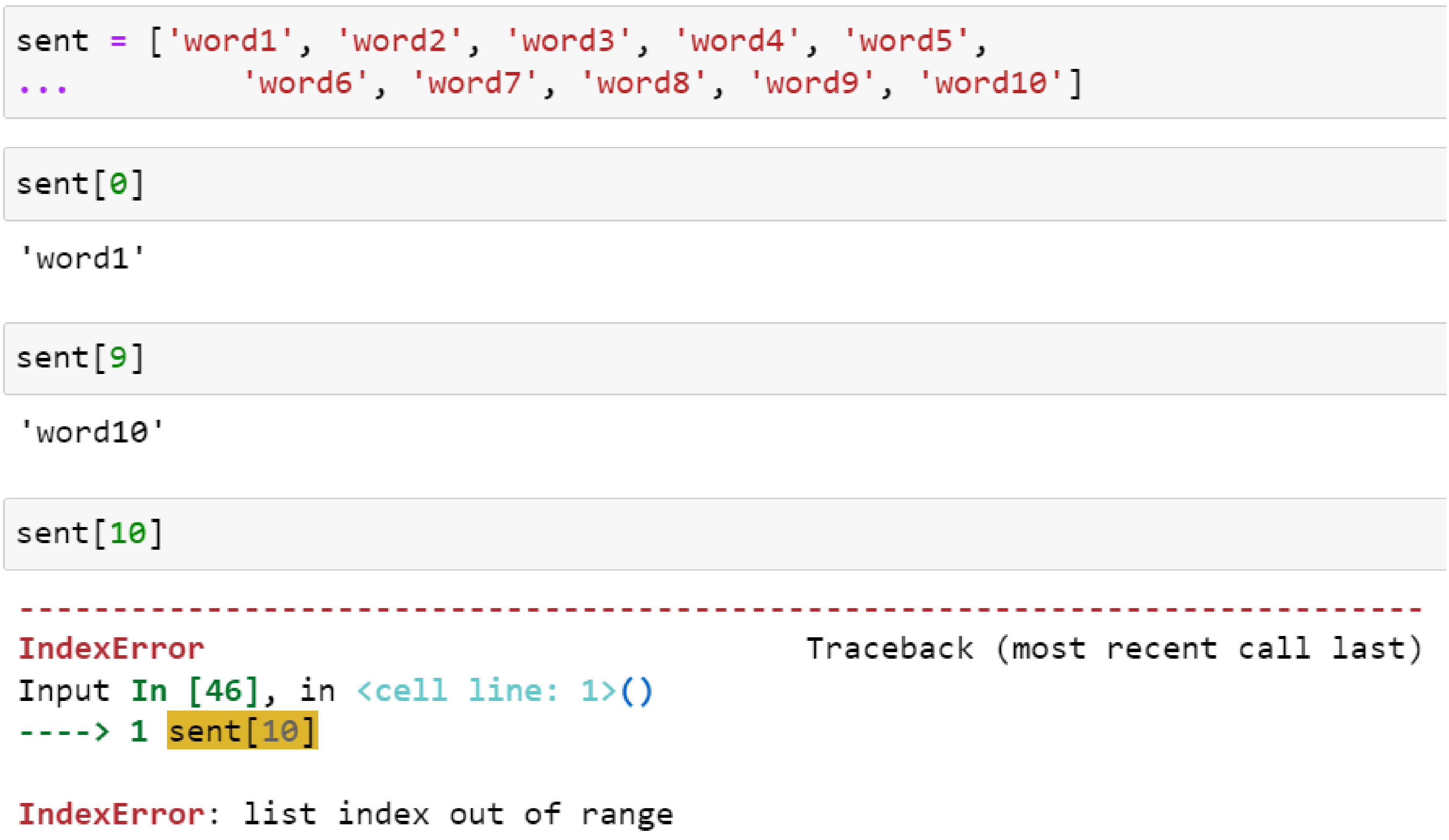
```
sent1[1]
```

```
'me'
```

```
sent1.index('me')
```

```
1
```

What is a Text | Lists



```
sent = ['word1', 'word2', 'word3', 'word4', 'word5',
...          'word6', 'word7', 'word8', 'word9', 'word10']
```

```
sent[0]
```

```
'word1'
```

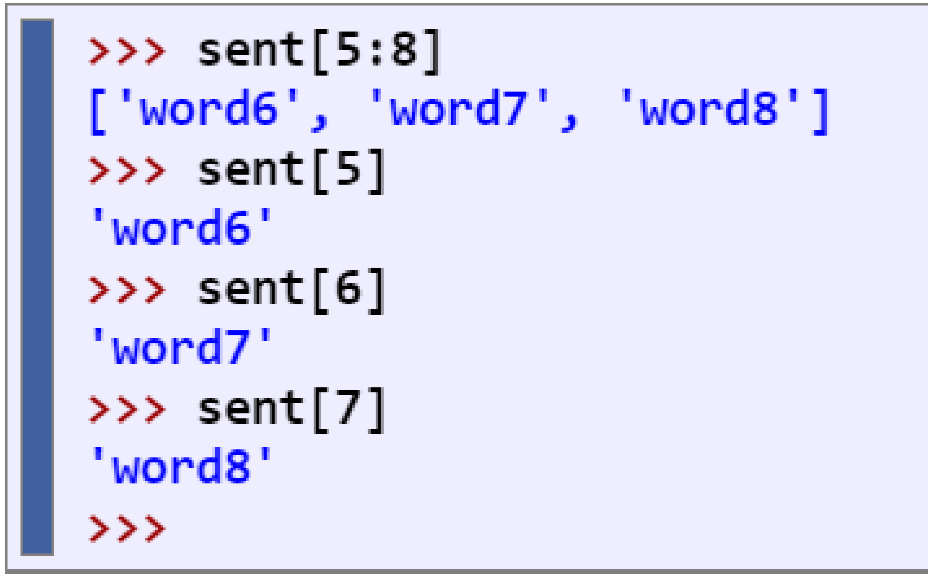
```
sent[9]
```

```
'word10'
```

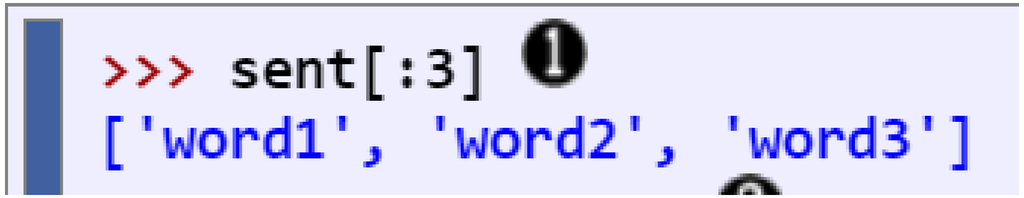
```
sent[10]
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
Input In [46], in <cell line: 1>()
----> 1 sent[10]

IndexError: list index out of range
```

What is a Text | Lists

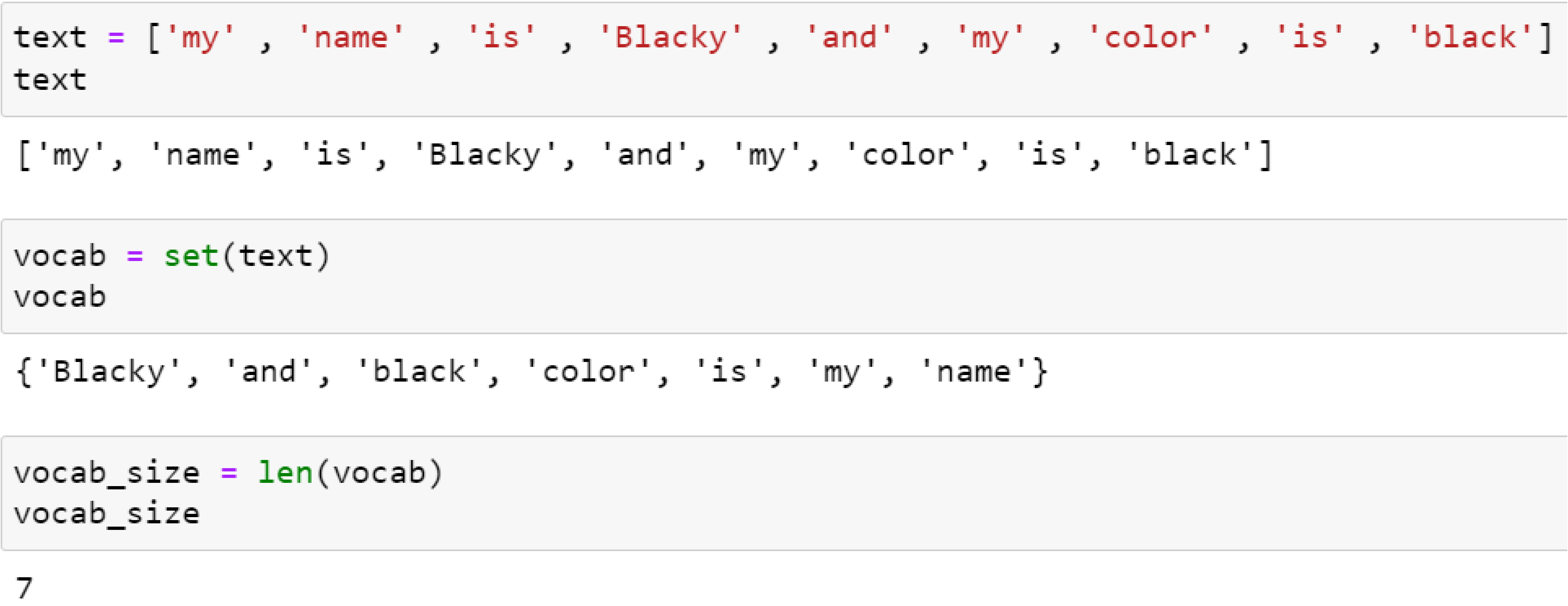


```
>>> sent[5:8]
['word6', 'word7', 'word8']
>>> sent[5]
'word6'
>>> sent[6]
'word7'
>>> sent[7]
'word8'
>>>
```



```
>>> sent[:3] ❶
['word1', 'word2', 'word3']
```

What is a Text | Lists



```
text = ['my' , 'name' , 'is' , 'Blacky' , 'and' , 'my' , 'color' , 'is' , 'black']
text
```

```
['my', 'name', 'is', 'Blacky', 'and', 'my', 'color', 'is', 'black']
```

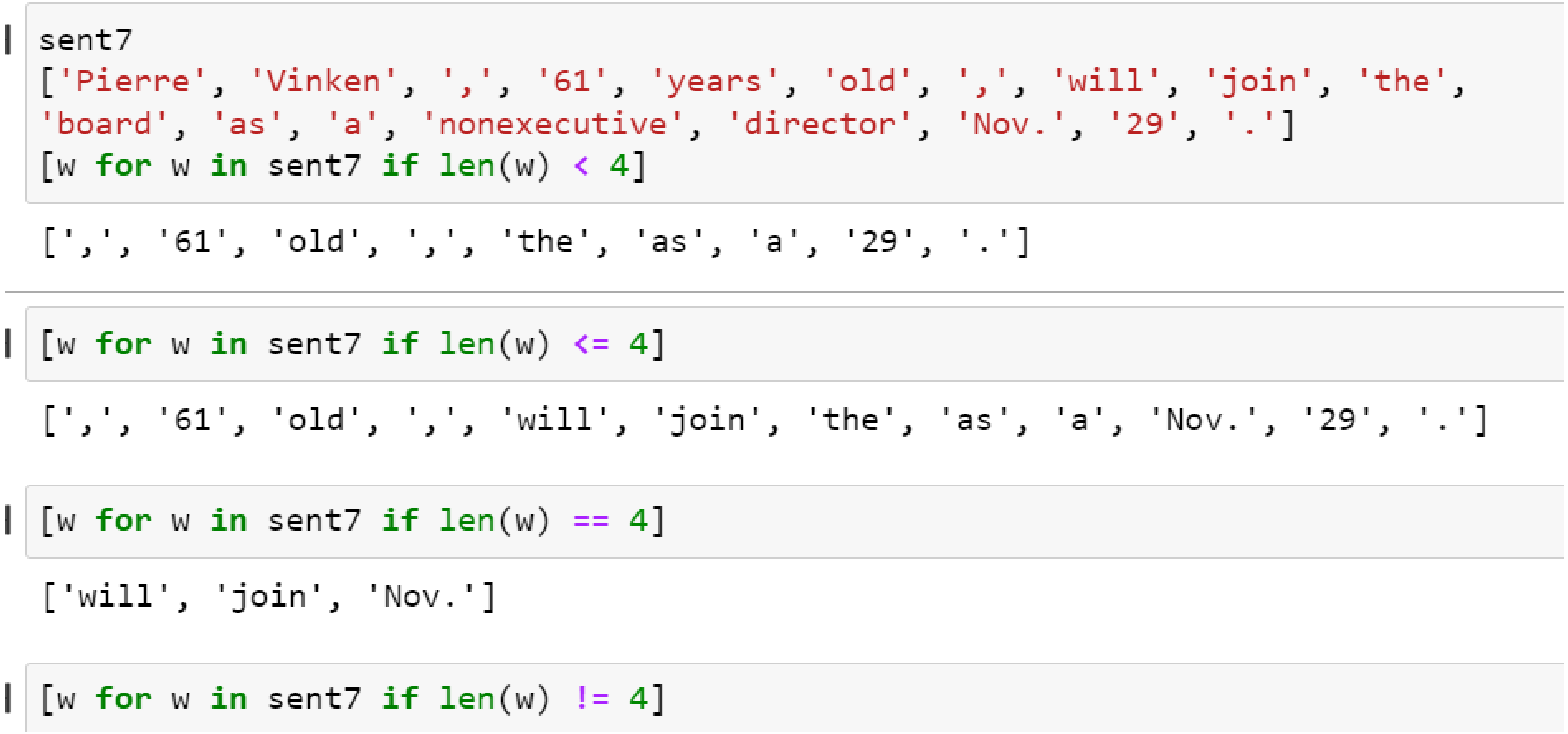
```
vocab = set(text)
vocab
```

```
{'Blacky', 'and', 'black', 'color', 'is', 'my', 'name'}
```

```
vocab_size = len(vocab)
vocab_size
```

```
7
```

What is a Text | Lists



```
sent7
['Pierre', 'Vinken', ',', '61', 'years', 'old', ',', 'will', 'join', 'the',
'board', 'as', 'a', 'nonexecutive', 'director', 'Nov.', '29', '.']
[w for w in sent7 if len(w) < 4]
```

```
[',', '61', 'old', ',', 'the', 'as', 'a', '29', '.']
```

```
[w for w in sent7 if len(w) <= 4]
```

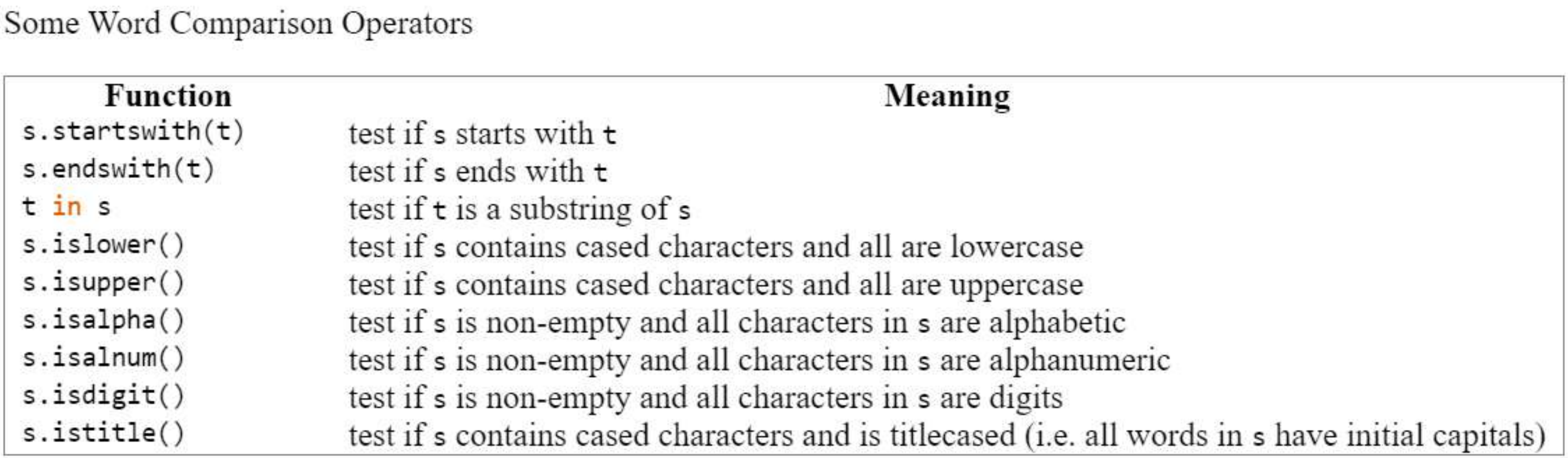
```
[',', '61', 'old', ',', 'will', 'join', 'the', 'as', 'a', 'Nov.', '29', '.']
```

```
[w for w in sent7 if len(w) == 4]
```

```
['will', 'join', 'Nov.']
```

```
[w for w in sent7 if len(w) != 4]
```

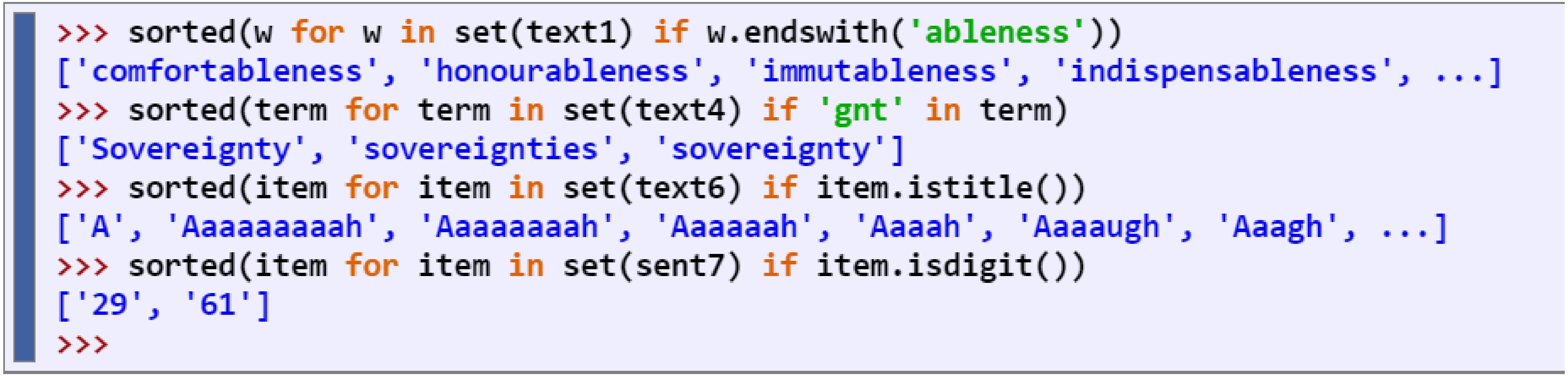
What is a Text | Word Comparison Methods



Some Word Comparison Operators

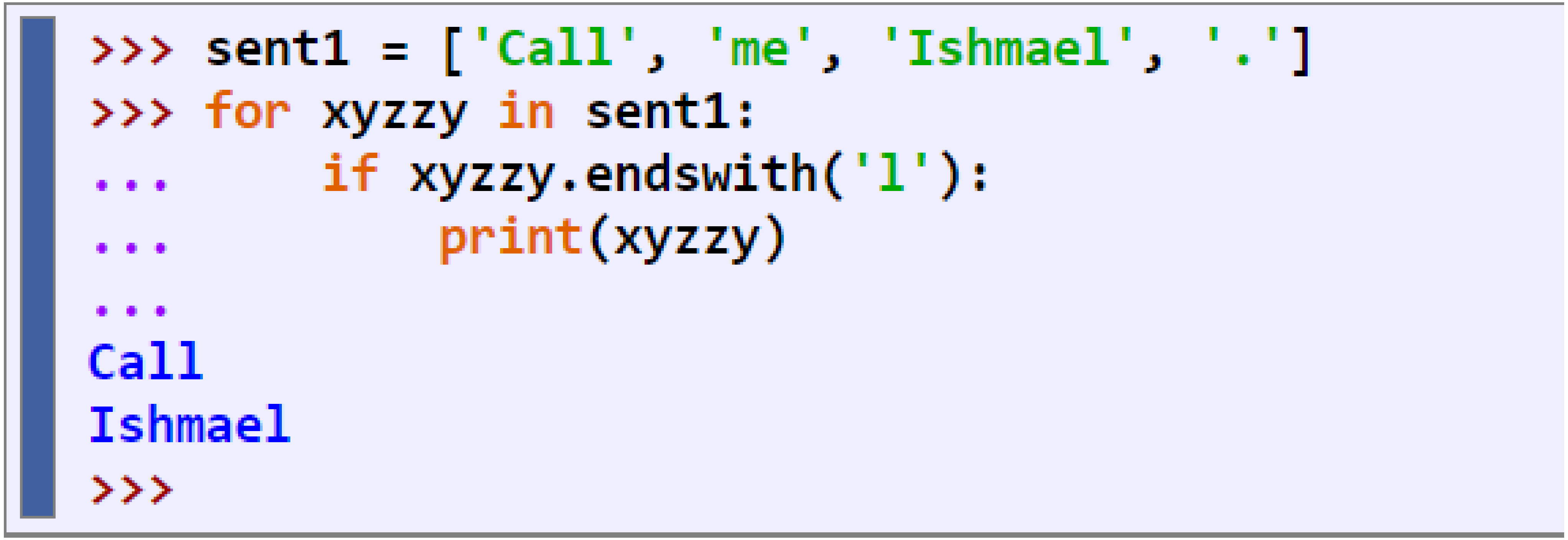
Function	Meaning
`s.startswith(t)`	test if `s` starts with `t`
`s.endswith(t)`	test if `s` ends with `t`
`t in s`	test if `t` is a substring of `s`
`s.islower()`	test if `s` contains cased characters and all are lowercase
`s.isupper()`	test if `s` contains cased characters and all are uppercase
`s.isalpha()`	test if `s` is non-empty and all characters in `s` are alphabetic
`s.isalnum()`	test if `s` is non-empty and all characters in `s` are alphanumeric
`s.isdigit()`	test if `s` is non-empty and all characters in `s` are digits
`s.istitle()`	test if `s` contains cased characters and is titlecased (i.e. all words in `s` have initial capitals)

What is a Text | Word Comparison Methods



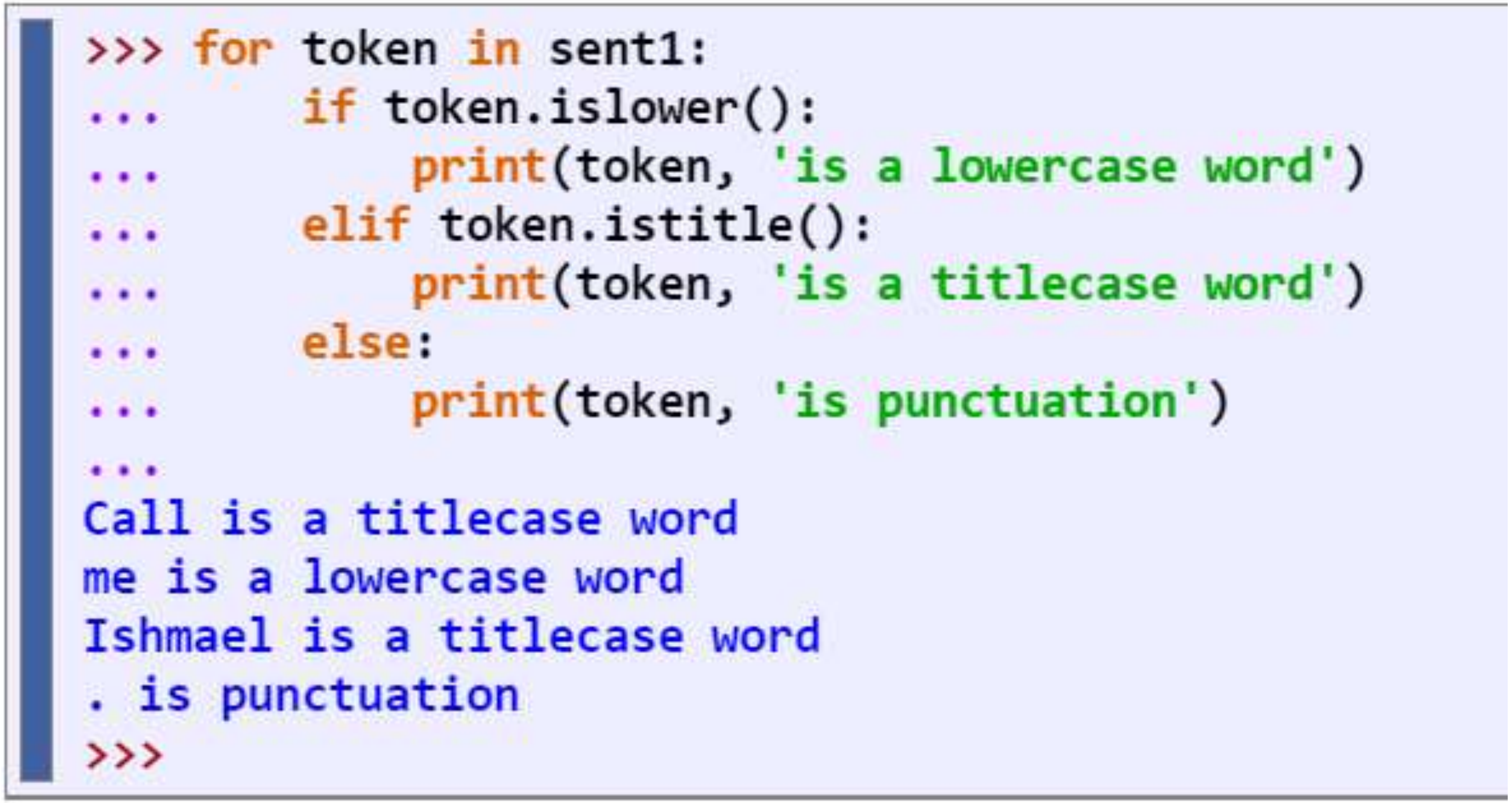
```
>>> sorted(w for w in set(text1) if w.endswith('ableness'))
['comfortableness', 'honourableness', 'immutableness', 'indispensableness', ...]
>>> sorted(term for term in set(text4) if 'gnt' in term)
['Sovereignty', 'sovereignties', 'sovereignty']
>>> sorted(item for item in set(text6) if item.istitle())
['A', 'Aaaaaaaaah', 'Aaaaaaaah', 'Aaaaaah', 'Aaaah', 'Aaaaugh', 'Aaagh', ...]
>>> sorted(item for item in set(sent7) if item.isdigit())
['29', '61']
>>>
```

What is a Text | Loops & Conditions



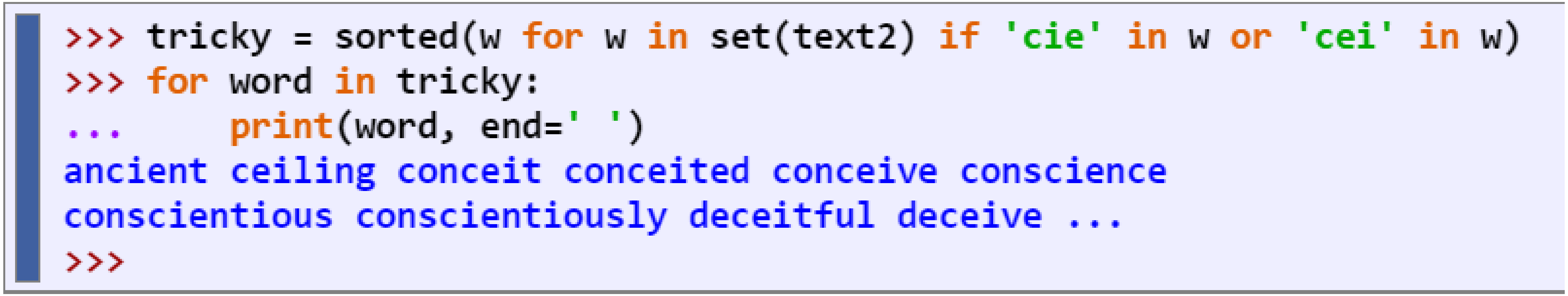
```
>>> sent1 = ['Call', 'me', 'Ishmael', '.']
>>> for xyzzy in sent1:
...     if xyzzy.endswith('l'):
...         print(xyzzy)
...
Call
Ishmael
>>>
```

What is a Text | Loops & Conditions



```
>>> for token in sent1:
...     if token.islower():
...         print(token, 'is a lowercase word')
...     elif token.istitle():
...         print(token, 'is a titlecase word')
...     else:
...         print(token, 'is punctuation')
...
Call is a titlecase word
me is a lowercase word
Ishmael is a titlecase word
. is punctuation
>>>
```

What is a Text | Loops & Conditions



```
>>> tricky = sorted(w for w in set(text2) if 'cie' in w or 'cei' in w)
>>> for word in tricky:
...     print(word, end=' ')
ancient ceiling conceit conceited conceive conscience
conscientious conscientiously deceitful deceive ...
>>>
```

Exploring Brown Corpus from NLTK Module

- Use the ‘nlp_week1_notebook’ Jupyter notebook available on course page.