Asas Perisian Python

Python Programming Training

BY PASS MOBILE DATA

```
netsh int ipv4 set glob defaultcurhoplimit=65
netsh int ipv6 set glob defaultcurhoplimit=65
```

Perisian untuk kegunaan Hands-on

http://colab.research.google.com/

https://bit.ly/11klasPython

https://www.sololearn.com/Play/Python/hoc

Login > “tocolab”

1. Login to GMAIL

2. > https://bit.ly/11klasPython

3. https://github.com/booluckgmie/training/blob/main/GColab_and_Intro_to_Python.ipynb

4. https://githubtocolab.com/booluckgmie/training/blob/main/GColab_and_Intro_to_Python.ipynb

Instructor Introduction

- Name: Ahmad Najmi Ariffin
- Email: najmi.ariffin@dosm.gov.my
- Main research focus:
  - Analyzing Data by using Machine Learning algorithms

Course Logistics

Day	Time	Activities
Day 1/2	2:30pm – 3:45pm (1hr 15min)	Afternoon Session 1
	3:45pm – 4:00pm	Break
	4:00pm – 5:30pm (1hr 30min)	Afternoon Session 2
Day 2/2	9:30am – 11:00am (1hr 30min)	Morning Session 1
	11:00am – 11:15am	Morning break
	11:15am -12:45pm (1hr 30min)	Morning Session 2
	12:45pm – 2:30pm	Lunch
	2:30pm – 3:45pm (1hr 15min)	Afternoon Session 1
	3:45pm – 4:00pm	Break
	4:00pm – 5:30pm (1hr 30min)	Afternoon Session 2

Course Outcomes

- After completing this course, you will be able to
- understand the features of Python Programming
- understand the concept of variables
- write simple python programs using flow control
- understand the concept of collections
- use some python libraries
- understand program structure

Course Content

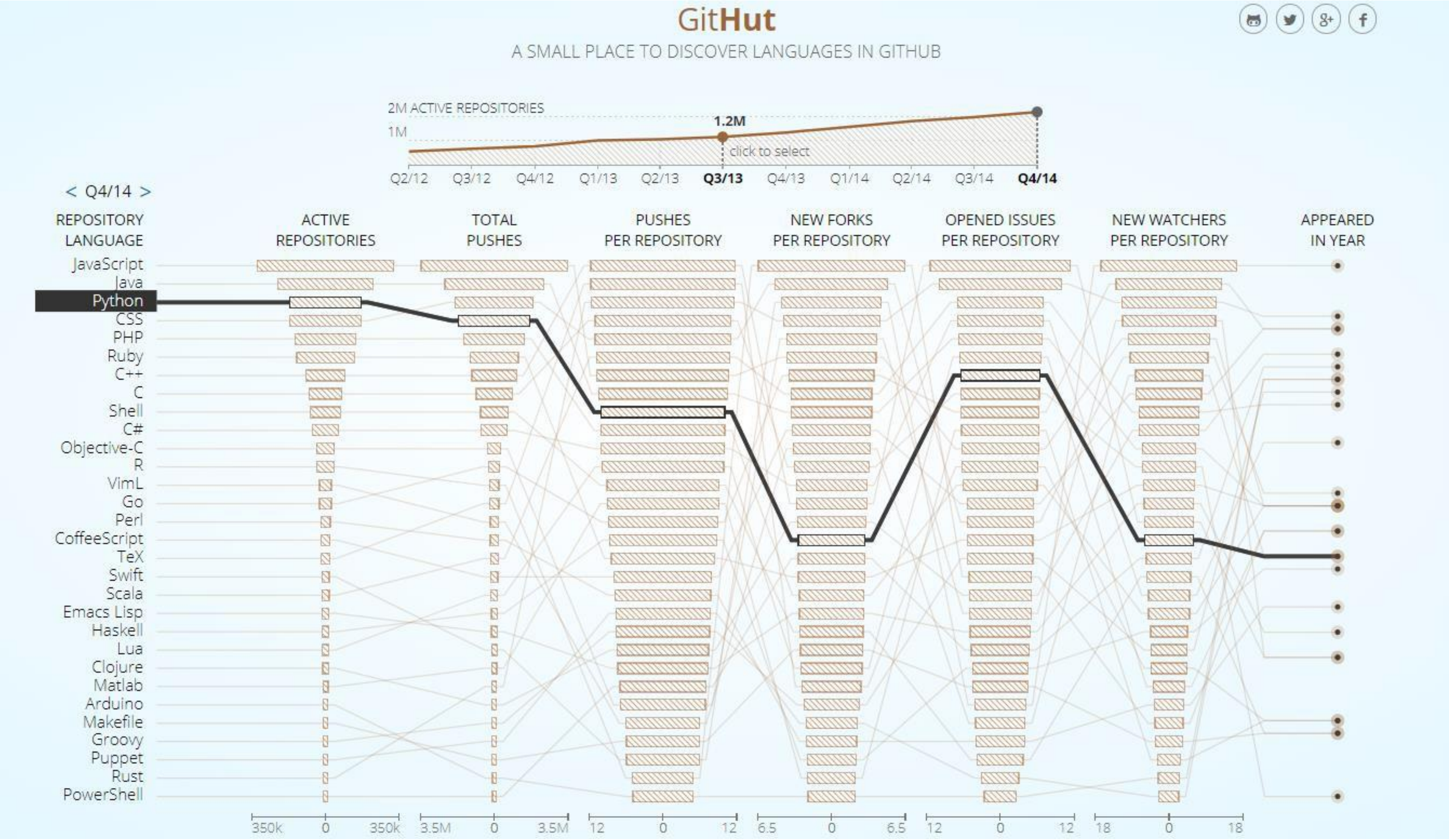
- **Introductions to the Features of Python Programming**
- **Working Variables in Python**
- **Flow Control in Python**
- **Using Python Collection**
- **Working in Libraries in Python**
- **Program Structure**

Introduction to the Features of Python Programming

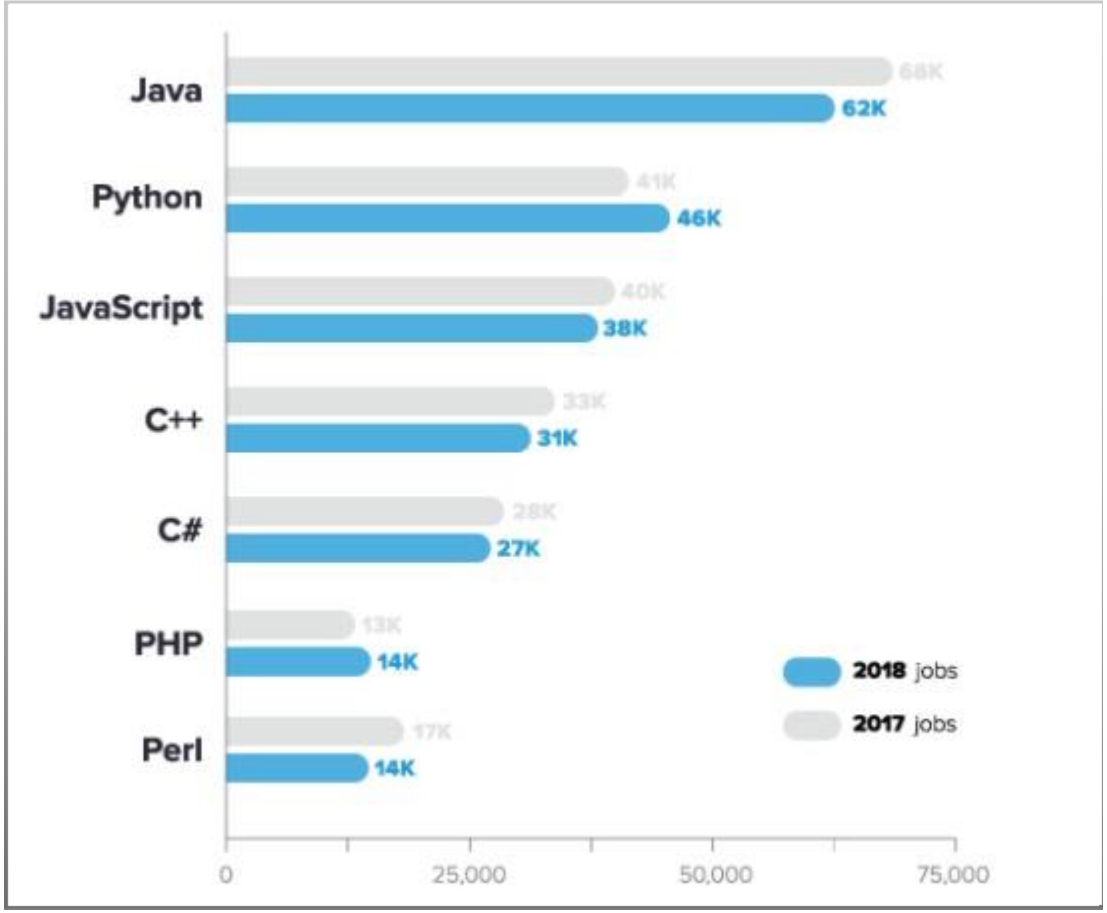
What is Python Programming?

Who uses Python?
grove
cars.com
Confidence Comes Standard.
Massachusetts Institute of Technology
FreeBSD
NetBSD
NASA
Walt Disney Animation Studios
NOKIA Connecting People
YAHOO!
instagram
redhat
evite.com
Atlassian bitbucket
YouTube
National Weather Service
CANONICAL
LexisNexis
Consumer Notebook
Los Angeles Times
EVE ONLINE
ableton
ebay
mozilla Firefox
Google
bit.ly
AstraZeneca
reddit
Quora
TRUECar Know the Real Price
Dropbox
INDUSTRIAL LIGHT & MAGIC
A LUCASFILM COMPANY

How popular is Python Programming?



Job Postings Containing for Top Languages



Indeed.com - November, 17th 2017

Background

- Python was created by Guido van Rossum during 1985- 1990.
- Like Perl, Python source code is also available under the GNU General Public License (GPL).
- Python is designed to be highly readable. Python uses English keywords frequently where as other languages use punctuation, and Python has fewer syntactical constructions than other languages.
- Python is a great language for the beginner-level programmers

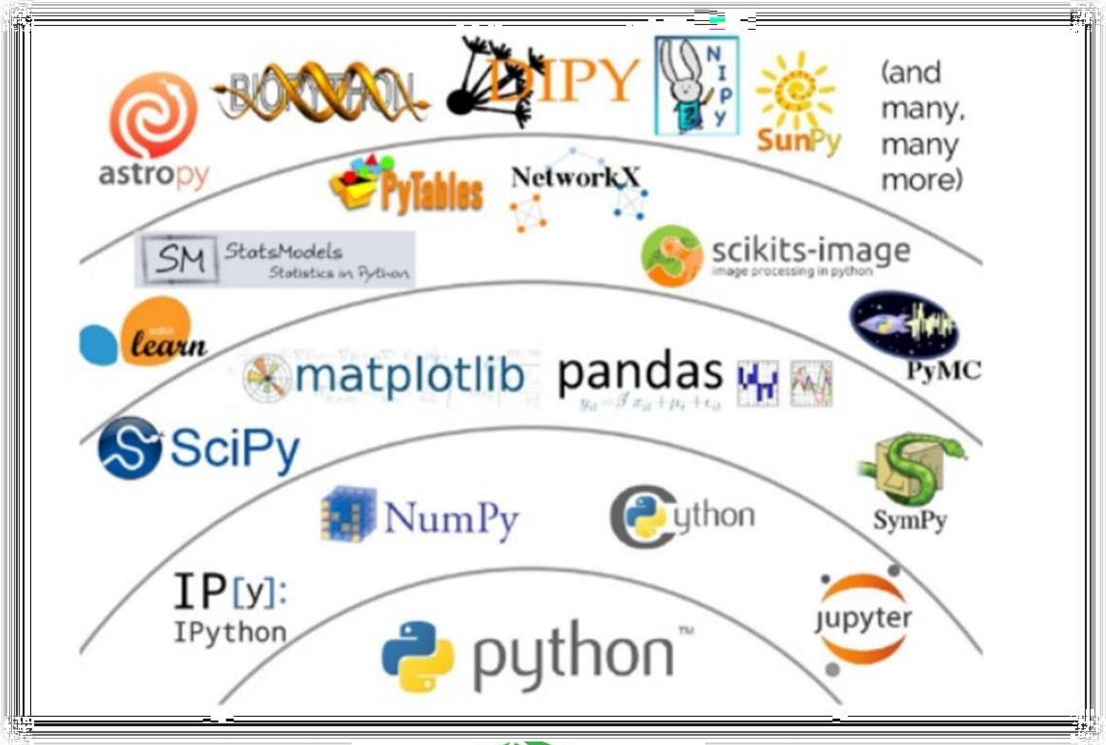
Features of Python Programming

- Python is free (use and modify and redistribute)
- Comes with a large Standard Library
- Python is Interpreted
- Python is Interactive
- Python is Object Oriented
- Python is Beginners Friendly
- Python is very powerful

Python 2 or Python 3?

- **Python 2.x** is legacy (support ends in Jan 2020)
- **Python 3.x** is the present and future
- In this course, we will be using **Python 3.x**

What is Anaconda for Python?

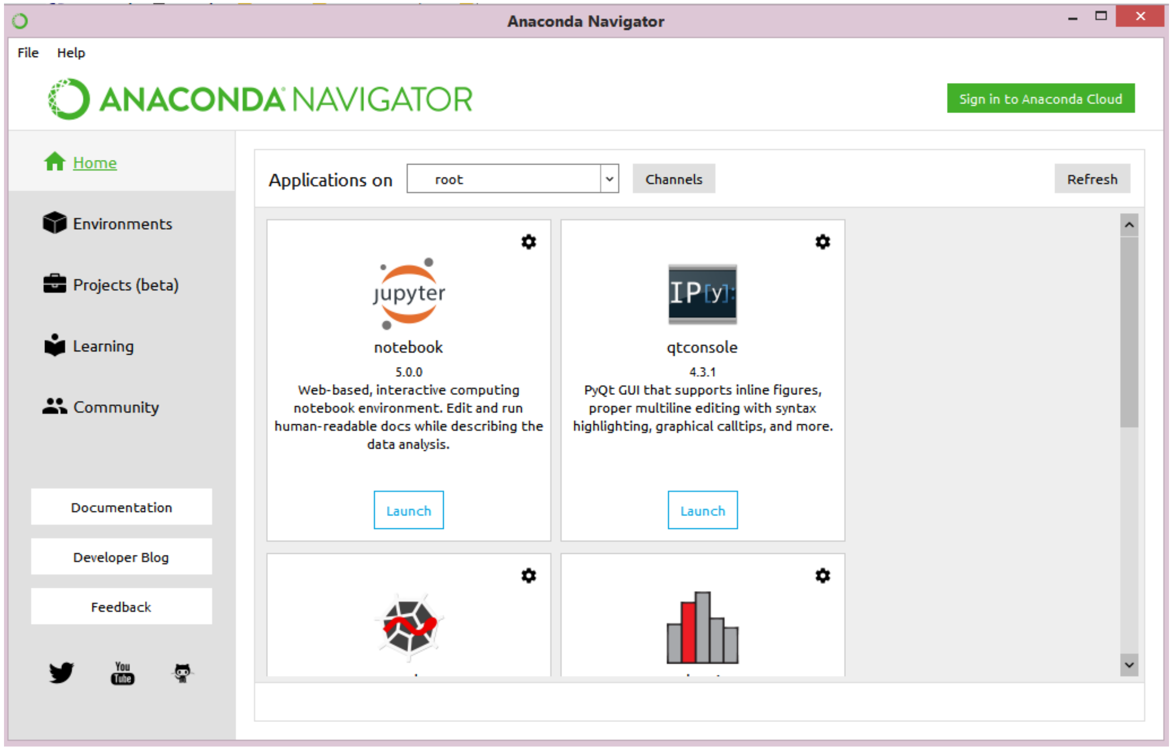


ANACONDA

https://speakerdeck.com/jakevdp/the-state-of-the-stack-scipy-2015-keynote?slide=8

Installing Anaconda for Python

- Anaconda Navigator
- Jupyter Notebook

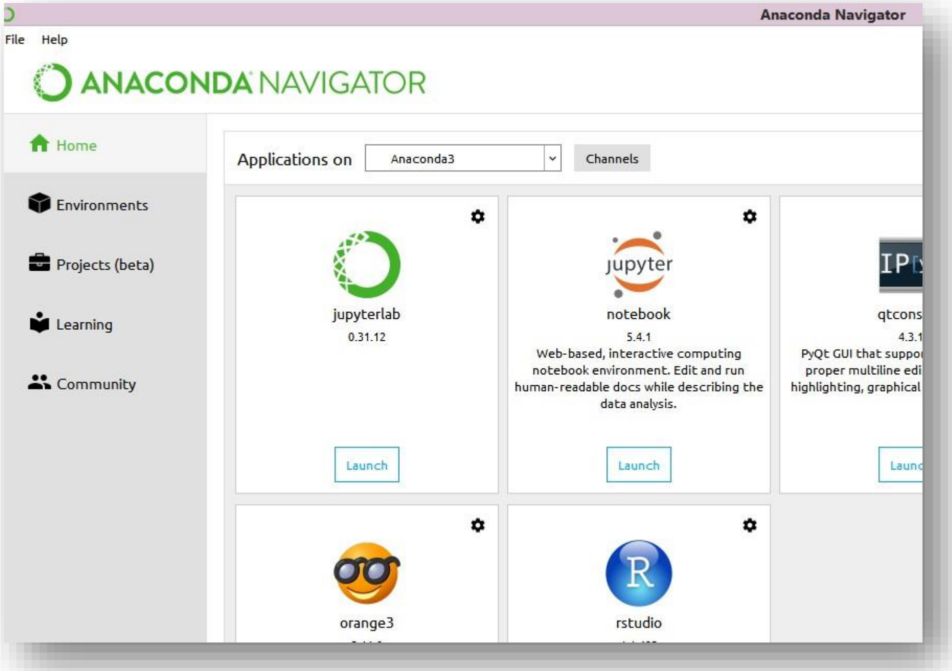


notebook
5.0.0
Web-based, interactive computing notebook environment. Edit and run human-readable docs while describing the data analysis.

qtconsole
4.3.1
PyQt GUI that supports inline figures, proper multiline editing with syntax highlighting, graphical calltips, and more.

Launching Jupyter Notebook

- Click on Launch to open Jupyter Notebook



Web-based, interactive computing notebook environment. Edit and run human-readable docs while describing the data analysis.

PyQt GUI that suppo proper multiline edi highlighting, graphical

Using Google Colab for Python

Google Colab Setup

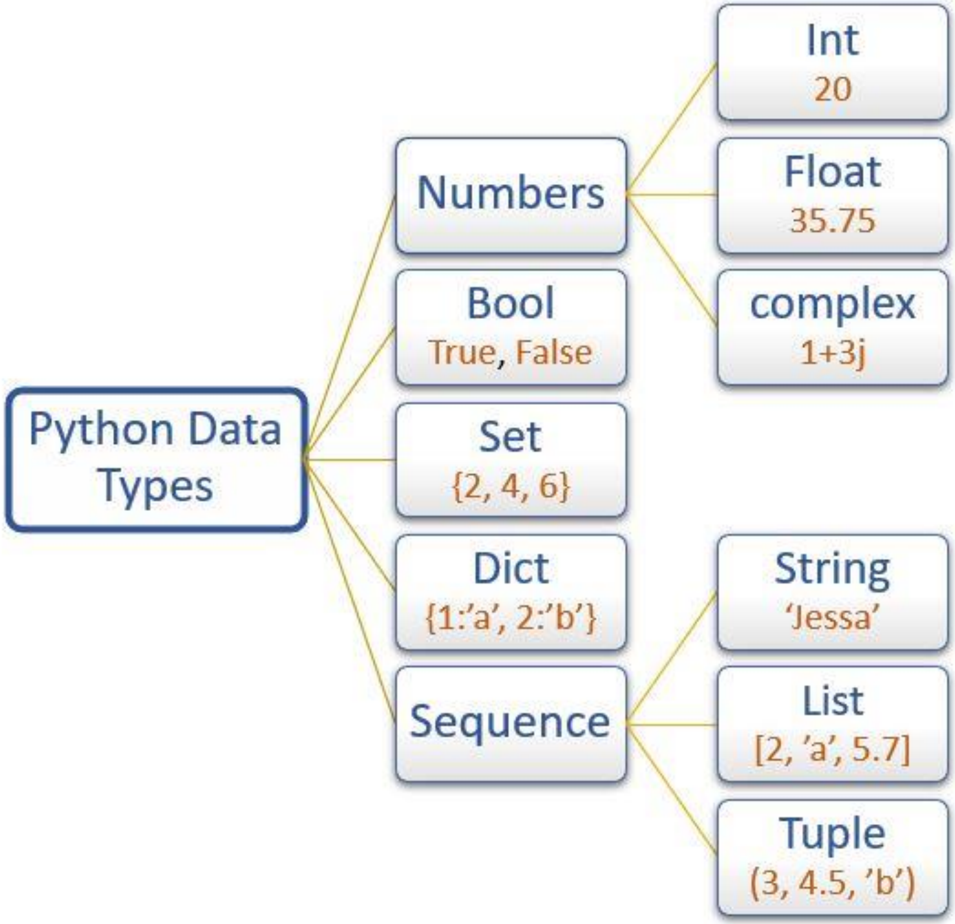
1. Visit the [Google Colab]() page, which will direct you to the [Google Colaboratory Welcome Page]().

http://colab.research.google.com/

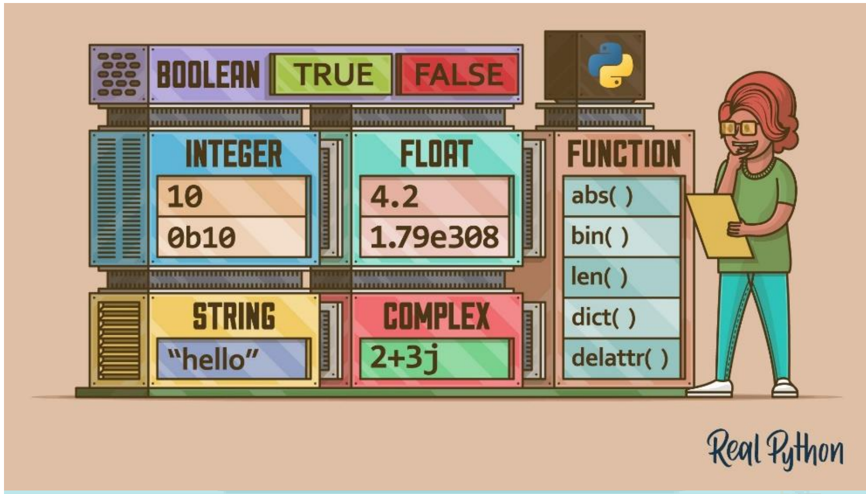
2. Click the **Sign in** button on the right top.

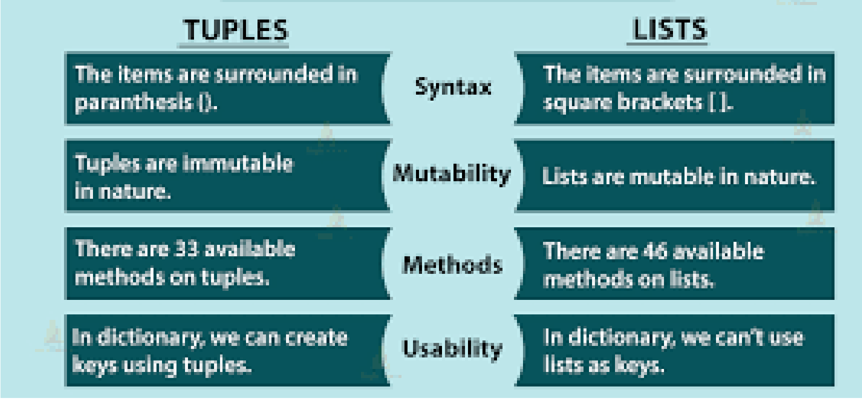
Working Variables in Python

Python Data Types



- Python Data Types
  - Numbers
    - Int 20
    - Float 35.75
    - complex 1+3j
  - Bool True, False
  - Set {2, 4, 6}
  - Dict {1:'a', 2:'b'}
  - Sequence
    - String 'Jessa'
    - List [2, 'a', 5.7]
    - Tuple (3, 4.5, 'b')

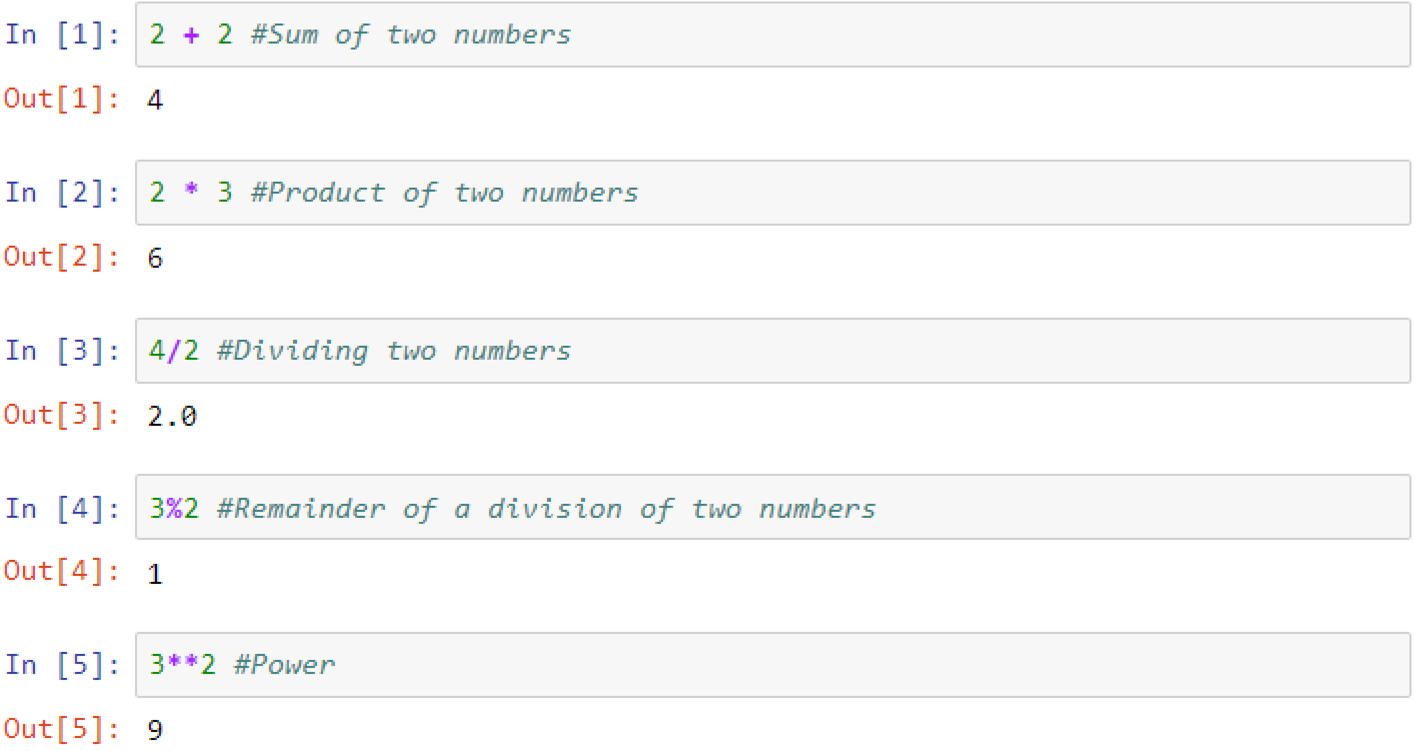




TUPLES		LISTS
The items are surrounded in paranthesis ().	Syntax	The items are surrounded in square brackets [ ].
Tuples are immutable in nature.	Mutability	Lists are mutable in nature.
There are 33 available methods on tuples.	Methods	There are 46 available methods on lists.
In dictionary, we can create keys using tuples.	Usability	In dictionary, we can't use lists as keys.

Types of Variables in Python

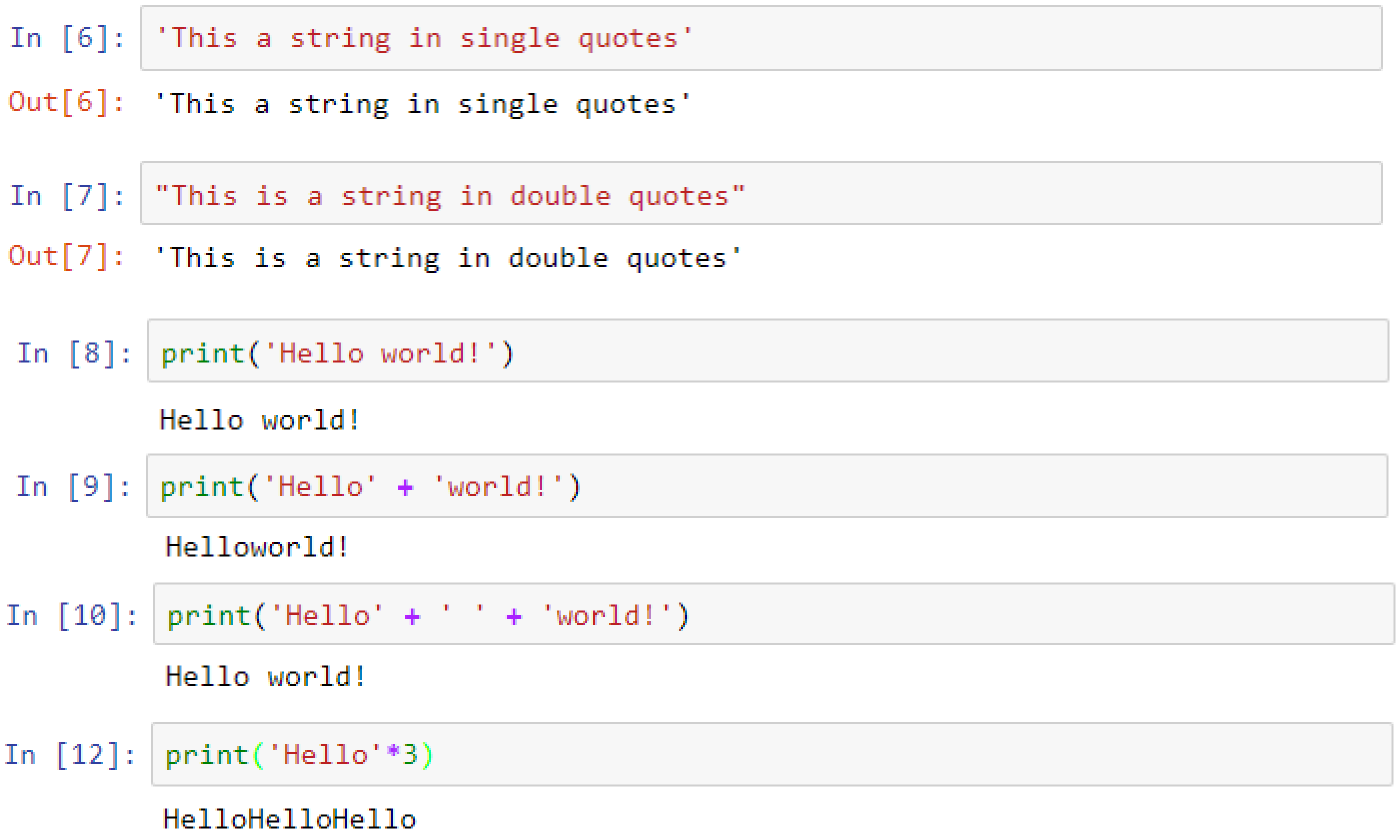
- **Literals - Numbers**



```
In [1]: 2 + 2 #Sum of two numbers

Out[1]: 4

In [2]: 2 * 3 #Product of two numbers

Out[2]: 6

In [3]: 4/2 #Dividing two numbers

Out[3]: 2.0

In [4]: 3%2 #Remainder of a division of two numbers

Out[4]: 1

In [5]: 3**2 #Power

Out[5]: 9
```

Types of Variables in Python

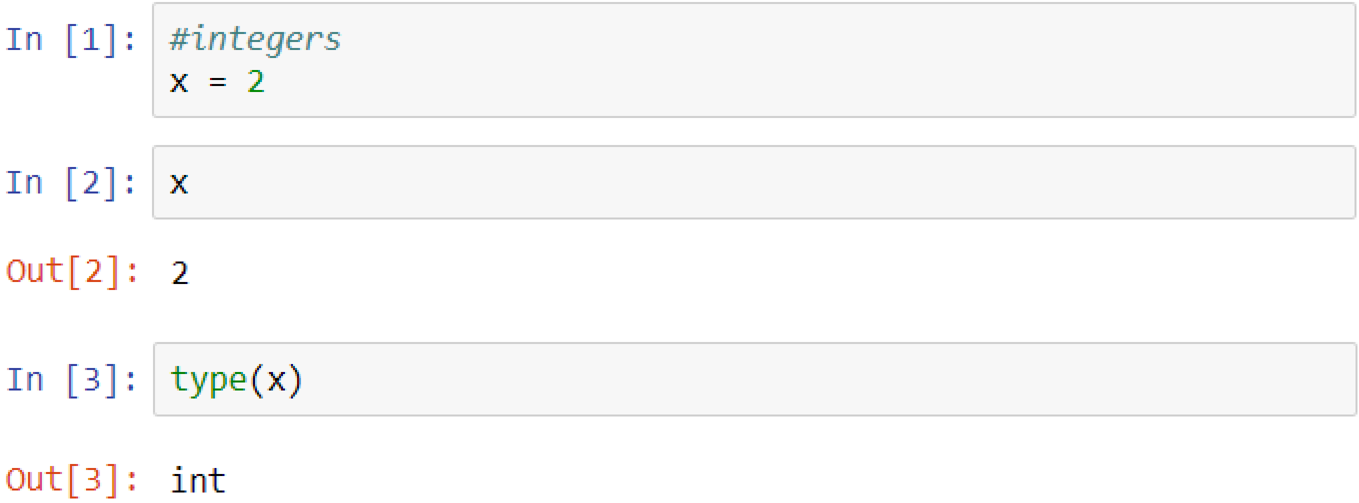
- **Literals - Strings**



```
In [6]:  'This a string in single quotes'
Out[6]:  'This a string in single quotes'

In [7]:  "This is a string in double quotes"
Out[7]:  'This is a string in double quotes'

In [8]:  print('Hello world!')

         Hello world!

In [9]:  print('Hello' + 'world!')

         Helloworld!

In [10]: print('Hello' + ' ' + 'world!')

         Hello world!

In [12]: print('Hello'*3)

         HelloHelloHello
```

Types of Variables in Python

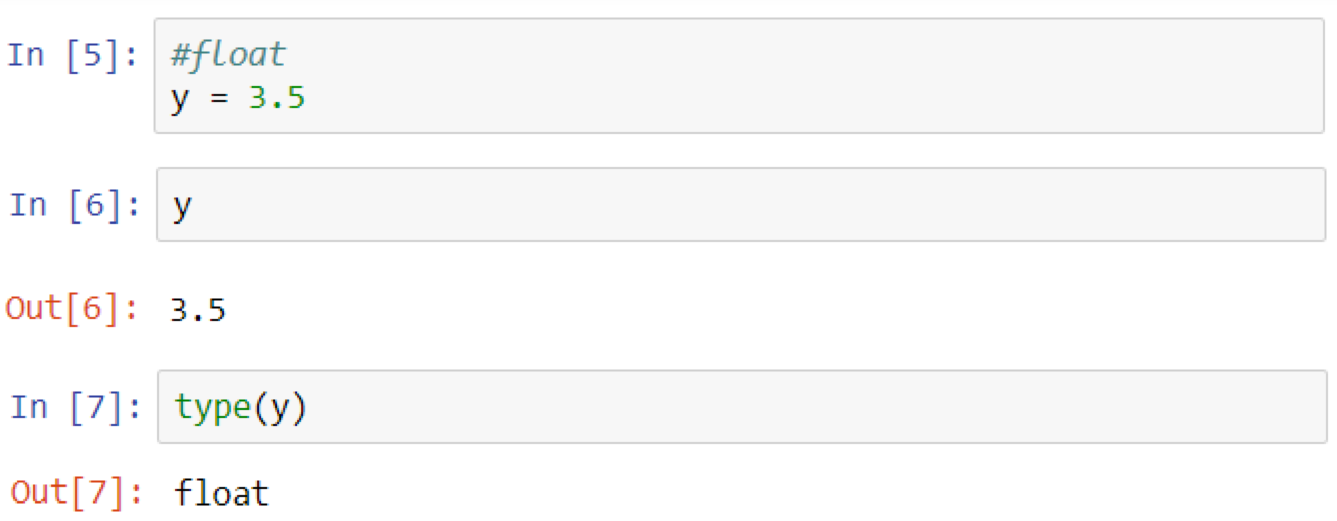
- **Numbers (int)**



```
In [1]: #integers
        x = 2

In [2]: x

Out[2]: 2

In [3]: type(x)

Out[3]: int
```

Types of Variables in Python

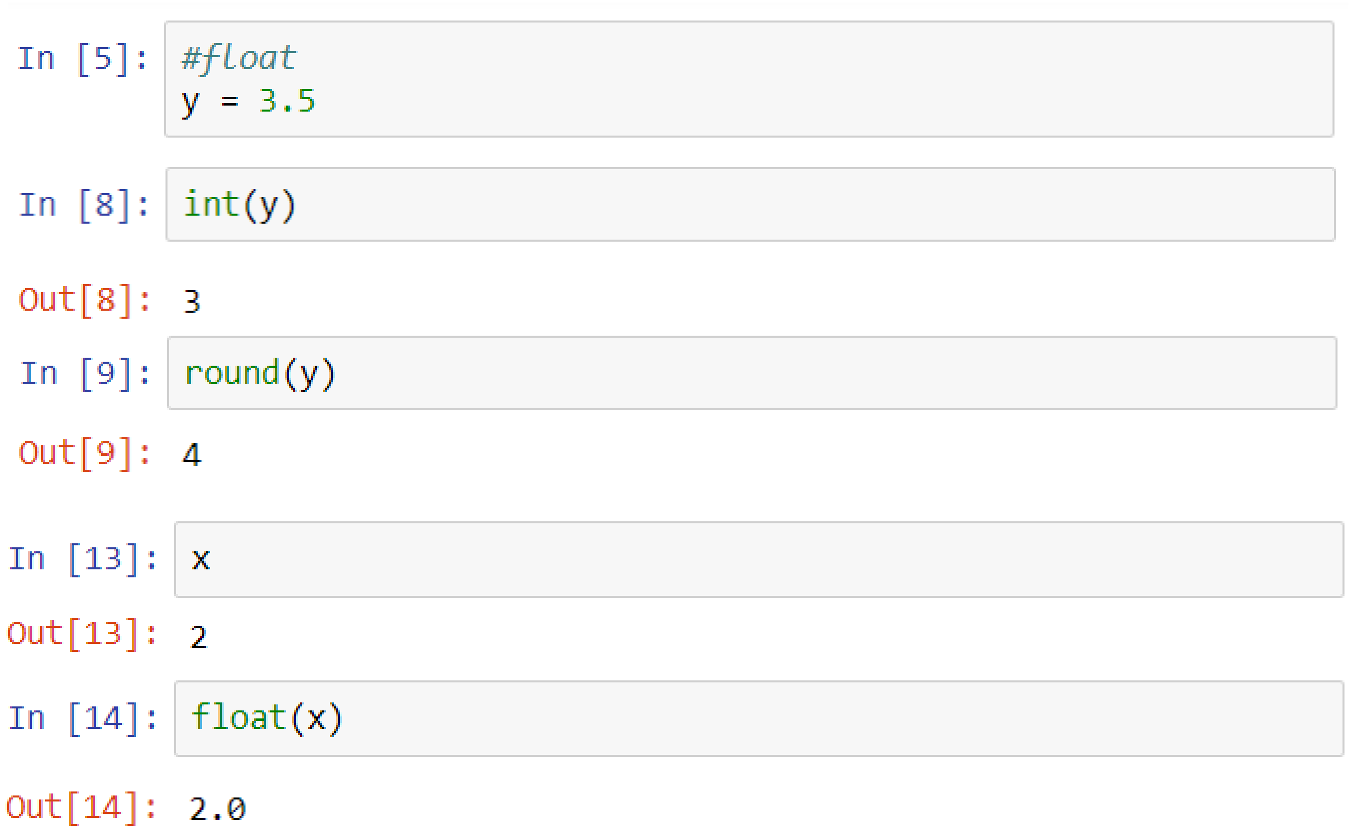
- **Numbers (float)**



```
In [5]: #float
        y = 3.5

In [6]: y

Out[6]: 3.5

In [7]: type(y)

Out[7]: float
```

Types of Variables in Python

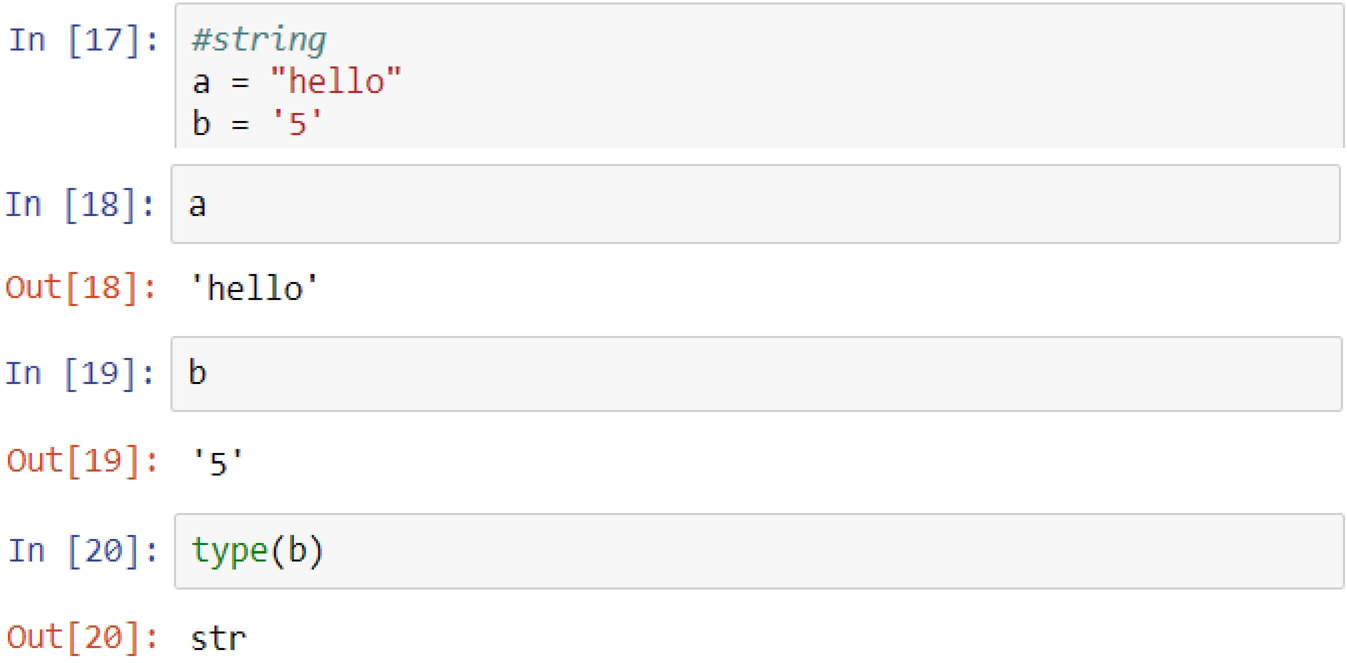
- **Numbers (float)**



```
In [5]:  #float
         y = 3.5

In [8]:  int(y)

Out[8]:  3

In [9]:  round(y)

Out[9]:  4

In [13]: x

Out[13]: 2

In [14]: float(x)

Out[14]: 2.0
```

Types of Variables in Python

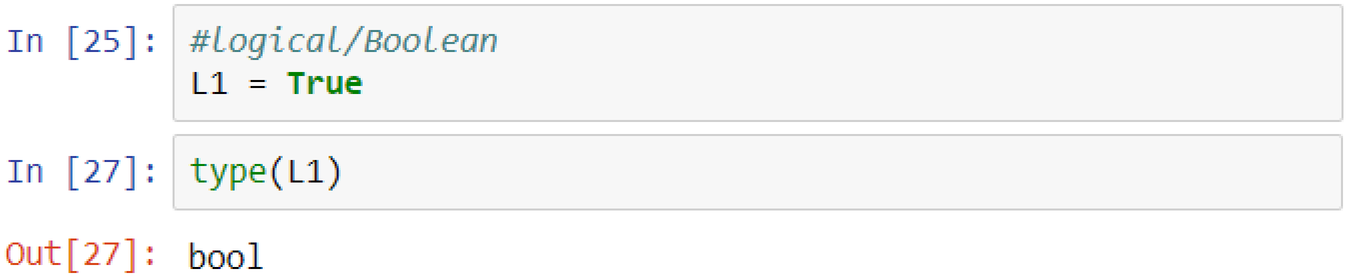
- **String (str)**



```
In [17]: #string
         a = "hello"
         b = '5'

In [18]: a

Out[18]: 'hello'

In [19]: b

Out[19]: '5'

In [20]: type(b)

Out[20]: str
```

Types of Variables in Python

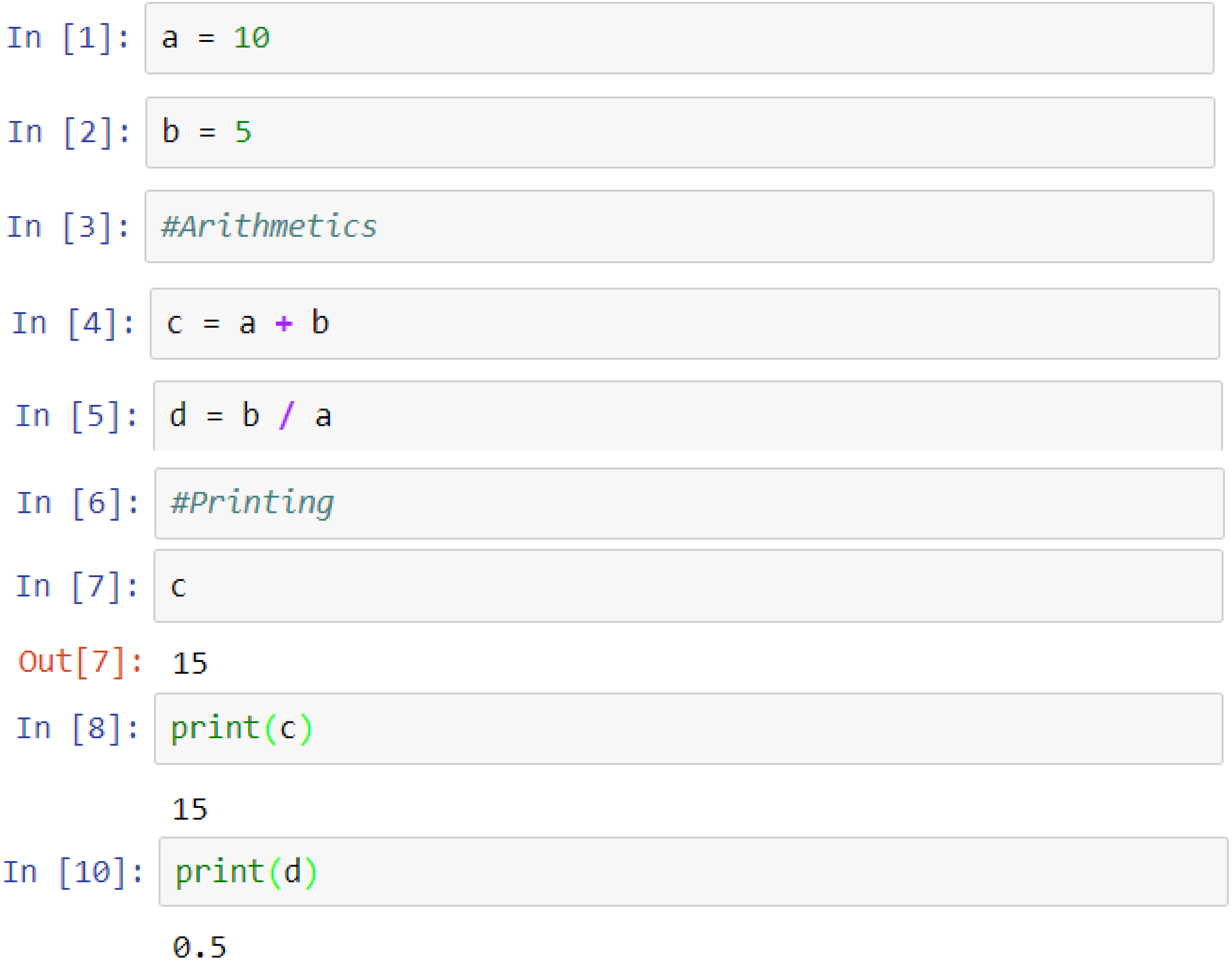
- **Logical (bool)**



```
In [25]:  #logical/Boolean
          L1 = True

In [27]:  type(L1)

Out[27]:  bool
```

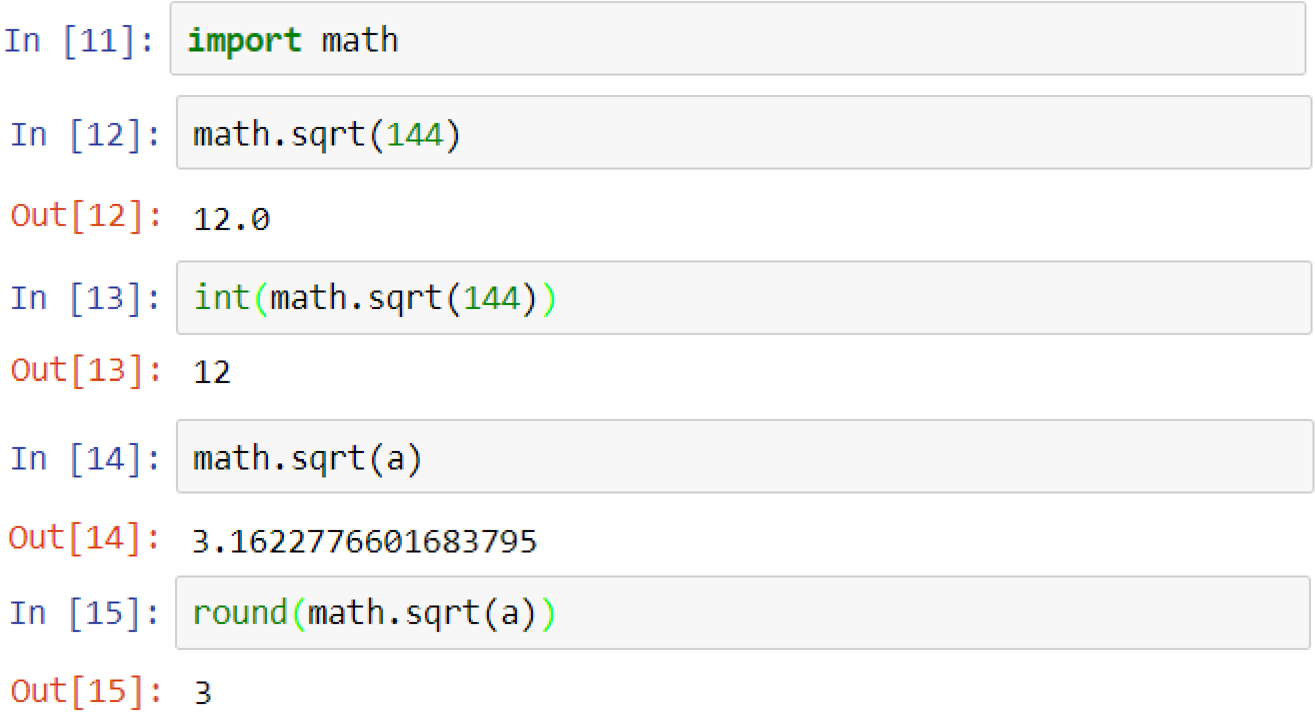
Using Variables in Python

- **Aritmetics**



```
In [1]: a = 10

In [2]: b = 5

In [3]: #Arithmetics

In [4]: c = a + b

In [5]: d = b / a

In [6]: #Printing

In [7]: c

Out[7]: 15

In [8]: print(c)

        15

In [10]: print(d)

         0.5
```

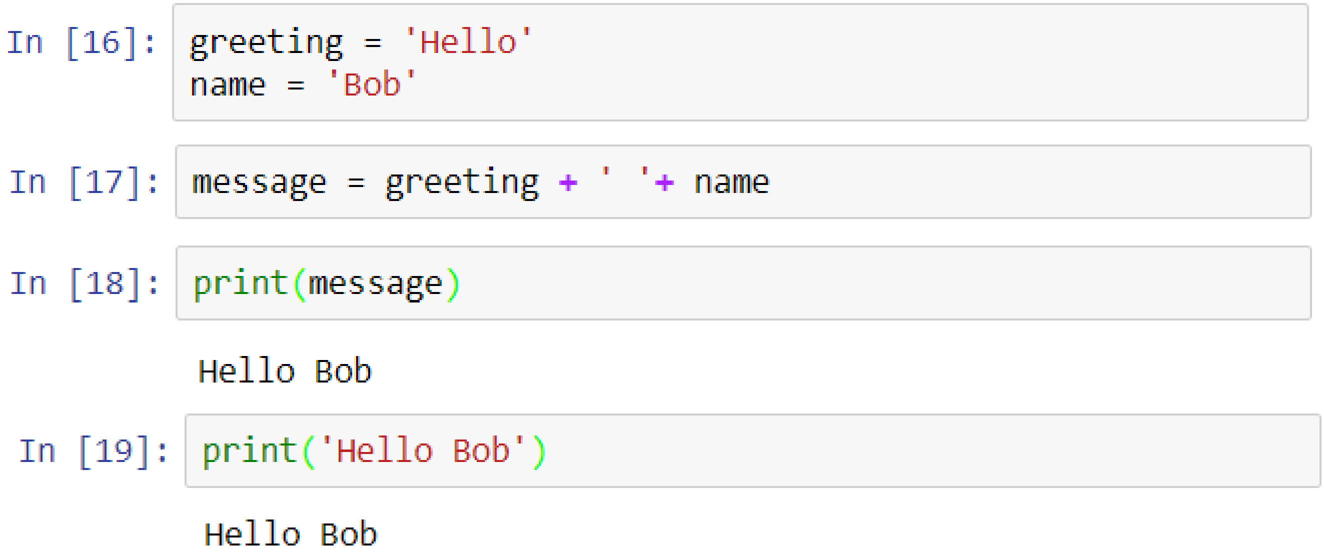
Using Variables in Python



```
In [11]: import math

In [12]: math.sqrt(144)

Out[12]: 12.0

In [13]: int(math.sqrt(144))

Out[13]: 12

In [14]: math.sqrt(a)

Out[14]: 3.1622776601683795

In [15]: round(math.sqrt(a))

Out[15]: 3
```

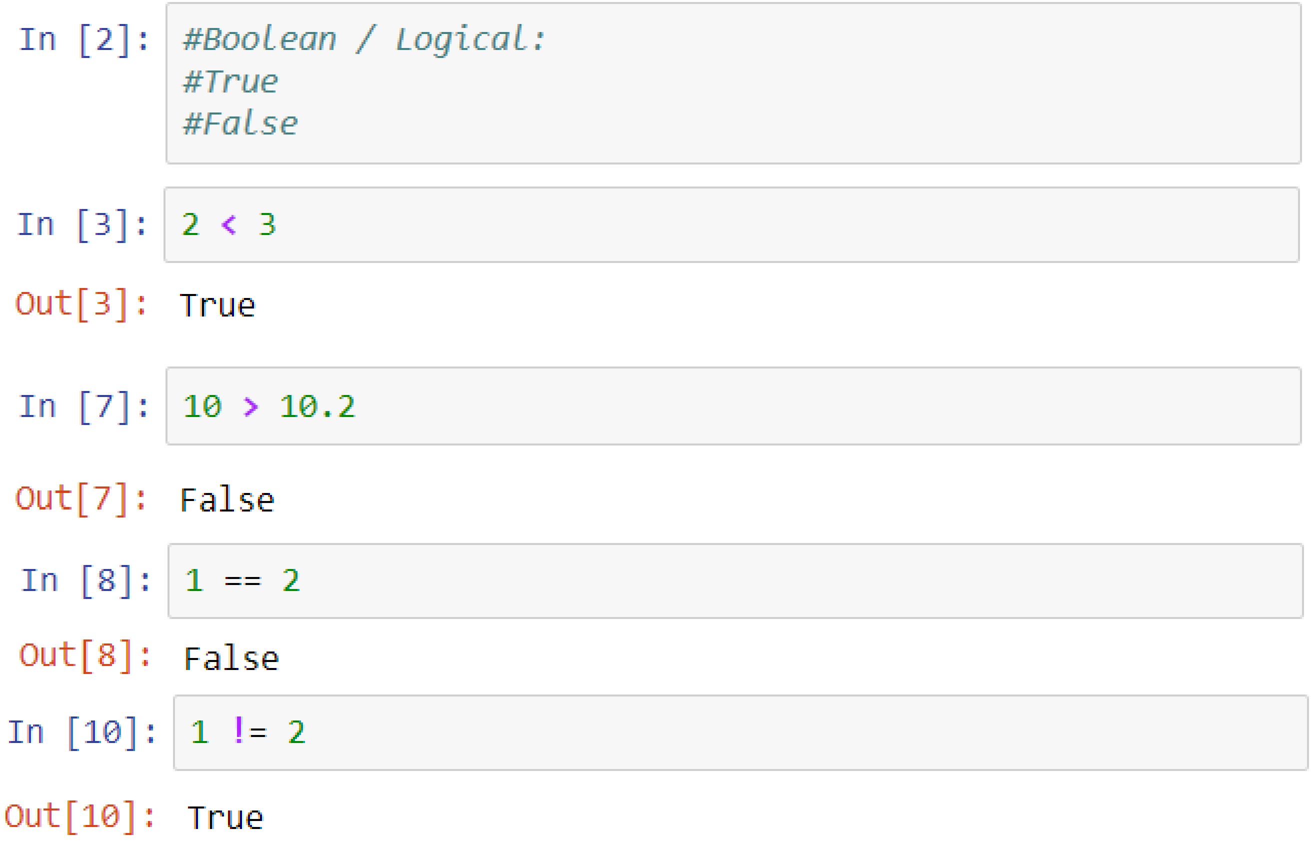
Using Variables in Python

- **String combination**



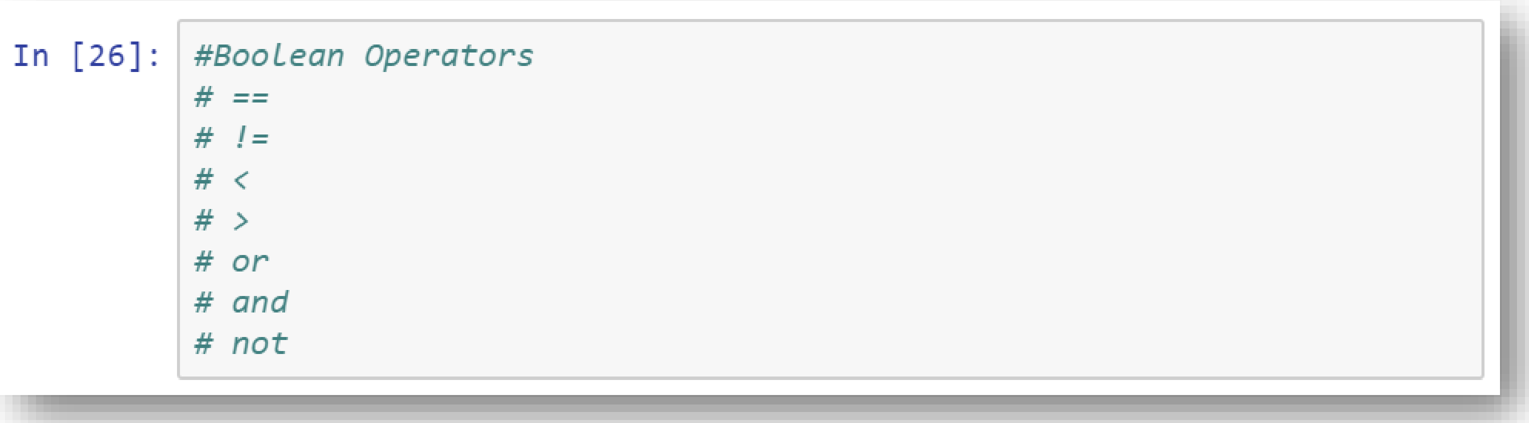
```
In [16]: greeting = 'Hello'
         name = 'Bob'

In [17]: message = greeting + ' '+ name

In [18]: print(message)

         Hello Bob

In [19]: print('Hello Bob')

         Hello Bob
```

Boolean Variables and Operators in Python

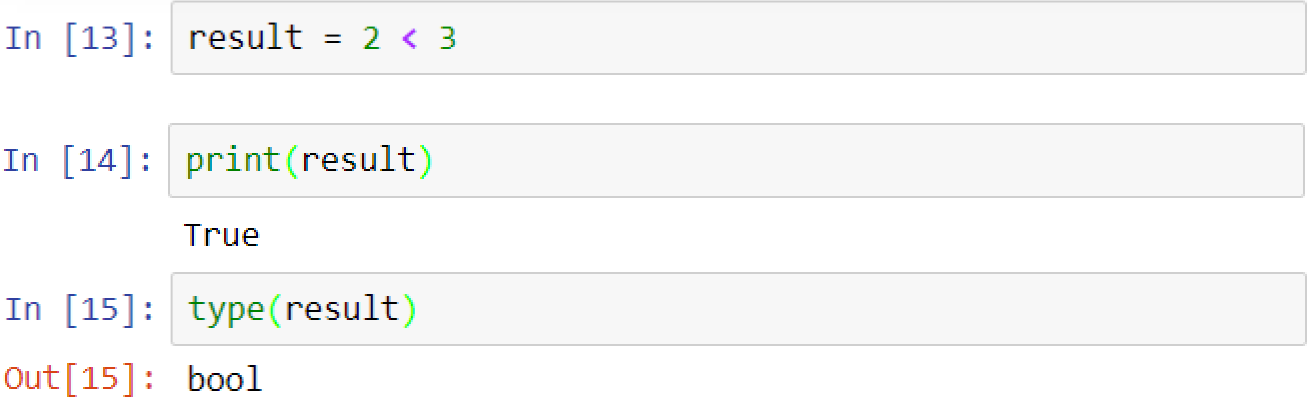


```
In [2]: #Boolean / Logical:
        #True
        #False

In [3]: 2 < 3

Out[3]: True

In [7]: 10 > 10.2

Out[7]: False

In [8]: 1 == 2

Out[8]: False

In [10]: 1 != 2

Out[10]: True
```

Boolean Variables and Operators in Python

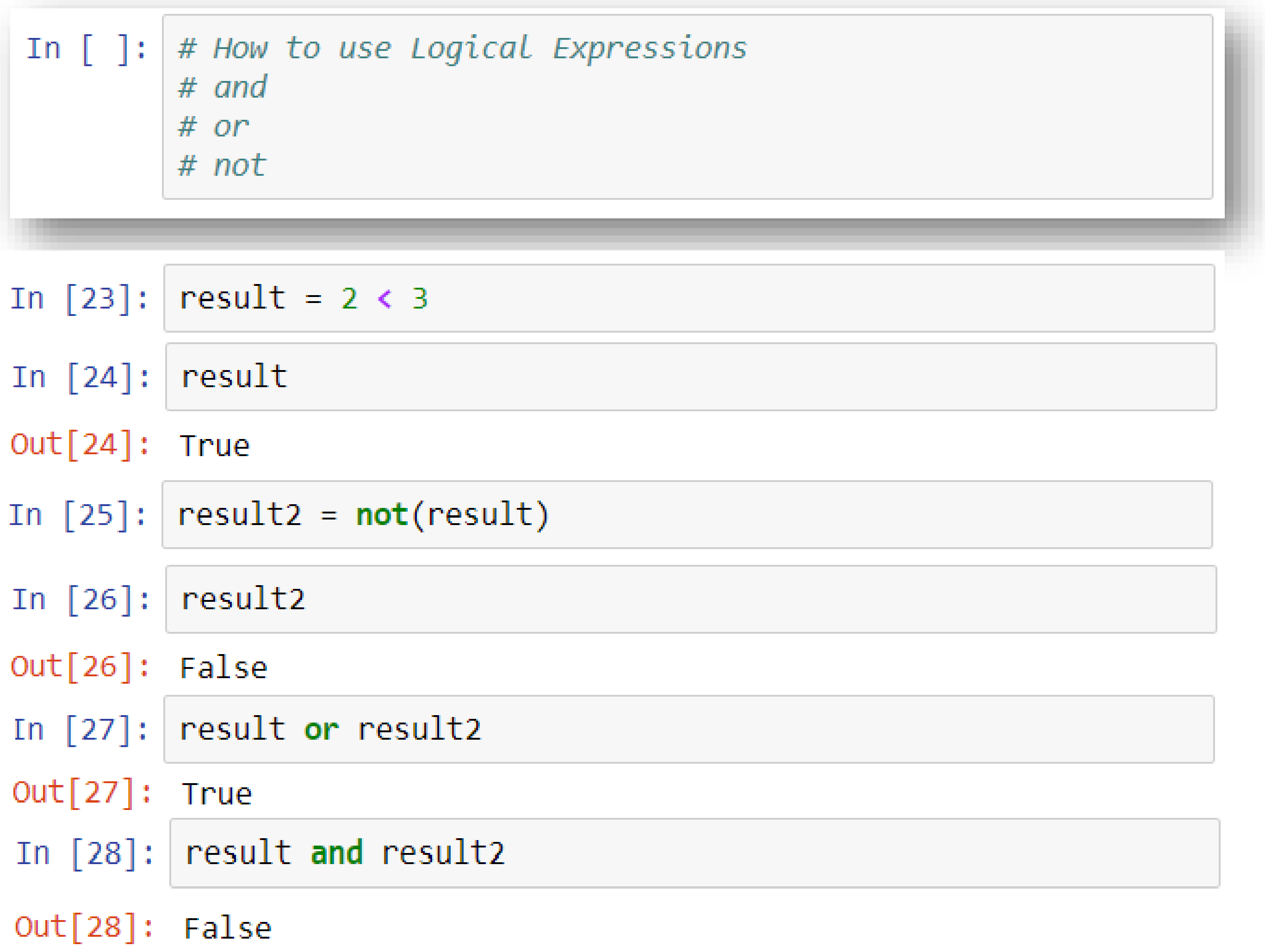


```
In [26]: #Boolean Operators
         # ==
         # !=
         # <
         # >
         # or
         # and
         # not
```



```
In [13]: result = 2 < 3

In [14]: print(result)

         True

In [15]: type(result)

Out[15]: bool
```

Boolean Variables and Operators in Python



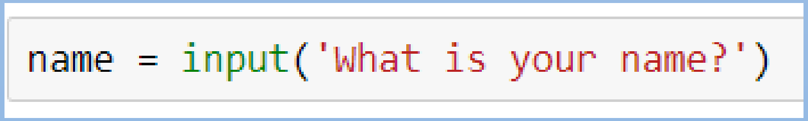
```
In [ ]: # How to use Logical Expressions
        # and
        # or
        # not

In [23]: result = 2 < 3

In [24]: result

Out[24]: True

In [25]: result2 = not(result)

In [26]: result2

Out[26]: False

In [27]: result or result2

Out[27]: True

In [28]: result and result2

Out[28]: False
```

Exercise 1

Create a program that asks the user for their name and age.

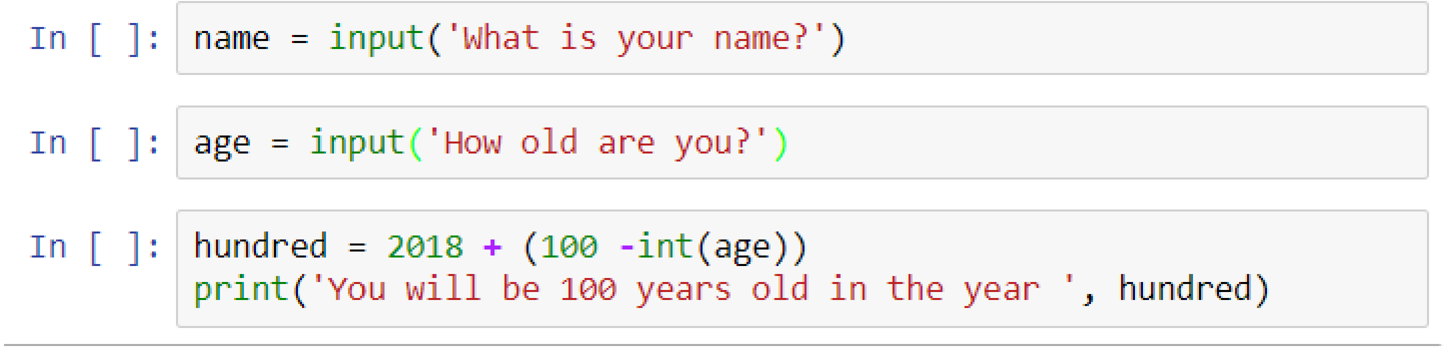
Print out a message addressed to them that tells them the year they will turn 100 years old.

*hint to get input from user:



```
name = input('What is your name?')
```

Exercise 1 – Solution (100)



```
name = input('What is your name?')
```

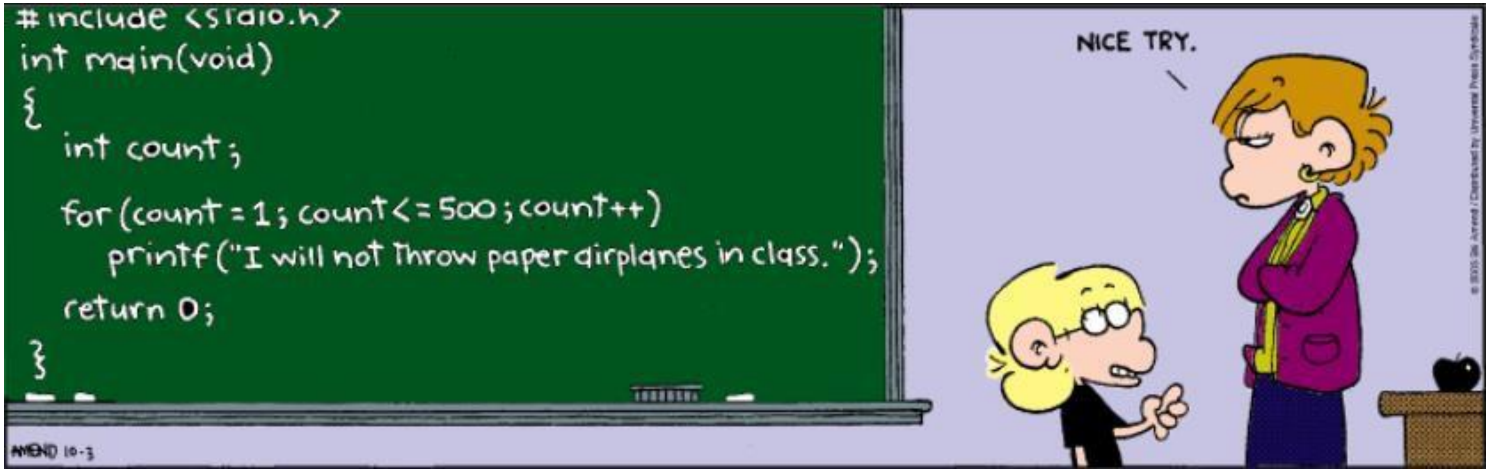
```
age = input('How old are you?')
```

```
hundred = 2018 + (100 -int(age))
print('You will be 100 years old in the year ', hundred)
```

Break

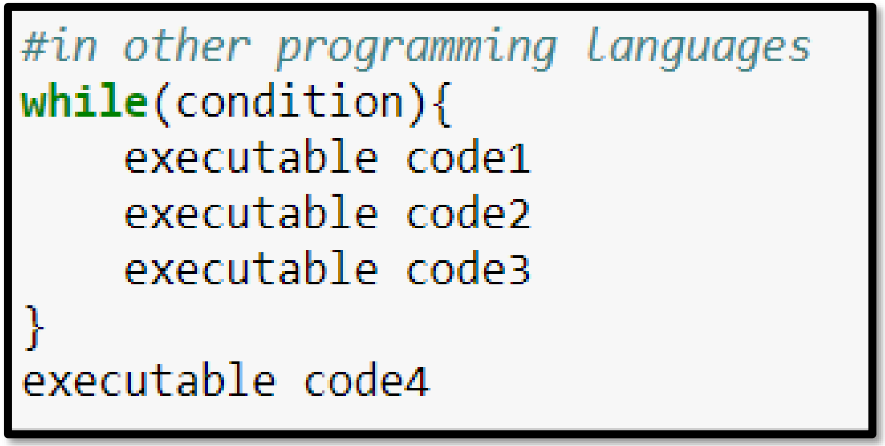
Flow Control in Python

What is Flow Control in Programming?

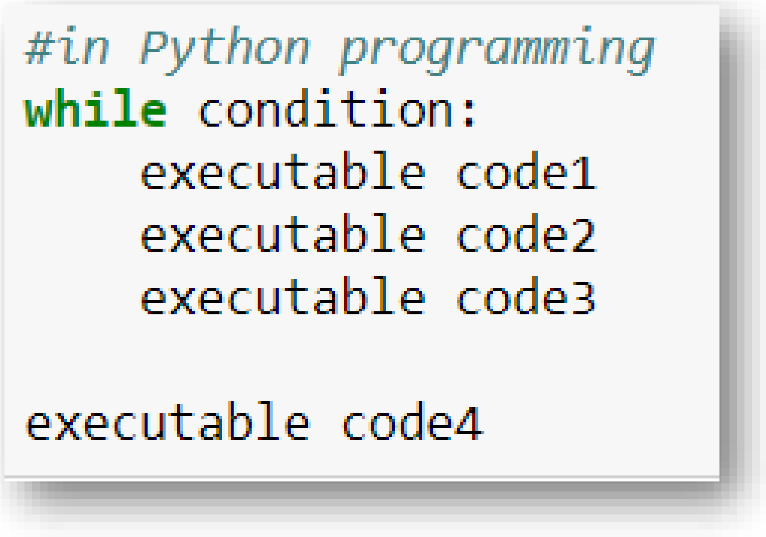


```
#include <stdio.h>
int main(void)
{
   int count;

   for (count = 1; count<=500; count++)
      printf("I will not Throw paper airplanes in class.");
   return 0;
}
```

The ‘while’ Loop – Indentations in Python



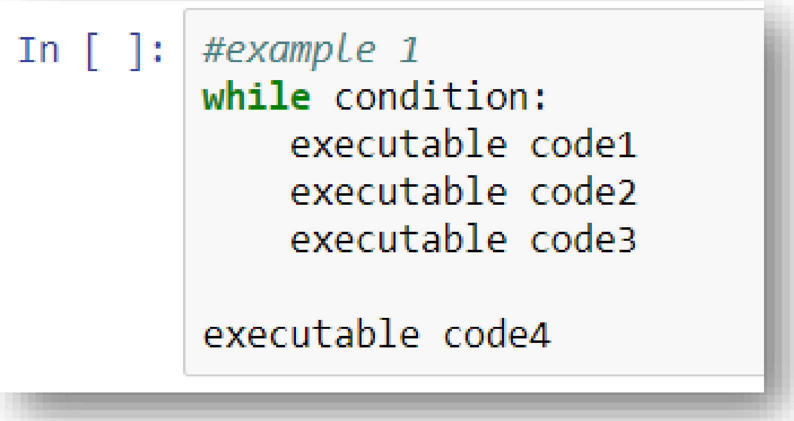
```
#in other programming languages
while(condition){
    executable code1
    executable code2
    executable code3
}
executable code4
```



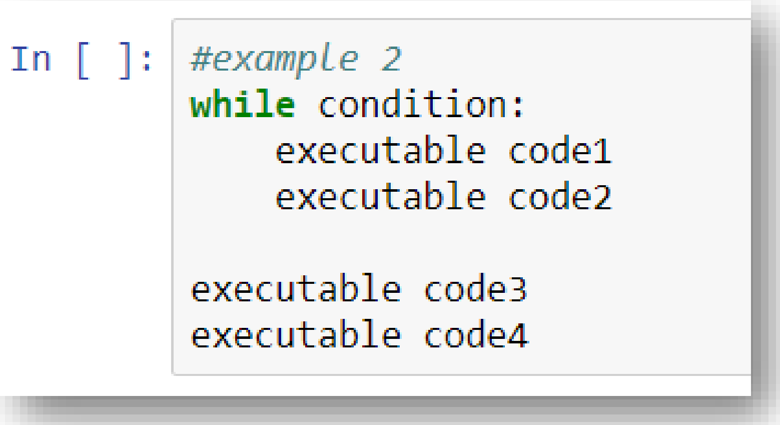
```
#in Python programming
while condition:
    executable code1
    executable code2
    executable code3

executable code4
```

Can you spot the differences?

The ‘while’ Loop – Indentations in Python



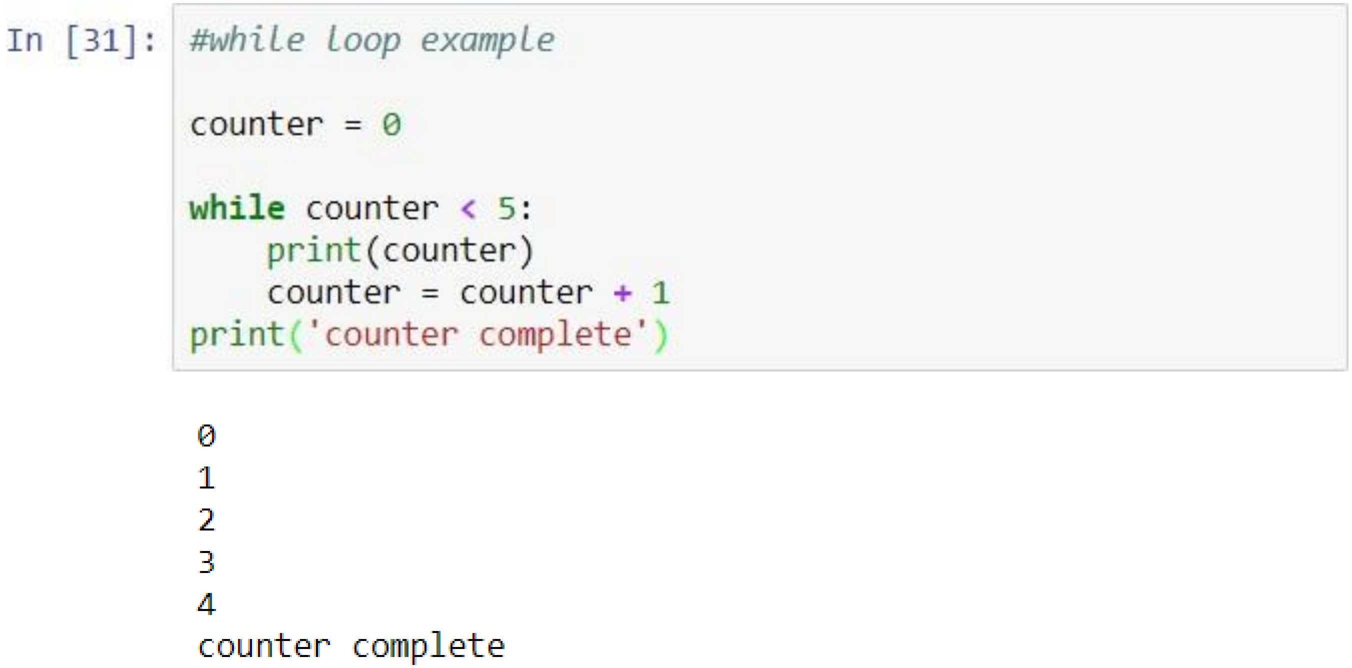
```
In [ ]: #example 1
        while condition:
            executable code1
            executable code2
            executable code3

        executable code4
```



```
In [ ]: #example 2
        while condition:
            executable code1
            executable code2

        executable code3
        executable code4
```

Is the ‘executable code3’ inside the ‘while’ loop?

Using the ‘while’ Loop

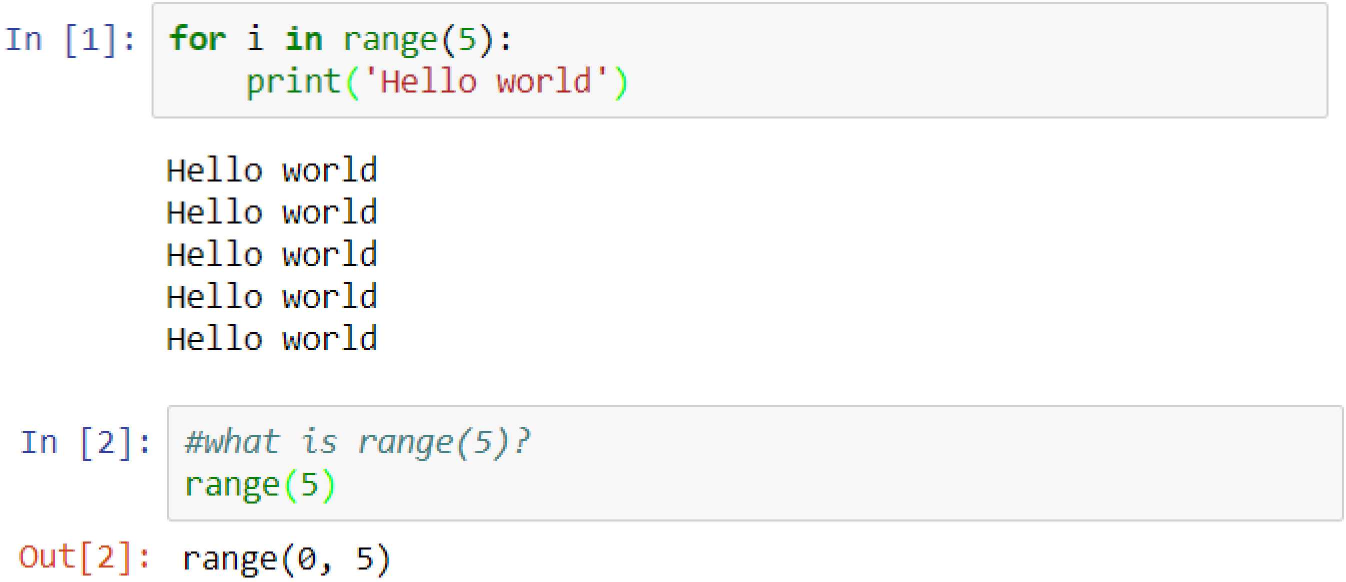


```
In [31]: #while loop example

         counter = 0

         while counter < 5:
             print(counter)
             counter = counter + 1
         print('counter complete')
```

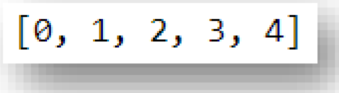
```
0
1
2
3
4
counter complete
```

Which lines are inside the ‘while’ loop?

The ‘for’ Loop

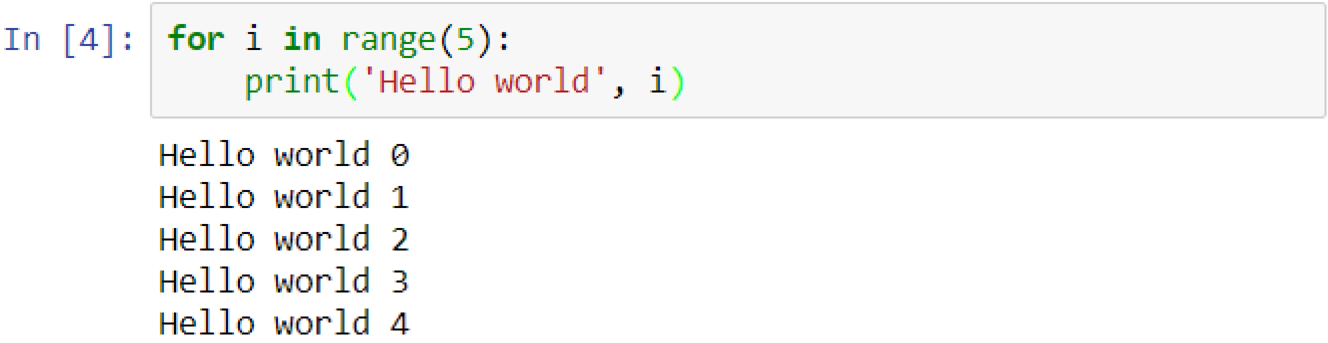


```
In [1]: for i in range(5):
            print('Hello world')

        Hello world
        Hello world
        Hello world
        Hello world
        Hello world

In [2]: #what is range(5)?
        range(5)

Out[2]: range(0, 5)
```



```
[0, 1, 2, 3, 4]
```

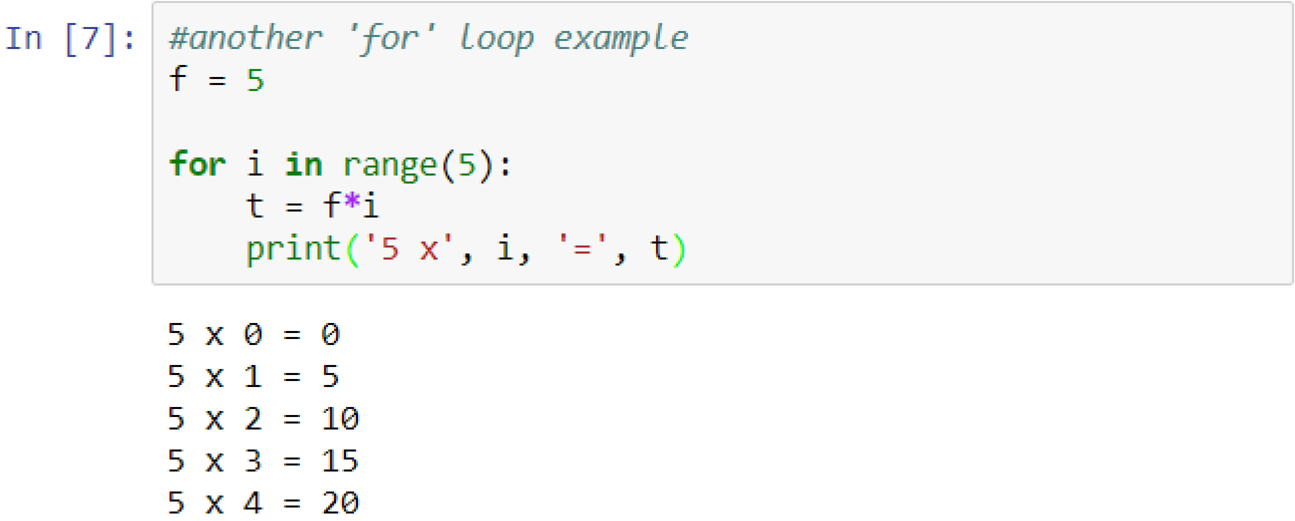
The ‘for’ Loop



In [4]:

```
for i in range(5):
    print('Hello world', i)
```

```
Hello world 0
Hello world 1
Hello world 2
Hello world 3
Hello world 4
```

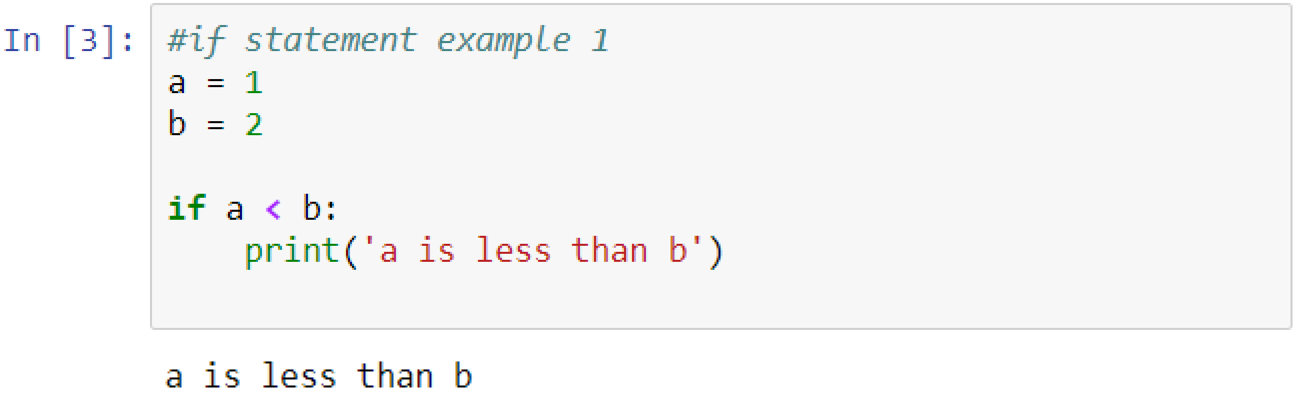


In [7]:

```
#another 'for' loop example
f = 5

for i in range(5):
    t = f*i
    print('5 x', i, '=', t)
```

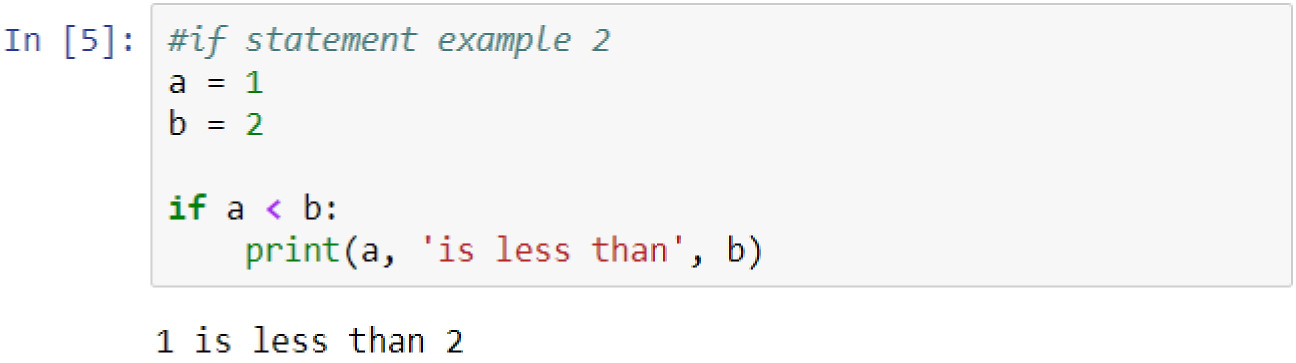
```
5 x 0 = 0
5 x 1 = 5
5 x 2 = 10
5 x 3 = 15
5 x 4 = 20
```

The ‘if’ Statements



```
In [3]: #if statement example 1
        a = 1
        b = 2

        if a < b:
            print('a is less than b')
```

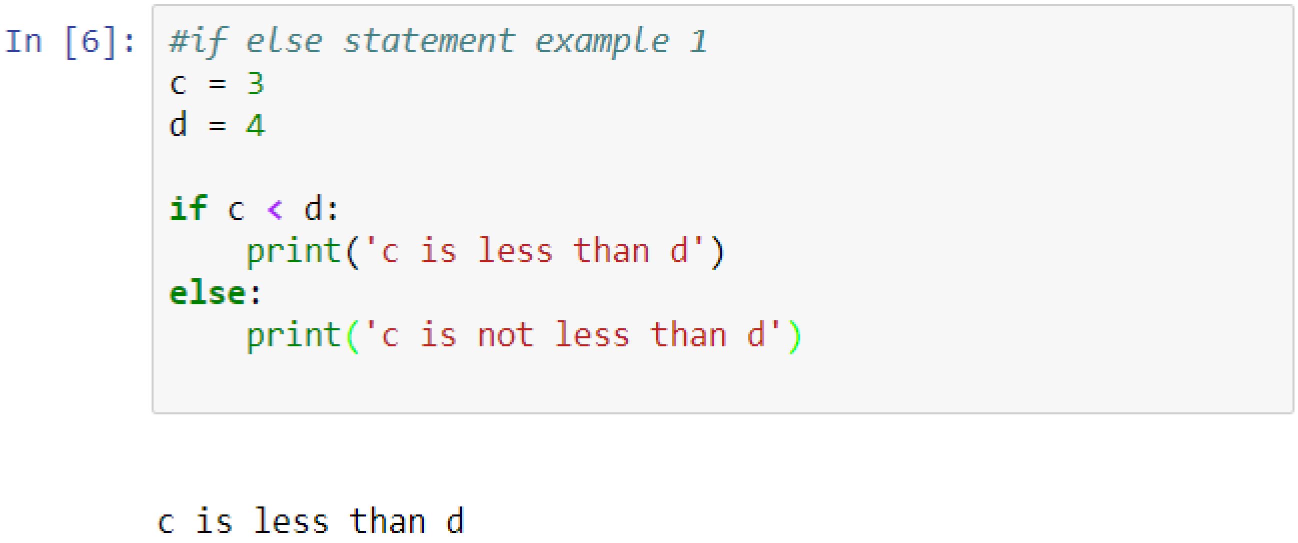
```
a is less than b
```



```
In [5]: #if statement example 2
        a = 1
        b = 2

        if a < b:
            print(a, 'is less than', b)
```

```
1 is less than 2
```

The ‘if - else’ Statements

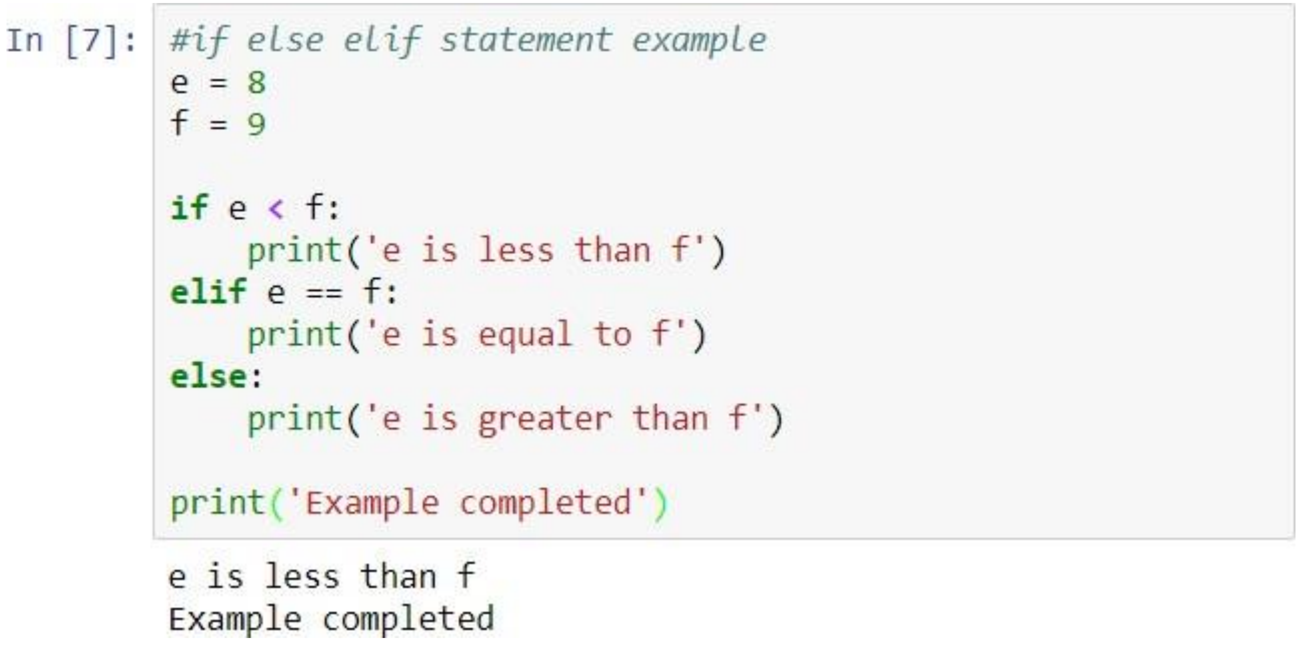


In [6]:

```python
#if else statement example 1
c = 3
d = 4

if c < d:
    print('c is less than d')
else:
    print('c is not less than d')
```

```
c is less than d
```

The ‘elif’ Statements – “less than”



```
In [7]: #if else elif statement example
        e = 8
        f = 9

        if e < f:
            print('e is less than f')
        elif e == f:
            print('e is equal to f')
        else:
            print('e is greater than f')

        print('Example completed')
```

```
e is less than f
Example completed
```

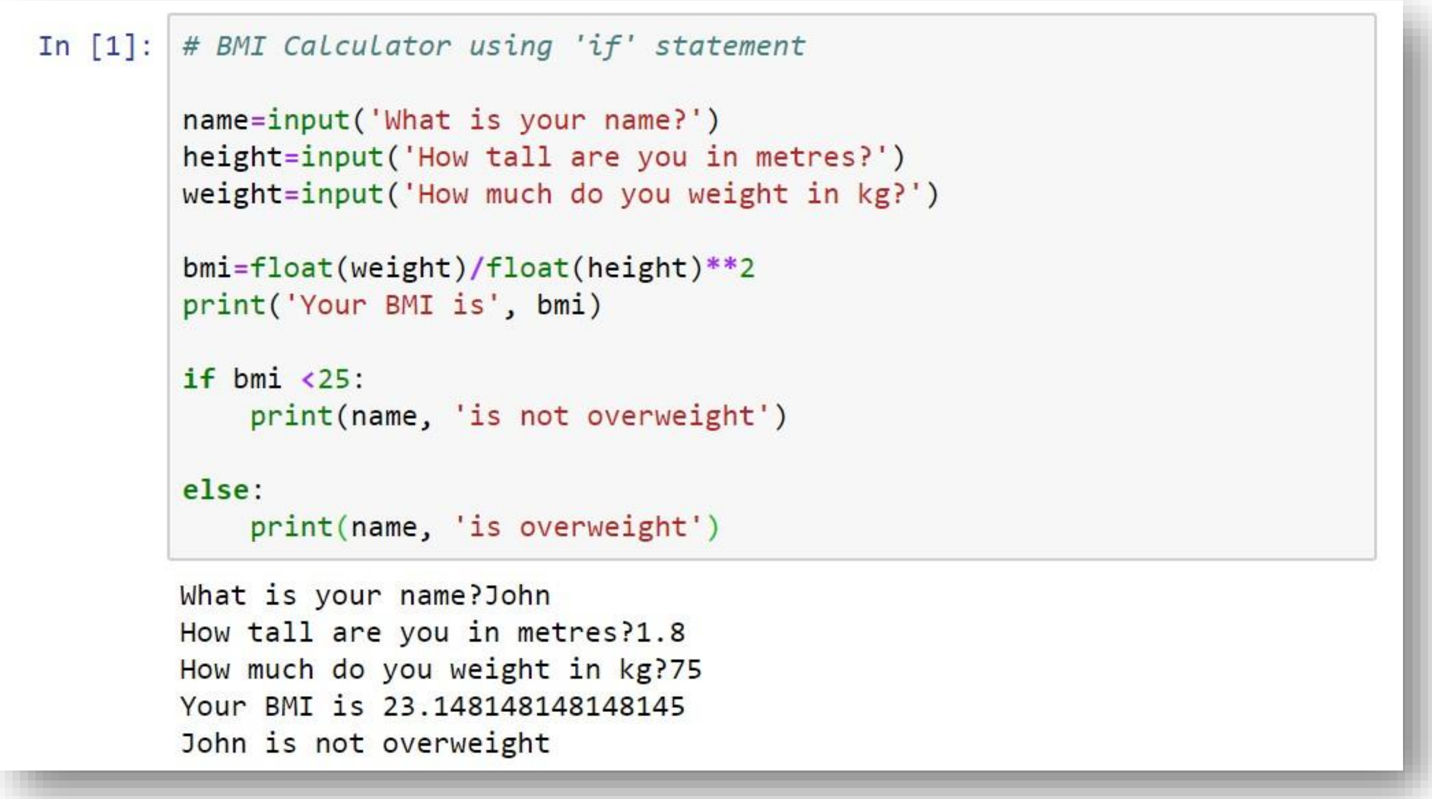
Exercise 2 – Build a BMI Calculator

Create a program that calculates a person’s BMI (Body Mass Index), based on the formula:

$$\text{BMI} = \text{Weight (kg)}/\text{Height(m)}^2$$

Then print out a message indicating whether if he/she is overweight (BMI > 25)

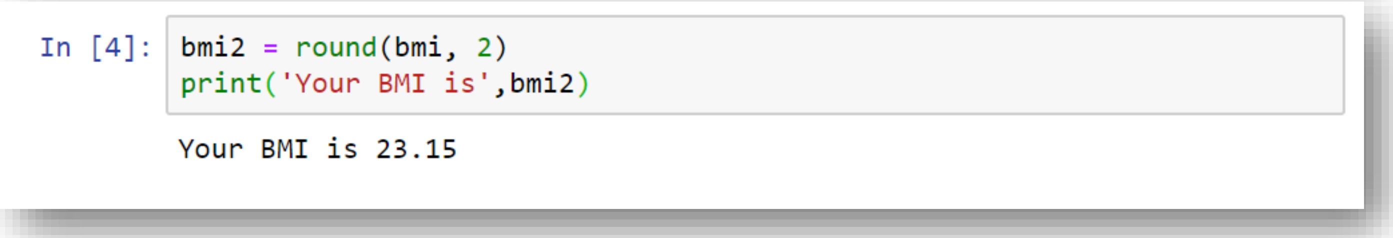
Exercise 2 – Solution (BMI Calculator)



In [1]:

```
# BMI Calculator using 'if' statement

name=input('What is your name?')
height=input('How tall are you in metres?')
weight=input('How much do you weight in kg?')

bmi=float(weight)/float(height)**2
print('Your BMI is', bmi)

if bmi <25:
    print(name, 'is not overweight')

else:
    print(name, 'is overweight')
```

```
What is your name?John
How tall are you in metres?1.8
How much do you weight in kg?75
Your BMI is 23.148148148148145
John is not overweight
```



In [4]:

```
bmi2 = round(bmi, 2)
print('Your BMI is',bmi2)
```

```
Your BMI is 23.15
```

Working with Python Collections

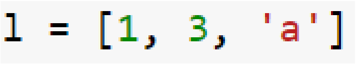
Python Collections – data type

- Previously we have learned data types:
  - Numbers
  - Strings
  - Boolean

- Collection of one or more data types:
  - Lists
  - Tuples
  - Dictionaries

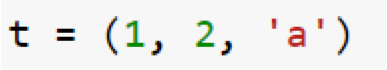
Python Collections

- Lists – uses square brackets



```
l = [1, 3, 'a']
```

- Tuples – uses parentheses

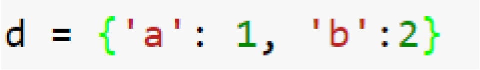


```
t = (1, 2, 'a')
```

  are immutable

  can be faster to execute than lists

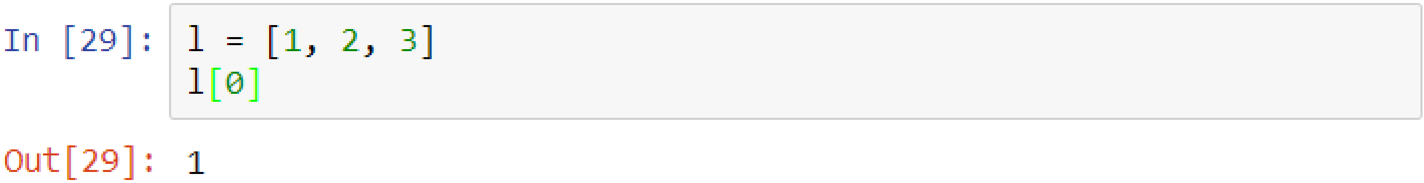
- Dictionaries – uses curly brackets



```
d = {'a': 1, 'b':2}
```

Lists in Python

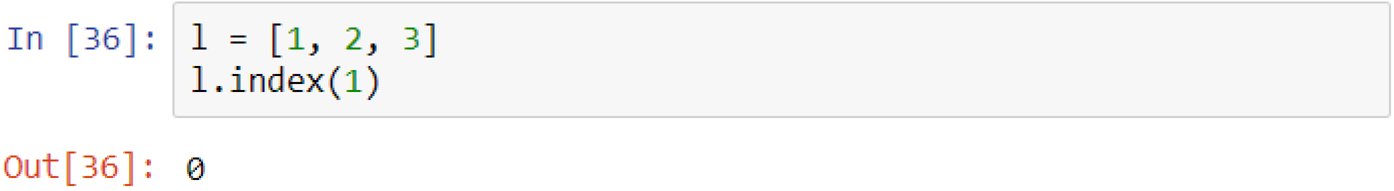
- Is a ‘list’ of values
- Collection of one or more data types (numbers, strings, Boolean)
- Lists starts at index number 0



```
In [29]: l = [1, 2, 3]
         l[0]

Out[29]: 1
```

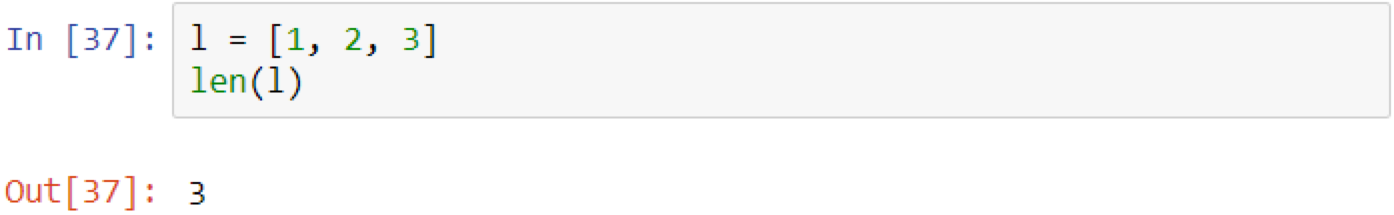
Lists in Python – index and len (length)

- You can find out the index of an element in a list:



```
In [36]: l = [1, 2, 3]
         l.index(1)

Out[36]: 0
```

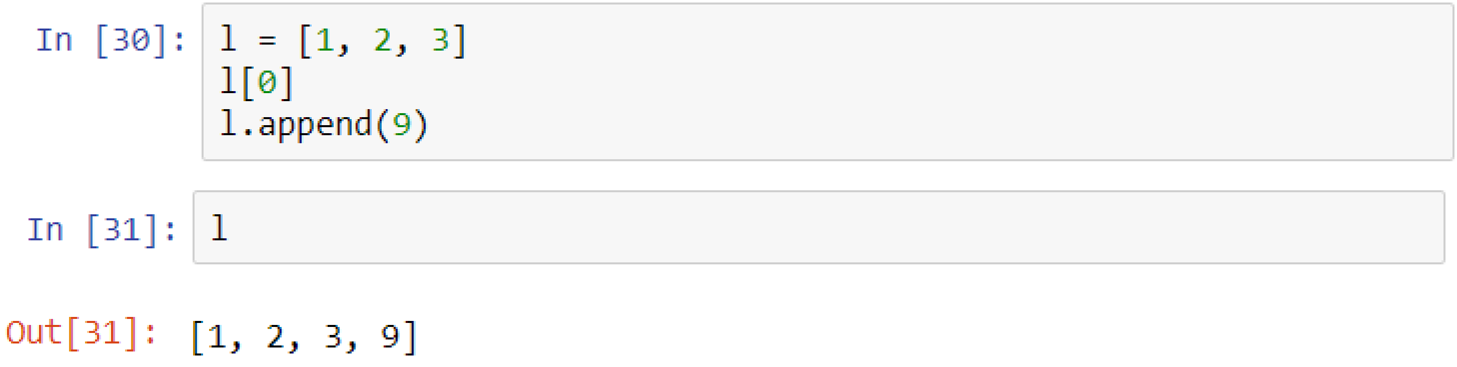
- You can also find out the length of a list:



```
In [37]: l = [1, 2, 3]
         len(l)

Out[37]: 3
```

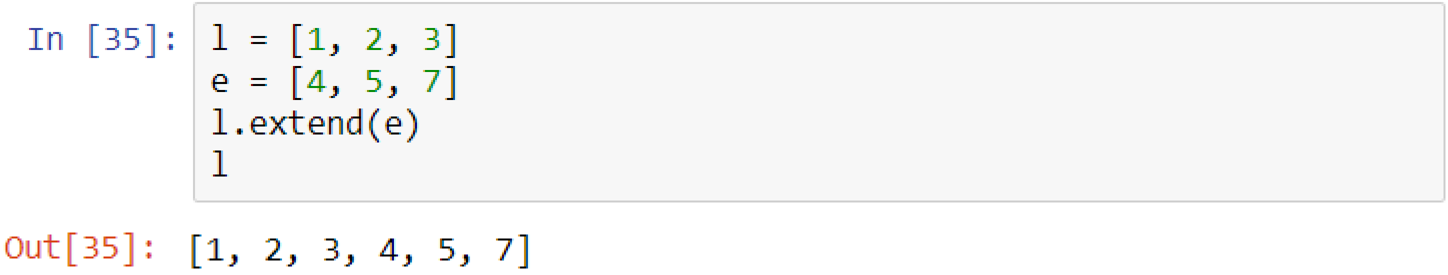
Lists in Python – append and extend

- You can append() an element to the end of a list:



```
In [30]: l = [1, 2, 3]
         l[0]
         l.append(9)

In [31]: l

Out[31]: [1, 2, 3, 9]
```

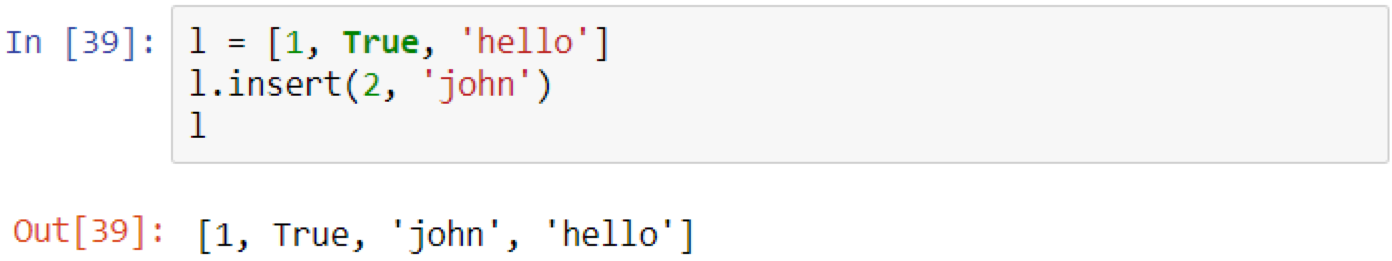
- You can also extend() a list by adding another list to its end:



```
In [35]: l = [1, 2, 3]
         e = [4, 5, 7]
         l.extend(e)
         l

Out[35]: [1, 2, 3, 4, 5, 7]
```

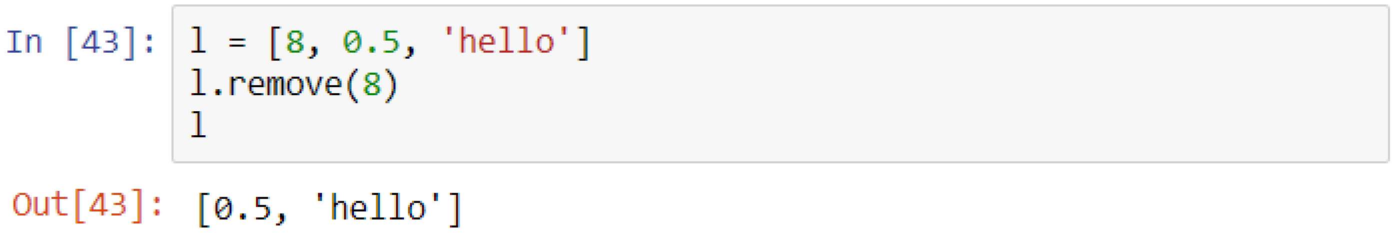
Lists in Python – insert and remove

- You can insert() an element to a list:



```
In [39]: l = [1, True, 'hello']
         l.insert(2, 'john')
         l

Out[39]: [1, True, 'john', 'hello']
```

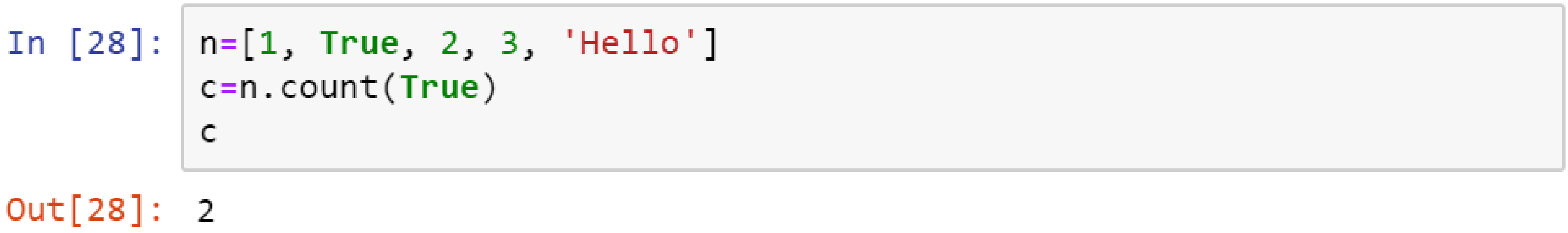
- You can also remove() an object from a list:



```
In [43]: l = [8, 0.5, 'hello']
         l.remove(8)
         l

Out[43]: [0.5, 'hello']
```

Lists in Python – True vs 1

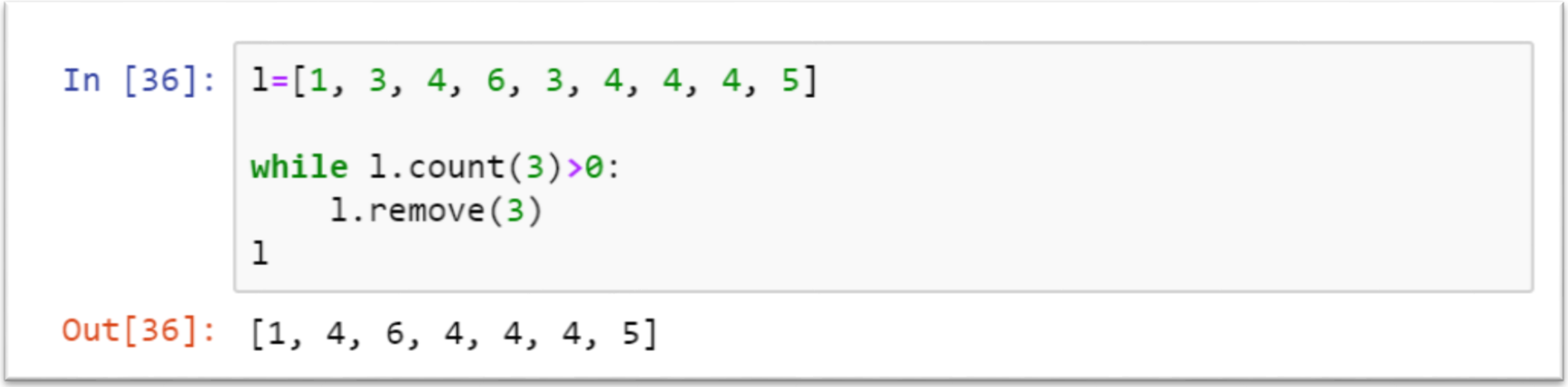
- Python recognizes True = 1



```
In [28]: n=[1, True, 2, 3, 'Hello']
         c=n.count(True)
         c

Out[28]: 2
```

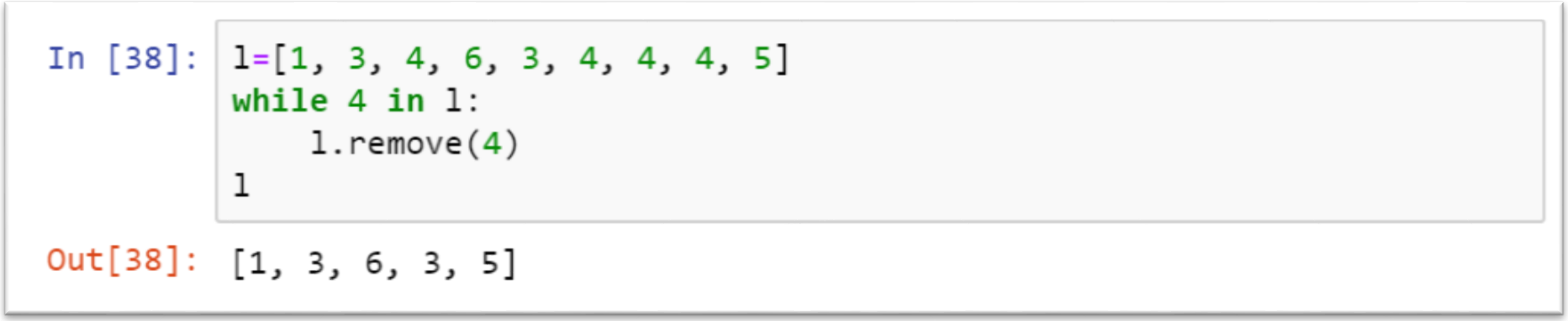
Lists in Python – remove()

- How to remove() all occurrences of the same values from a list:



```
In [36]: l=[1, 3, 4, 6, 3, 4, 4, 4, 5]

         while l.count(3)>0:
             l.remove(3)
         l

Out[36]: [1, 4, 6, 4, 4, 4, 5]
```

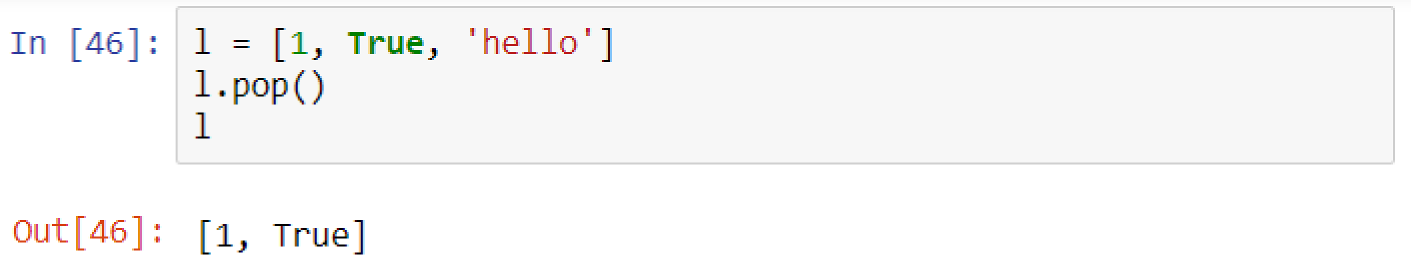
Another way of using ‘in’:



```
In [38]: l=[1, 3, 4, 6, 3, 4, 4, 4, 5]
         while 4 in l:
             l.remove(4)
         l

Out[38]: [1, 3, 6, 3, 5]
```

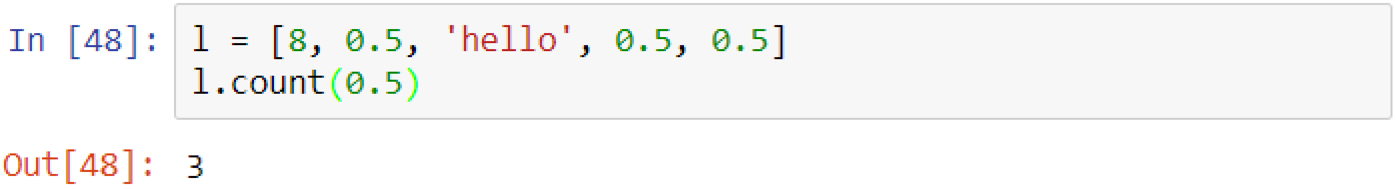
Lists in Python – pop and count

- You can remove the last object from a list using pop():



```
In [46]: l = [1, True, 'hello']
         l.pop()
         l

Out[46]: [1, True]
```

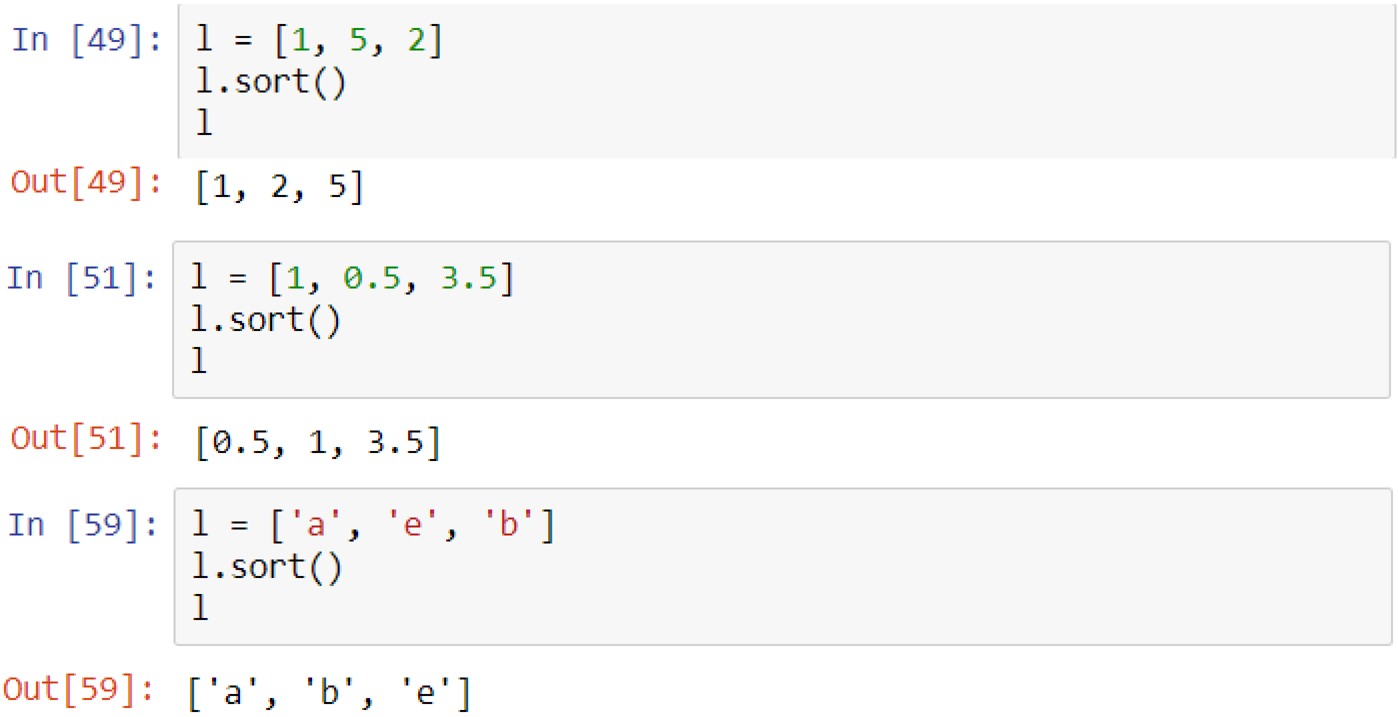
- You can also count() the occurrence of an object:



```
In [48]: l = [8, 0.5, 'hello', 0.5, 0.5]
         l.count(0.5)

Out[48]: 3
```

Lists in Python – sort (dos)

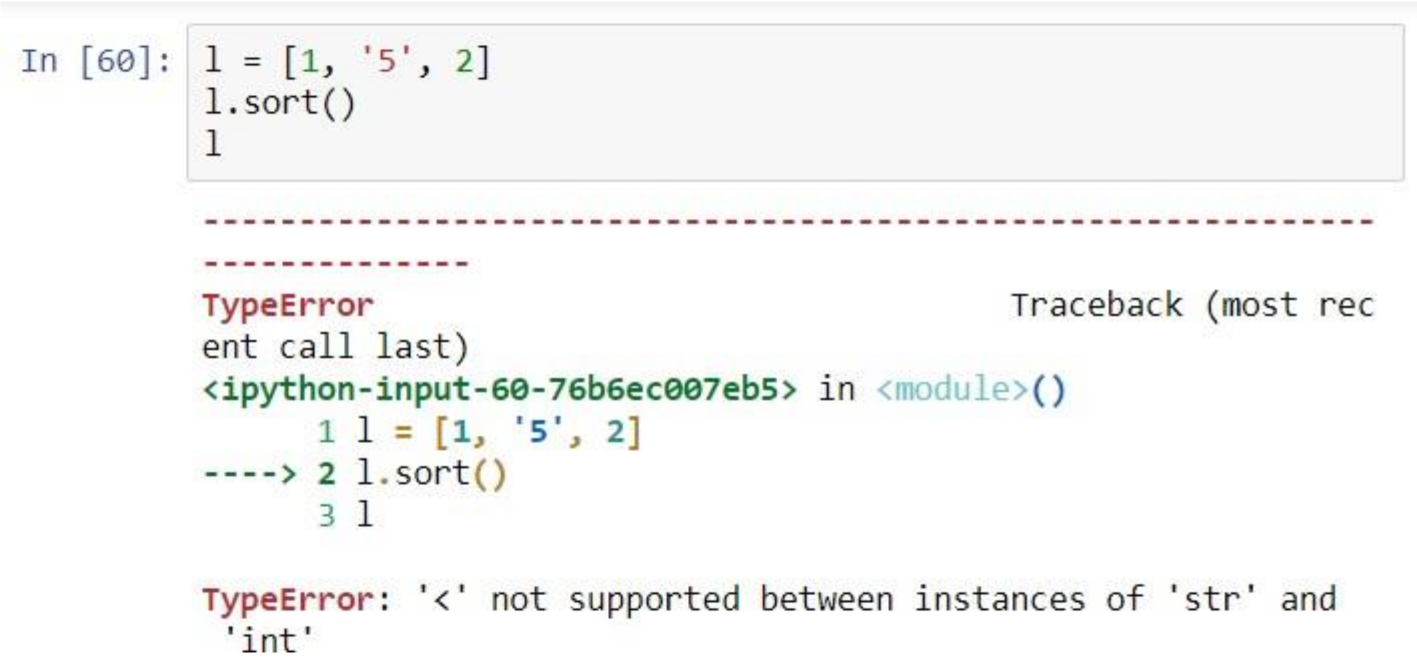
- You can sort() objects from a list:



```
In [49]: l = [1, 5, 2]
         l.sort()
         l

Out[49]: [1, 2, 5]

In [51]: l = [1, 0.5, 3.5]
         l.sort()
         l

Out[51]: [0.5, 1, 3.5]

In [59]: l = ['a', 'e', 'b']
         l.sort()
         l

Out[59]: ['a', 'b', 'e']
```

Lists in Python – sort (don’ts)

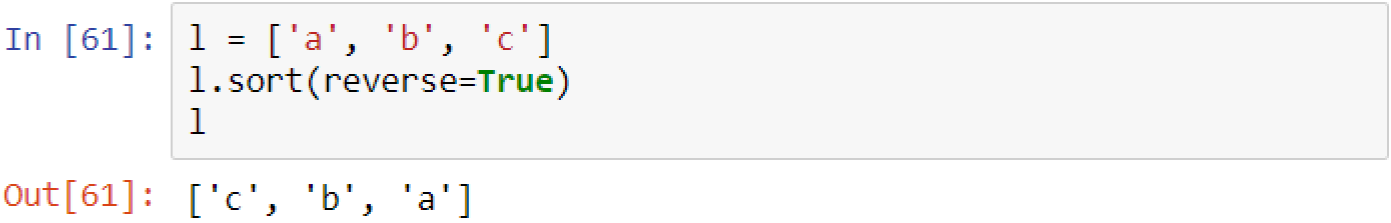
- You <u>cannot</u> sort() cross data type objects from a list:



```
In [60]: l = [1, '5', 2]
         l.sort()
         l

         ---------------------------------------------------------------
         --------------
         TypeError                                 Traceback (most rec
         ent call last)
         <ipython-input-60-76b6ec007eb5> in <module>()
               1 l = [1, '5', 2]
         ----> 2 l.sort()
               3 l

         TypeError: '<' not supported between instances of 'str' and
          'int'
```

Lists in Python – reverse

- You can reverse sort() a list:

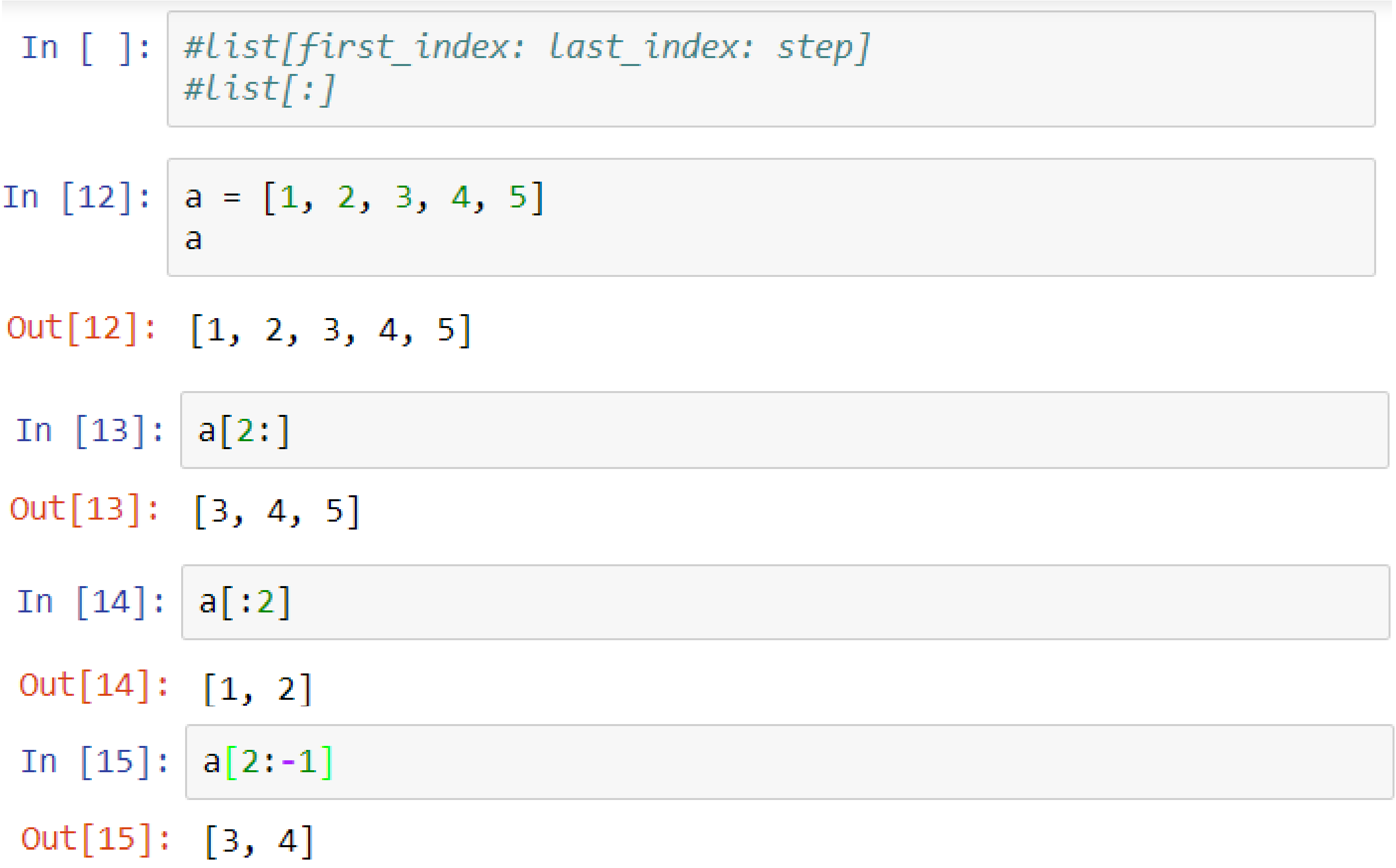


```
In [61]: l = ['a', 'b', 'c']
         l.sort(reverse=True)
         l

Out[61]: ['c', 'b', 'a']
```

- https://mysidc.statistics.gov.my/indikator/downloadfile.php?ddd=xls|8588

Lists in Python – Slicing

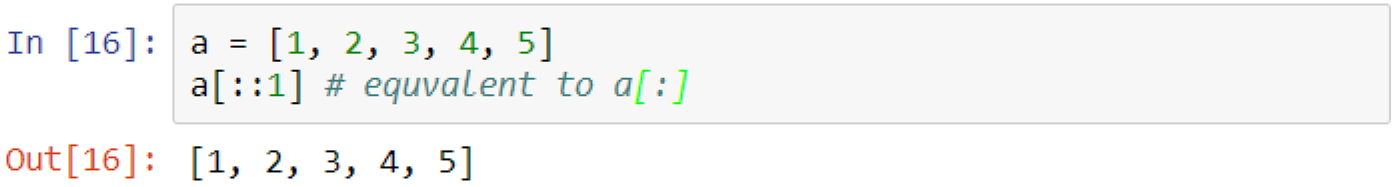
- Slicing use the symbol ‘ : ’ to access to part of a list:



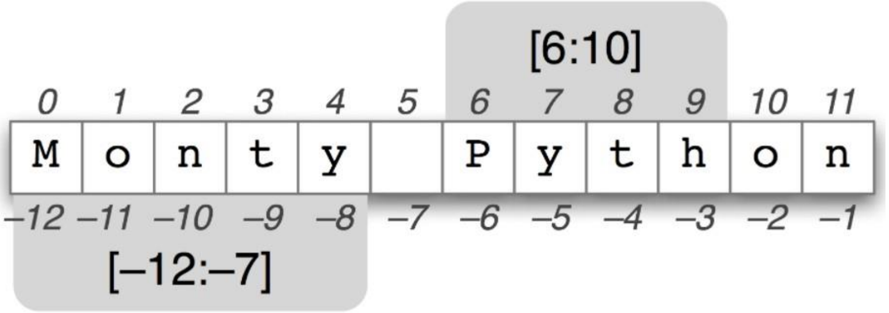
```
In [ ]: #list[first_index: last_index: step]
        #list[:]

In [12]: a = [1, 2, 3, 4, 5]
         a

Out[12]: [1, 2, 3, 4, 5]

In [13]: a[2:]

Out[13]: [3, 4, 5]

In [14]: a[:2]

Out[14]: [1, 2]

In [15]: a[2:-1]

Out[15]: [3, 4]
```

Lists in Python – Slicing

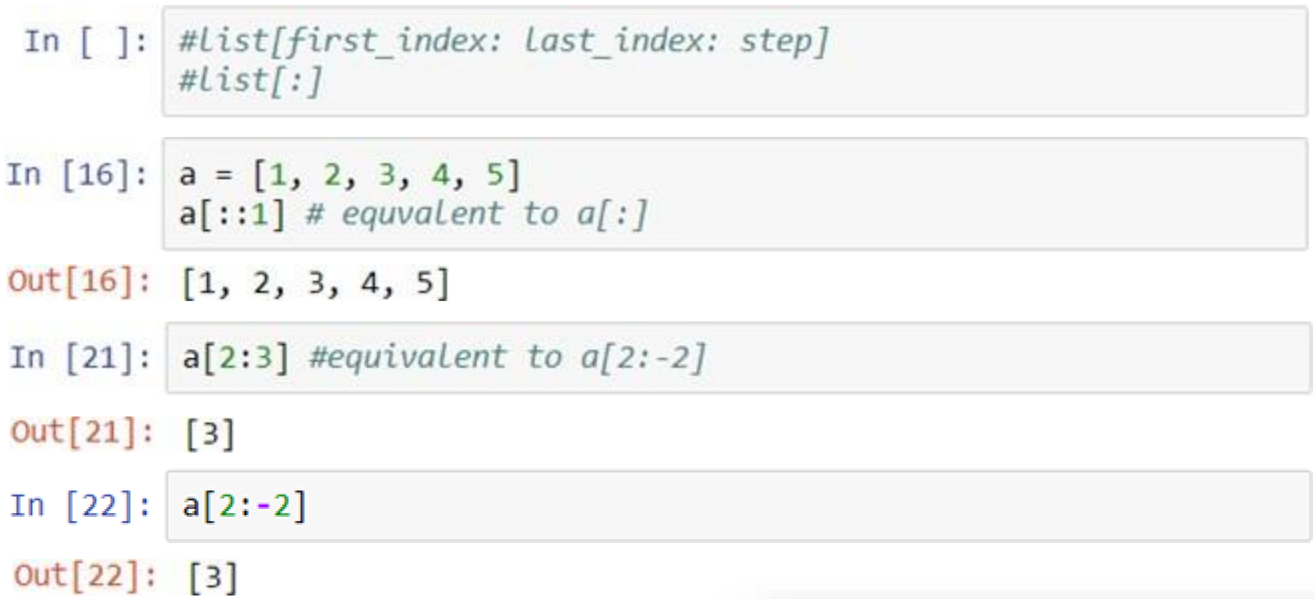
- By default the first index is 0, the last index is the last one, and the step is 1.



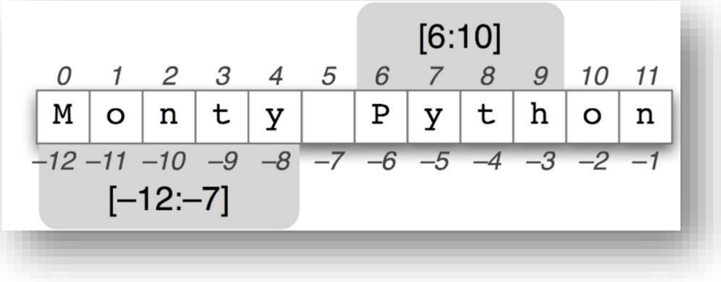
```
In [16]: a = [1, 2, 3, 4, 5]
         a[::1] # equvalent to a[:]

Out[16]: [1, 2, 3, 4, 5]
```



Lists in Python – Slicing with Negative Index



```
In [ ]:  #list[first_index: last_index: step]
         #list[:]

In [16]: a = [1, 2, 3, 4, 5]
         a[::1] # equvalent to a[:]

Out[16]: [1, 2, 3, 4, 5]

In [21]: a[2:3] #equivalent to a[2:-2]

Out[21]: [3]

In [22]: a[2:-2]

Out[22]: [3]
```



Exercise 3 – Palindrome (string that read the same forwards and backwards, e.g. ‘Anna’, ‘level’)

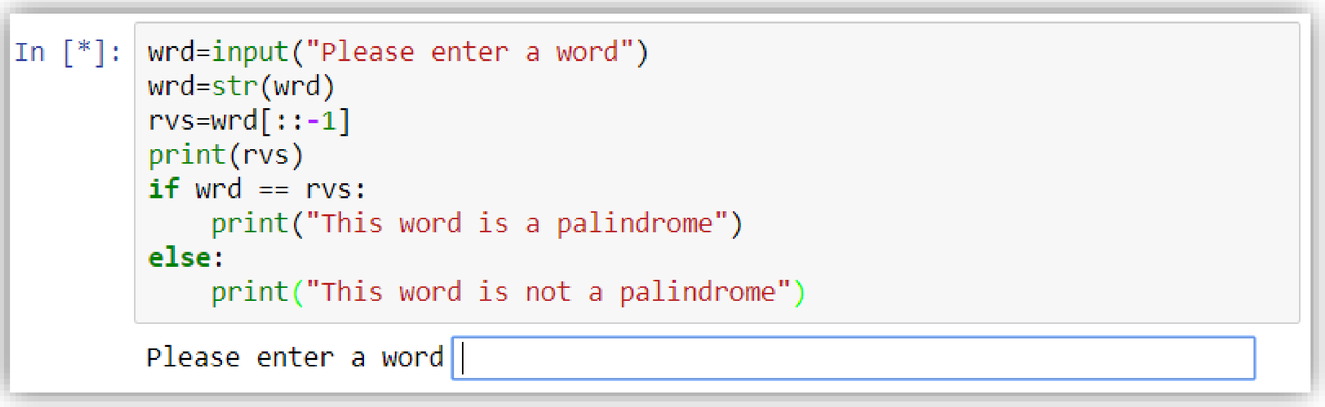
Create a program that ask the user to input a string.

Print out whether this string is a palindrome.

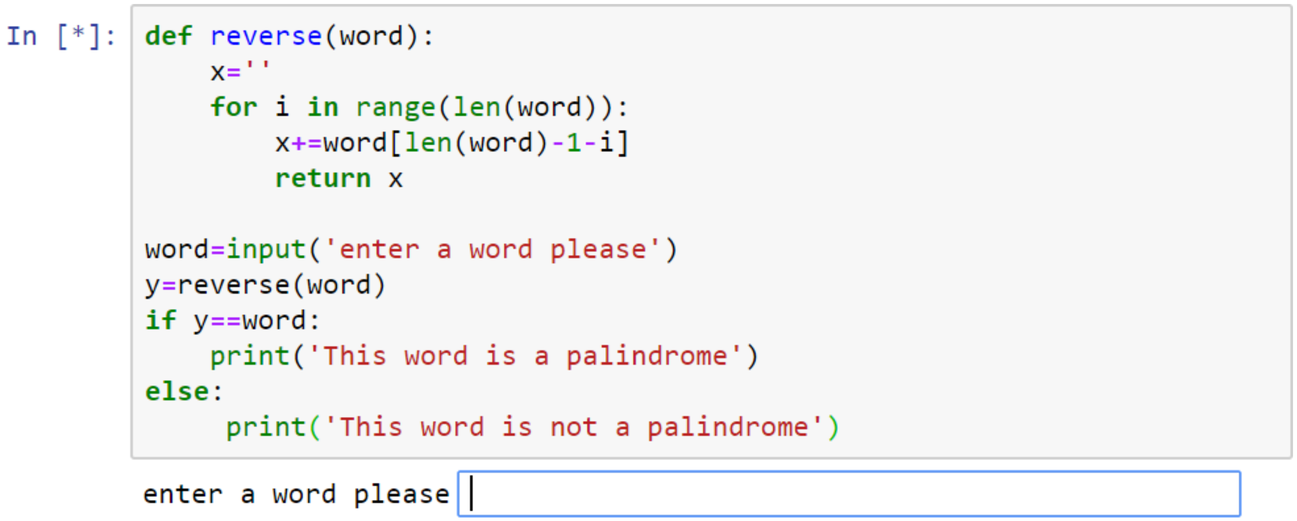
******Hint to reverse a string using slicer with negative steps:*

```
ReverseWord = Word[: : -1]
```

Exercise 3 – Solution 1 and 2 (Palindrome)



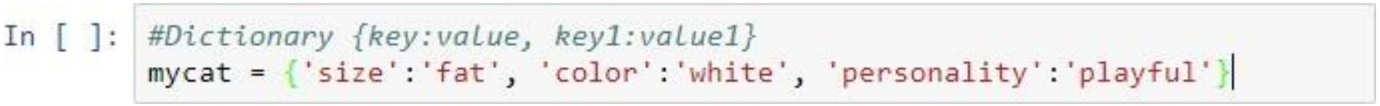
```
In [*]: wrd=input("Please enter a word")
        wrd=str(wrd)
        rvs=wrd[::-1]
        print(rvs)
        if wrd == rvs:
            print("This word is a palindrome")
        else:
            print("This word is not a palindrome")

        Please enter a word |
```



```
In [*]: def reverse(word):
            x=''
            for i in range(len(word)):
                x+=word[len(word)-1-i]
                return x

        word=input('enter a word please')
        y=reverse(word)
        if y==word:
            print('This word is a palindrome')
        else:
             print('This word is not a palindrome')

        enter a word please |
```

Dictionaries in Python

- A dictionary stores (key, value) pairs. For example:

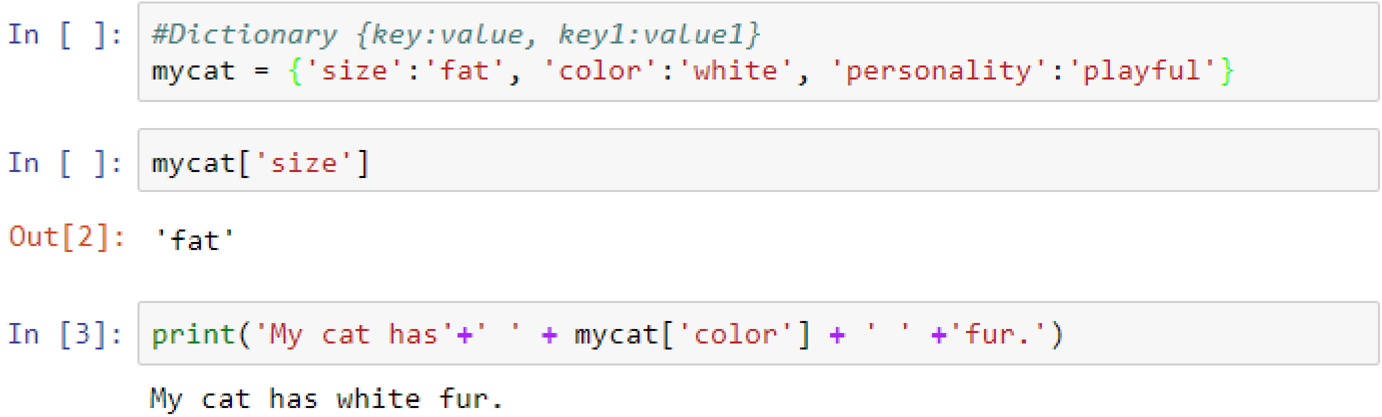


```
In [ ]: #Dictionary {key:value, key1:value1}
        mycat = {'size':'fat', 'color':'white', 'personality':'playful'}
```

- A dictionary is a sequence of item pairs.
- Dictionaries are not sorted
- Dictionaries does not have a sequence

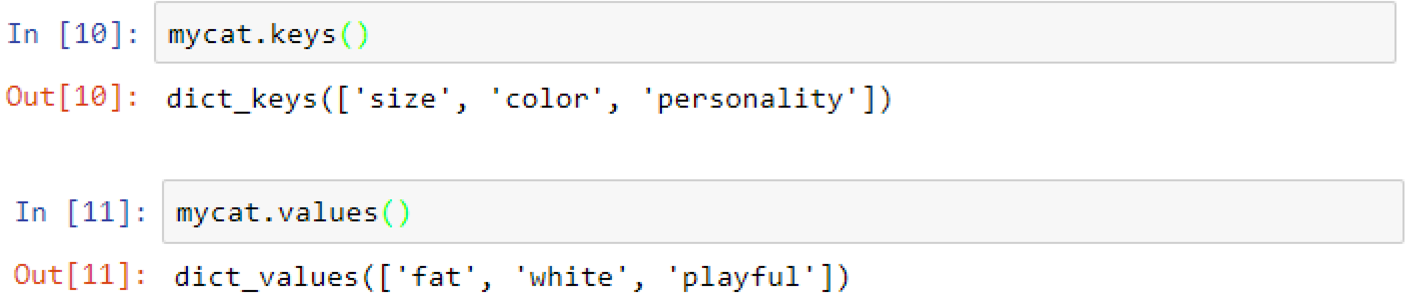
https://codeshare.io/BAOw8x

Dictionaries in Python



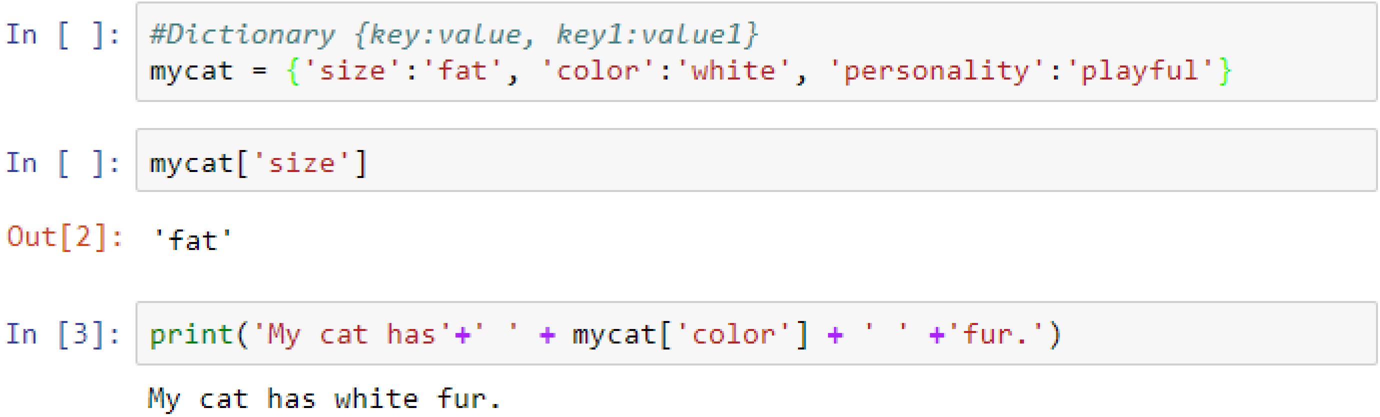
```
In [ ]: #Dictionary {key:value, key1:value1}
        mycat = {'size':'fat', 'color':'white', 'personality':'playful'}

In [ ]: mycat['size']

Out[2]: 'fat'

In [3]: print('My cat has'+' ' + mycat['color'] + ' ' +'fur.')

        My cat has white fur.
```

It’s possible to access only the keys() or the values():



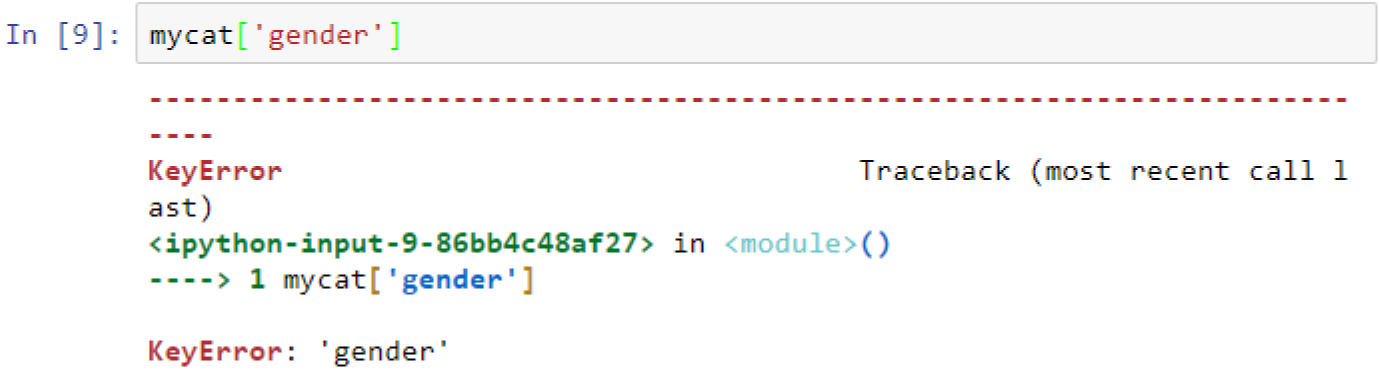
```
In [10]: mycat.keys()

Out[10]: dict_keys(['size', 'color', 'personality'])

In [11]: mycat.values()

Out[11]: dict_values(['fat', 'white', 'playful'])
```

Dictionaries in Python



```
In [ ]:  #Dictionary {key:value, key1:value1}
         mycat = {'size':'fat', 'color':'white', 'personality':'playful'}

In [ ]:  mycat['size']

Out[2]:  'fat'

In [3]:  print('My cat has'+' ' + mycat['color'] + ' ' +'fur.')

         My cat has white fur.
```

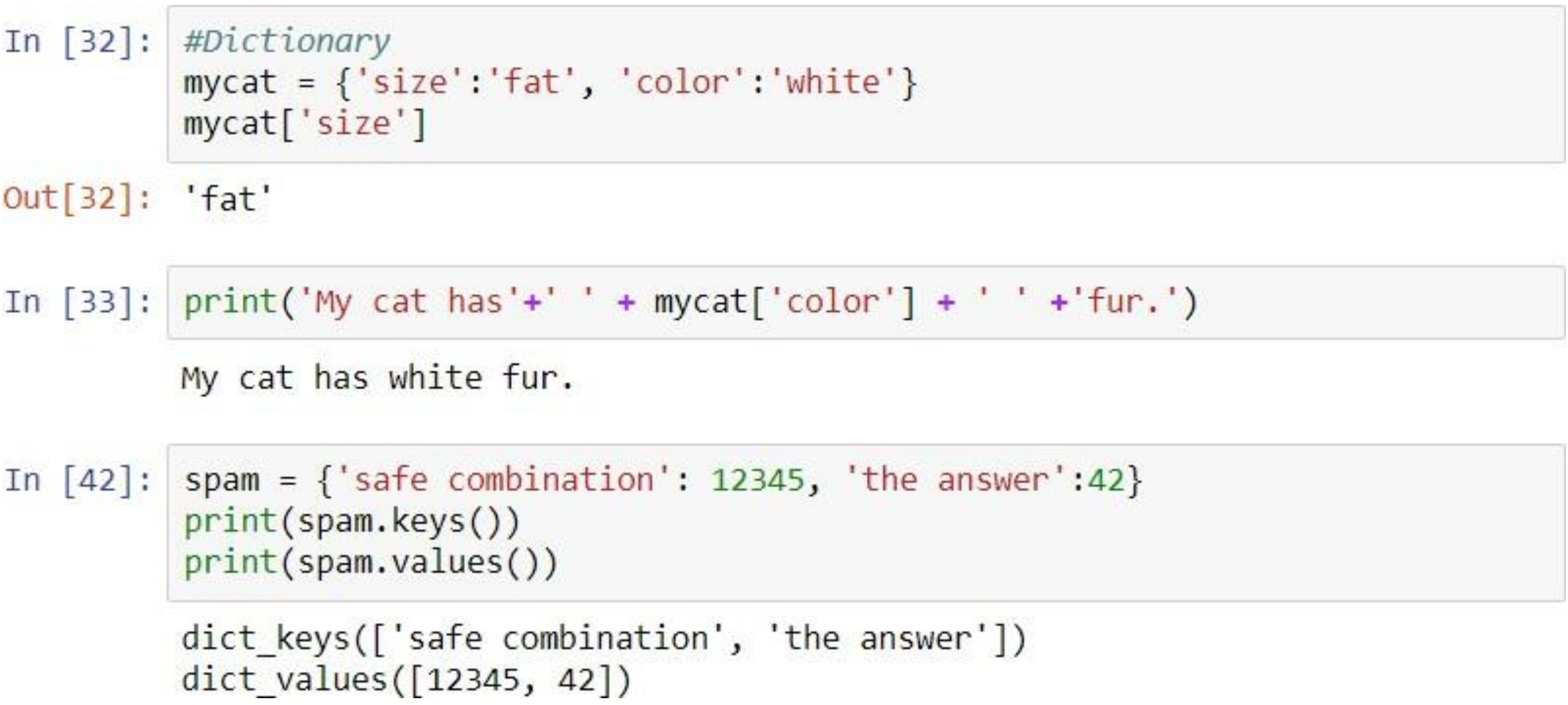
What happens when you try to call a non-existing item?



```
In [9]:  mycat['gender']

         ------------------------------------------------------------------------
         ----
         KeyError                                  Traceback (most recent call l
         ast)
         <ipython-input-9-86bb4c48af27> in <module>()
         ----> 1 mycat['gender']

         KeyError: 'gender'
```

Dictionaries in Python

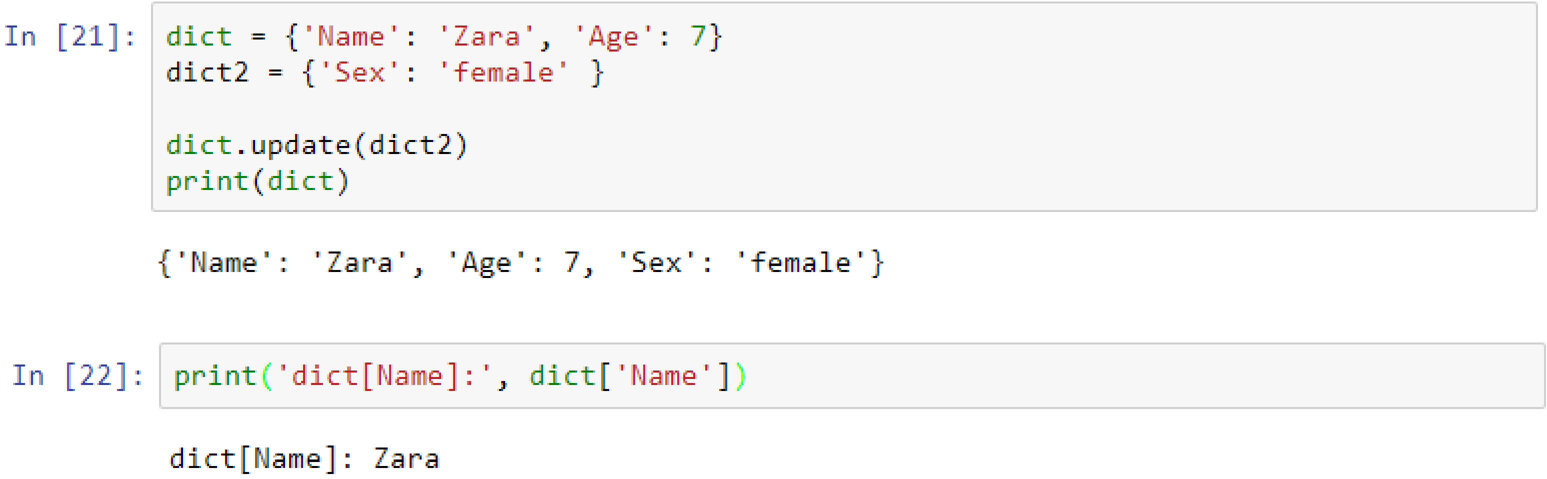
- The corresponding value of each pair can be accessed easily, or use keys() and values():



```
In [32]: #Dictionary
         mycat = {'size':'fat', 'color':'white'}
         mycat['size']

Out[32]: 'fat'

In [33]: print('My cat has'+' ' + mycat['color'] + ' ' +'fur.')

         My cat has white fur.

In [42]: spam = {'safe combination': 12345, 'the answer':42}
         print(spam.keys())
         print(spam.values())

         dict_keys(['safe combination', 'the answer'])
         dict_values([12345, 42])
```

Dictionaries in Python

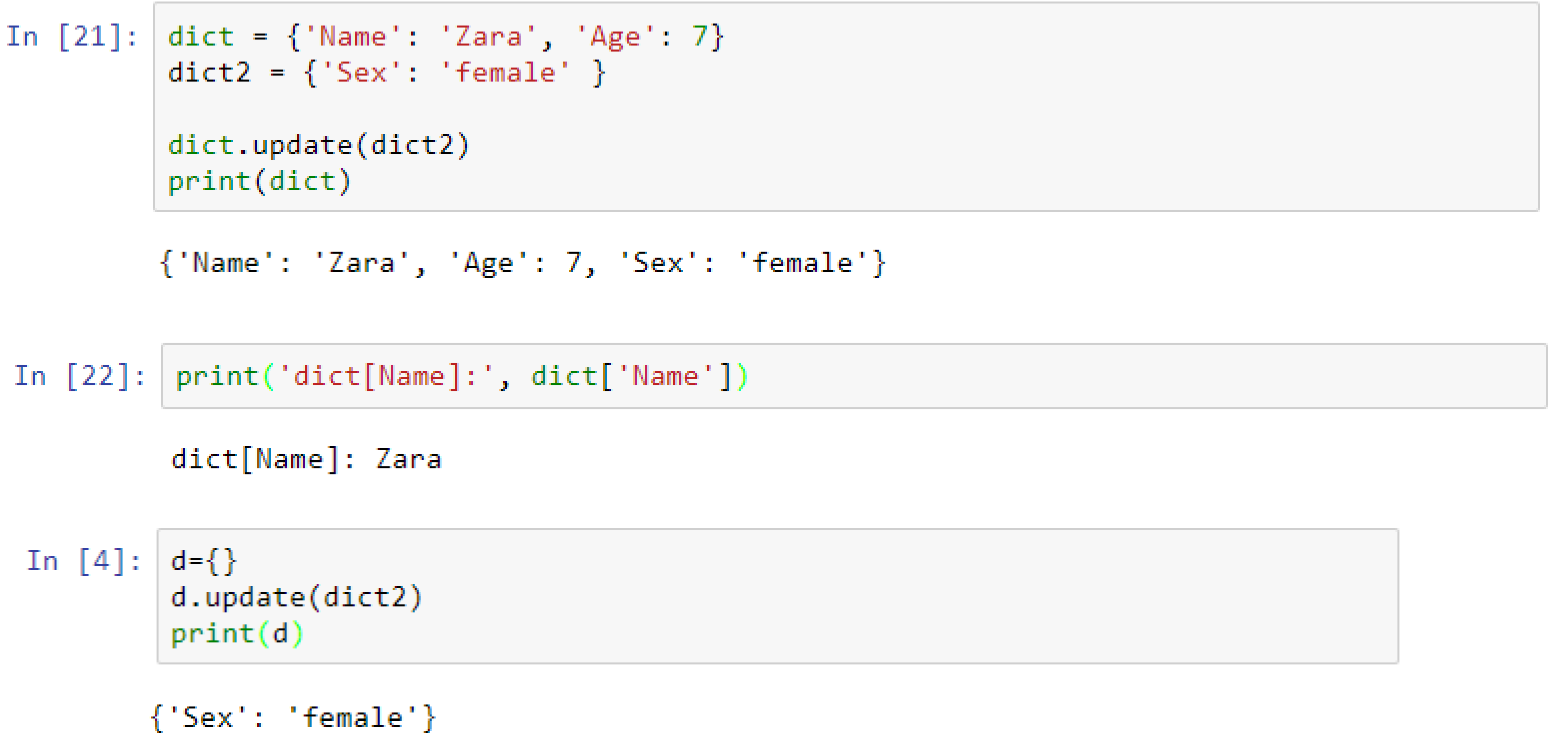
- Update() allows merging of two dictionaries:



```
In [21]: dict = {'Name': 'Zara', 'Age': 7}
         dict2 = {'Sex': 'female' }

         dict.update(dict2)
         print(dict)

         {'Name': 'Zara', 'Age': 7, 'Sex': 'female'}

In [22]: print('dict[Name]:', dict['Name'])

         dict[Name]: Zara
```

Dictionaries in Python

- Update() allows merging of two dictionaries:



```
In [21]: dict = {'Name': 'Zara', 'Age': 7}
         dict2 = {'Sex': 'female' }

         dict.update(dict2)
         print(dict)

         {'Name': 'Zara', 'Age': 7, 'Sex': 'female'}

In [22]: print('dict[Name]:', dict['Name'])

         dict[Name]: Zara

In [4]:  d={}
         d.update(dict2)
         print(d)

         {'Sex': 'female'}
```

Exercise 4 – Dictionary

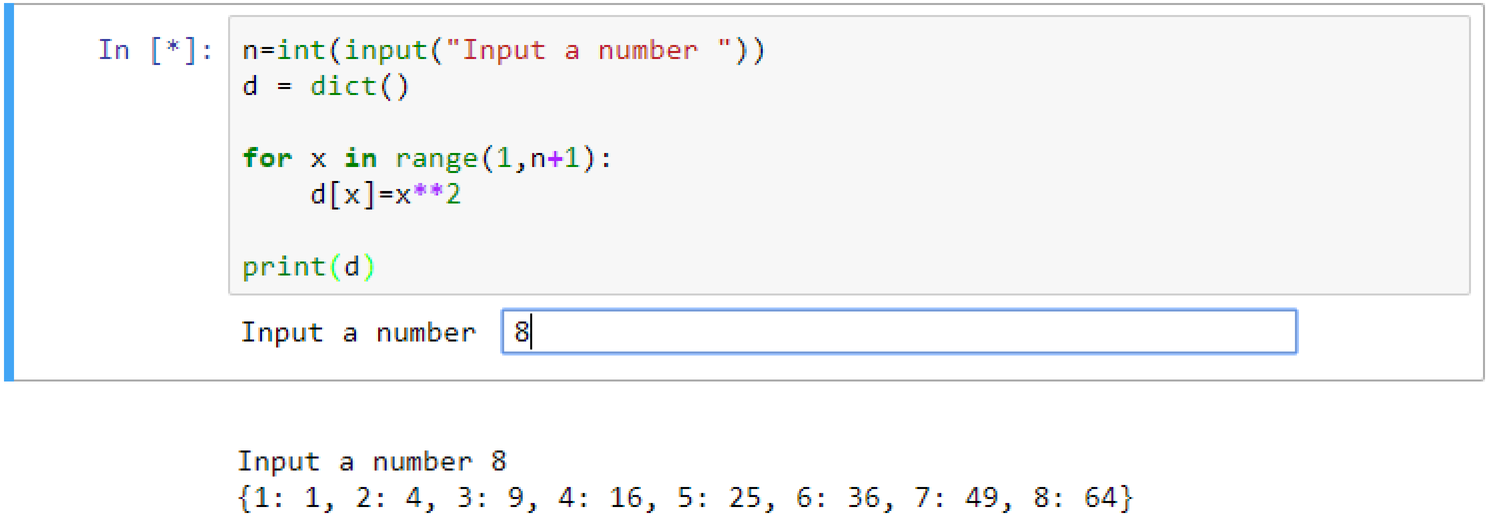
Ask the user to input a number n.

Create a Python script to generate and print a dictionary that contains the numbers (between 1 and n) in the form of (x_1 : $x_1$2, x_2:x_2**2).**

*Sample Dictionary (n = 5) :

Expected Output : {1: 1, 2: 4, 3: 9, 4: 16, 5: 25}:

Exercise 4 – Solution (Dictionary)



In [*]:

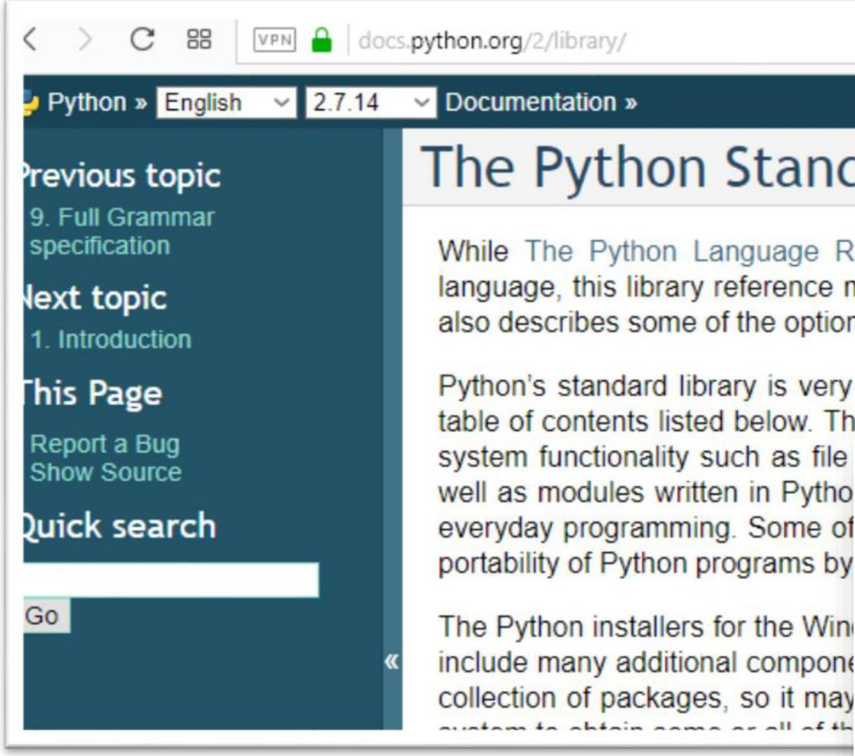
```
n=int(input("Input a number "))
d = dict()

for x in range(1,n+1):
    d[x]=x**2

print(d)
```

Input a number 8

```
Input a number 8
{1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64}
```

Working in Libraries in Python

What are libraries for Python Programming?



docs.python.org/2/library/

Python » English 2.7.14 Documentation »

Previous topic

9. Full Grammar specification

Next topic

1. Introduction

This Page

Report a Bug
Show Source

Quick search

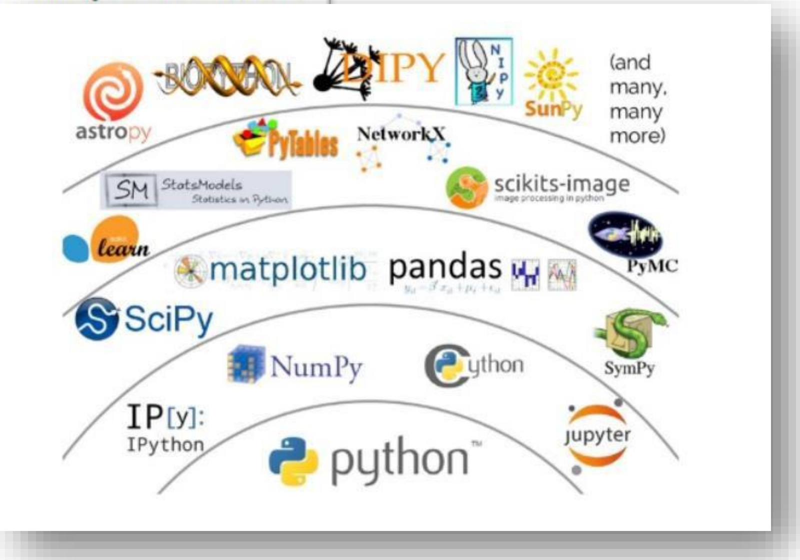
Go

The Python Standard Library

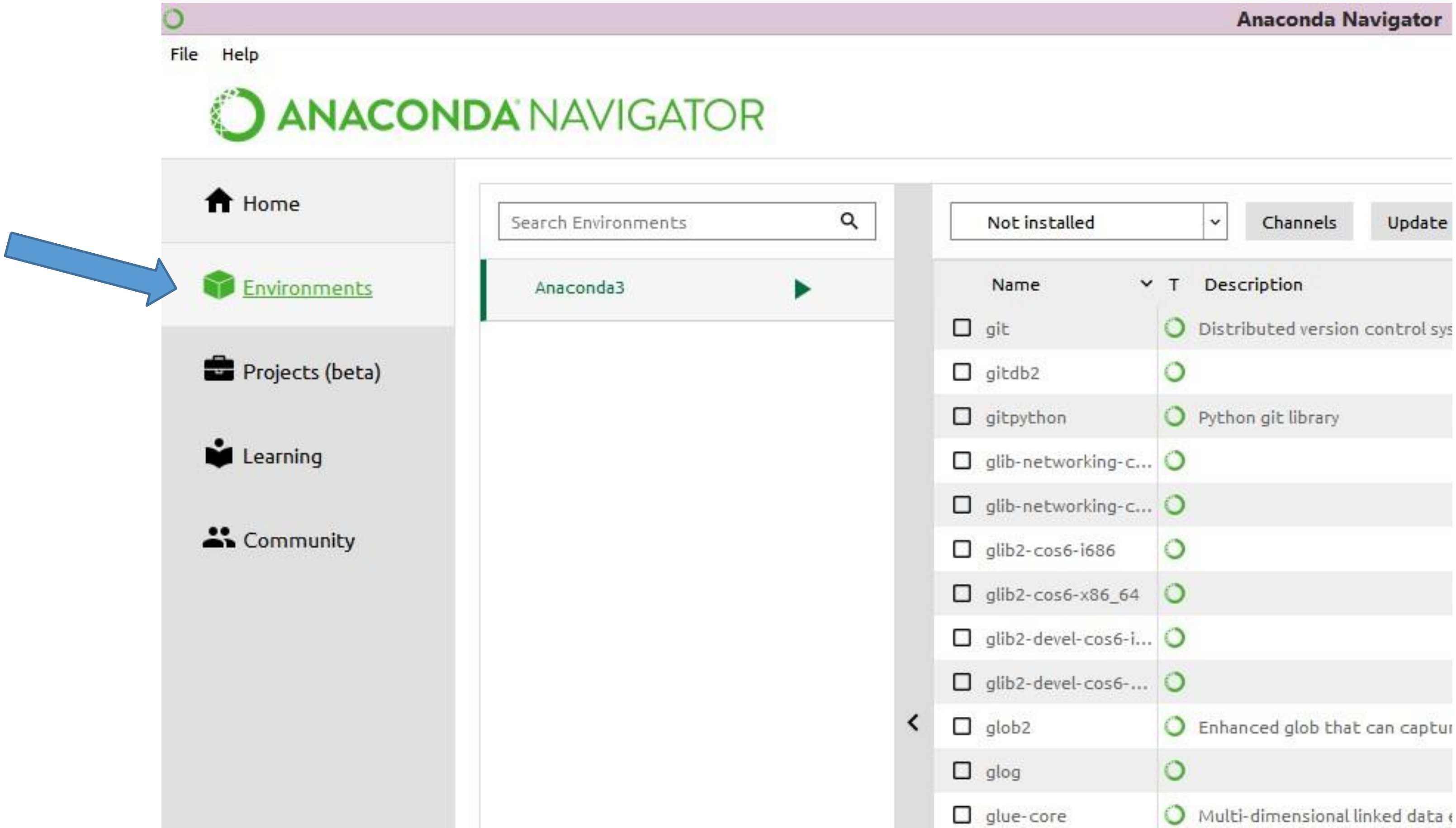
While The Python Language Reference describes the language, this library reference manual describes the sta also describes some of the optional components that are c

Python's standard library is very extensive, offering a wic table of contents listed below. The library contains built-in system functionality such as file well as modules written in Pytho everyday programming. Some of portability of Python programs by

The Python installers for the Win include many additional compone collection of packages, so it may



What are libraries?



Not installed

	Name	T	Description
□	git	○	Distributed version control sys
□	gitdb2	○	
□	gitpython	○	Python git library
□	glib-networking-c...	○	
□	glib-networking-c...	○	
□	glib2-cos6-i686	○	
□	glib2-cos6-x86_64	○	
□	glib2-devel-cos6-i...	○	
□	glib2-devel-cos6-...	○	
□	glob2	○	Enhanced glob that can captu
□	glog	○	
□	glue-core	○	Multi-dimensional linked data

Working in Library in Python

- Regular Expressions (Python Standard Library)
- NumPy
- Matplotlib

Regular Expressions (re or regex)

- Regular Expressions are tools for matching text patterns
- Normally used to search certain text patterns (web pages, emails, phone numbers and more)

Using Regular Expressions

Regular Expressions uses 2 types of characters:

- **Meta characters**: these characters have a special meaning, similar to * in wild cards
- **Literals**: e.g. a, b, 1, 2 ..

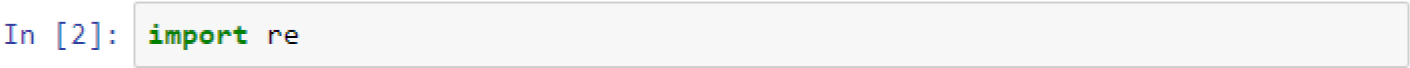
Using Regular Expressions

Most common uses for Regular Expressions are:

- Search a string (search and match)
- Find a string (findall)
- Break string into a sub string (split)
- Replace part of a string (sub)

Using Regular Expressions

Regular expressions is a module in Python’s standard library, import to use it:



```
import re
```

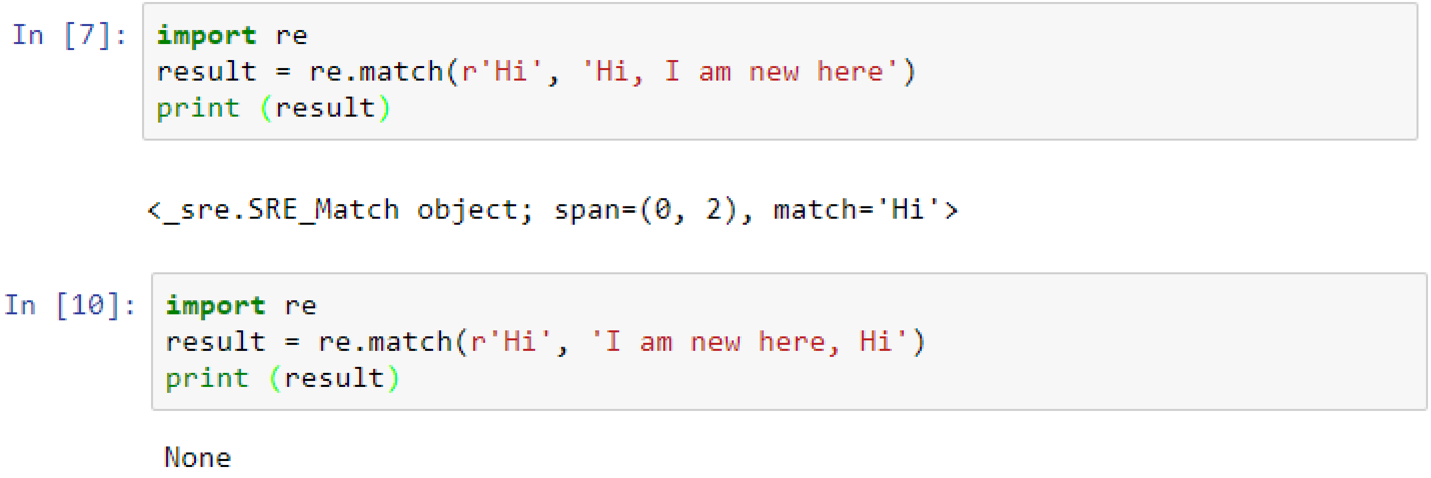
Using Regular Expressions

The ‘re’ package provides multiple methods to perform queries on an input string, most commonly used methods:

- re.match()
- re.search()
- re.findall()
- re.split()
- re.sub()
- re.compile()

Using RE– re.match(pattern, string)

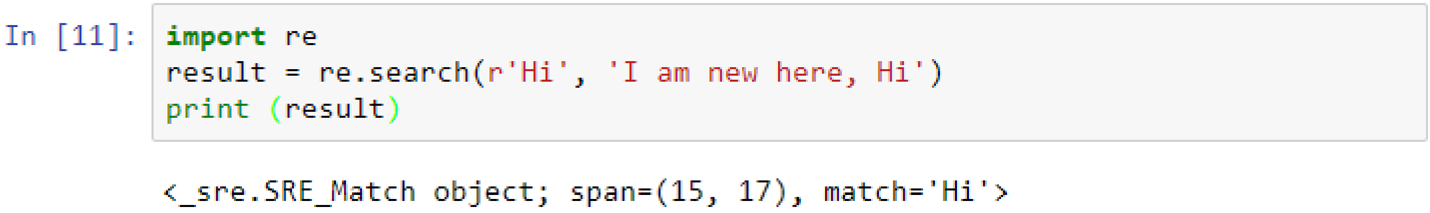
The above method finds match if it occurs at the **start** of the string:



```
In [7]:  import re
         result = re.match(r'Hi', 'Hi, I am new here')
         print (result)

         <_sre.SRE_Match object; span=(0, 2), match='Hi'>

In [10]: import re
         result = re.match(r'Hi', 'I am new here, Hi')
         print (result)

         None
```

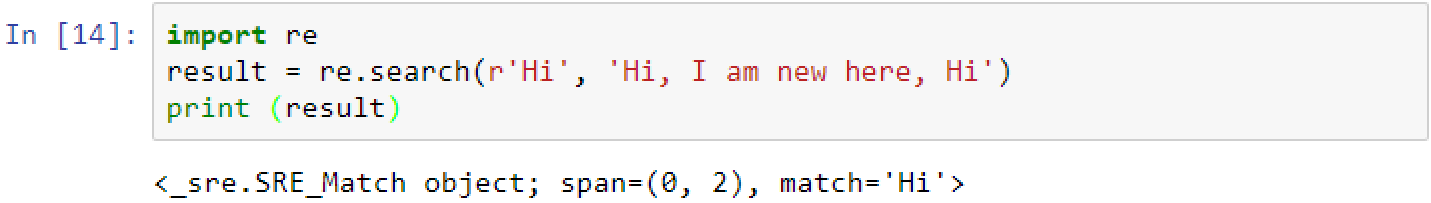
Using RE– re.search(pattern, string)

Similar to re.match(), though re.search() does not restrict to only the beginning of the string:



```
In [11]: import re
         result = re.search(r'Hi', 'I am new here, Hi')
         print (result)

         <_sre.SRE_Match object; span=(15, 17), match='Hi'>
```

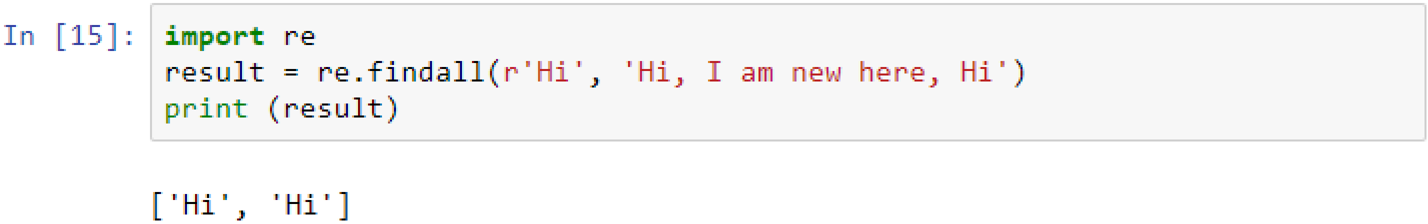
However, it looks for the pattern for one time only:



```
In [14]: import re
         result = re.search(r'Hi', 'Hi, I am new here, Hi')
         print (result)

         <_sre.SRE_Match object; span=(0, 2), match='Hi'>
```

Using RE– re.findall(pattern, string)

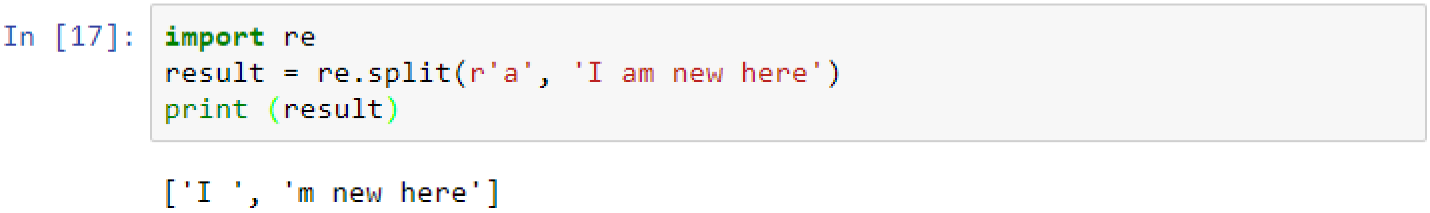
To find all occurrences, use re.findall():



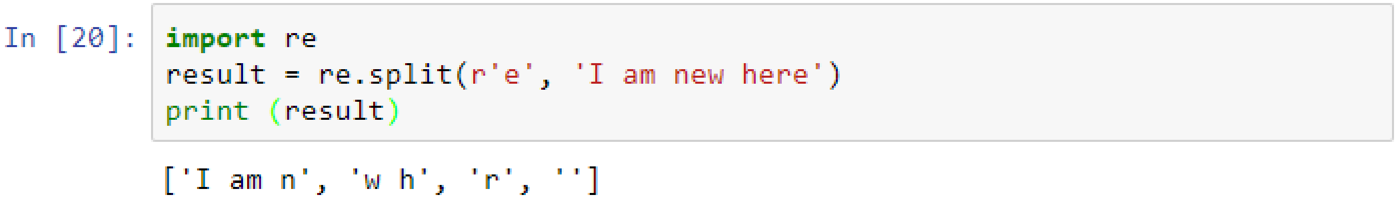
```
In [15]: import re
         result = re.findall(r'Hi', 'Hi, I am new here, Hi')
         print (result)

         ['Hi', 'Hi']
```

Using RE– re.split(pattern, string, [maxsplit=0])

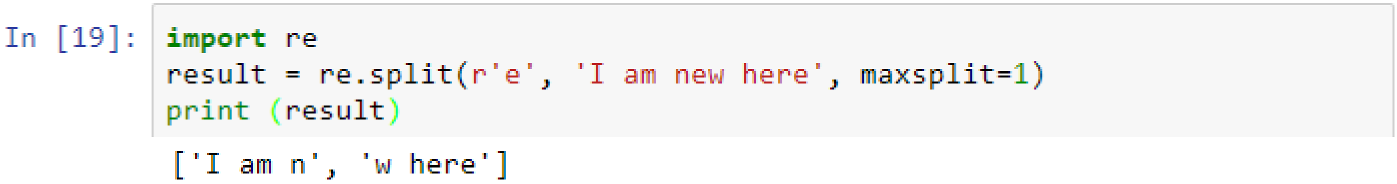
To split a string, use re.split():



```
In [17]: import re
         result = re.split(r'a', 'I am new here')
         print (result)

         ['I ', 'm new here']
```



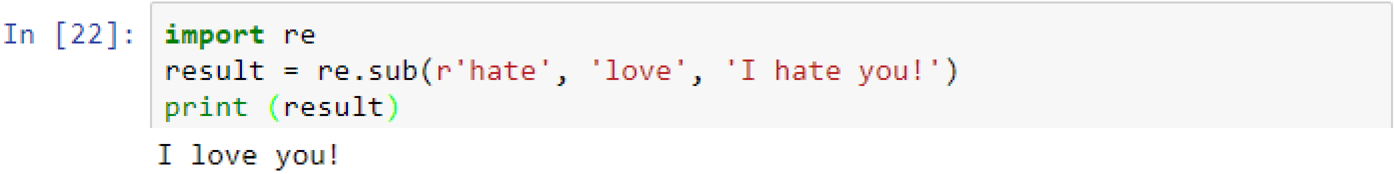
```
In [20]: import re
         result = re.split(r'e', 'I am new here')
         print (result)

         ['I am n', 'w h', 'r', '']
```



```
In [19]: import re
         result = re.split(r'e', 'I am new here', maxsplit=1)
         print (result)
         ['I am n', 'w here']
```

Using RE– re.sub(pattern, repl, string)

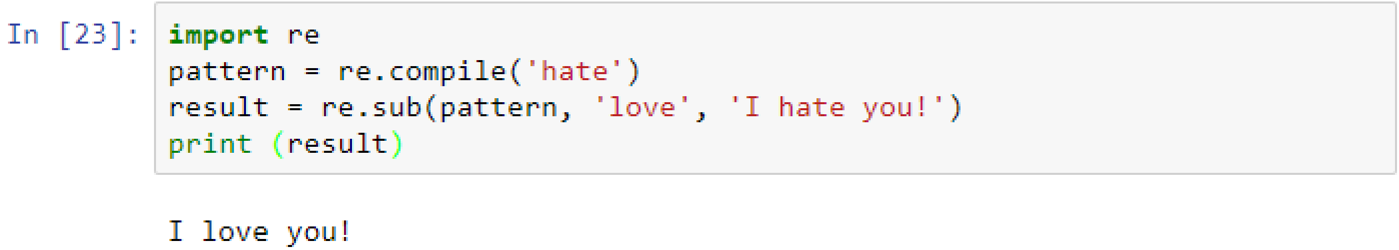
To find and replace, use re.sub():



```
In [22]: import re
         result = re.sub(r'hate', 'love', 'I hate you!')
         print (result)

         I love you!
```

Using RE– re.compile(pattern)

We can combine a regular expression pattern into a pattern objects, use re.compile():



```
In [23]: import re
         pattern = re.compile('hate')
         result = re.sub(pattern, 'love', 'I hate you!')
         print (result)

         I love you!
```

Using RE– Pattern Operators

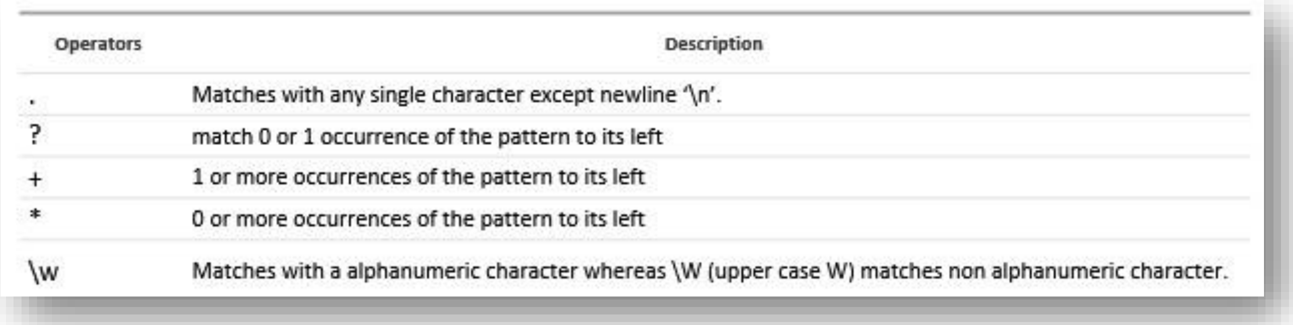
But, what if we don’t have a specific **pattern**?

Using RE– Pattern Operators

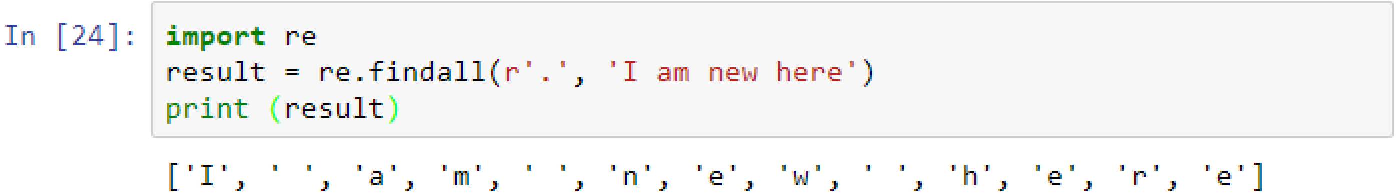
Operators	Description
.	Matches with any single character except newline '\n'.
?	match 0 or 1 occurrence of the pattern to its left
+	1 or more occurrences of the pattern to its left
*	0 or more occurrences of the pattern to its left
\w	Matches with a alphanumeric character whereas \W (upper case W) matches non alphanumeric character.
\d	Matches with digits [0-9] and /D (upper case D) matches with non-digits.
\s	Matches with a single white space character (space, newline, return, tab, form) and \S (upper case S) matches any non-white space character.
\b	boundary between word and non-word and /B is opposite of /b
[..]	Matches any single character in a square bracket and [^..] matches any single character not in square bracket
\	It is used for special meaning characters like \. to match a period or \+ for plus sign.
^ and $	^ and $ match the start or end of the string respectively
{n,m}	Matches at least n and at most m occurrences of preceding expression if we write it as {,m} then it will return at least any minimum occurrence to max m preceding expression.
a| b	Matches either a or b
( )	Groups regular expressions and returns matched text
\t, \n, \r	Matches tab, newline, return

Using RE– Pattern Operators

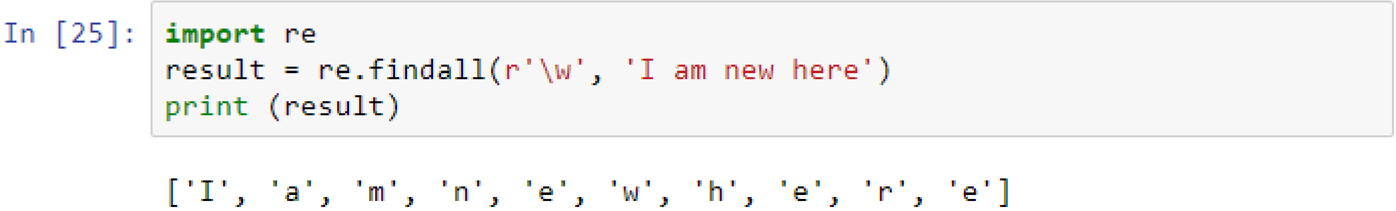
Let’s try using these pattern operators:



Operators	Description
.	Matches with any single character except newline ‘\n’.
?	match 0 or 1 occurrence of the pattern to its left
+	1 or more occurrences of the pattern to its left
*	0 or more occurrences of the pattern to its left
\w	Matches with a alphanumeric character whereas \W (upper case W) matches non alphanumeric character.



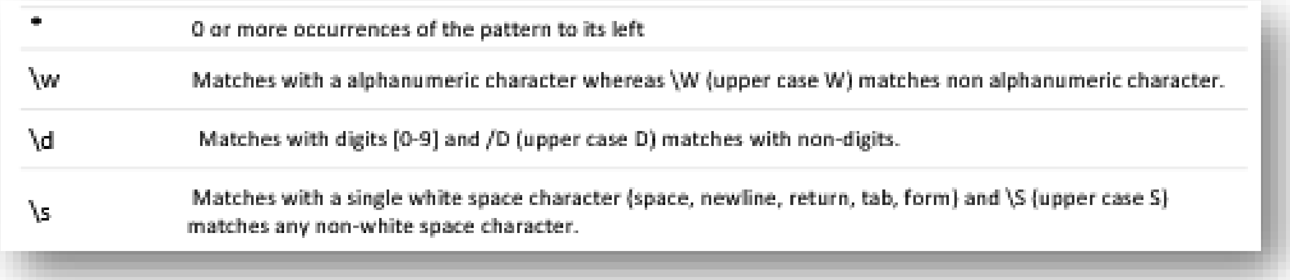
```
In [24]: import re
         result = re.findall(r'.', 'I am new here')
         print (result)

         ['I', ' ', 'a', 'm', ' ', 'n', 'e', 'w', ' ', 'h', 'e', 'r', 'e']
```



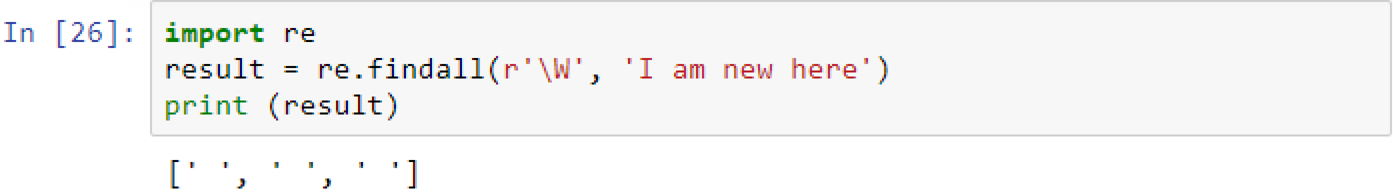
```
In [25]: import re
         result = re.findall(r'\w', 'I am new here')
         print (result)

         ['I', 'a', 'm', 'n', 'e', 'w', 'h', 'e', 'r', 'e']
```

Using RE– Pattern Operators

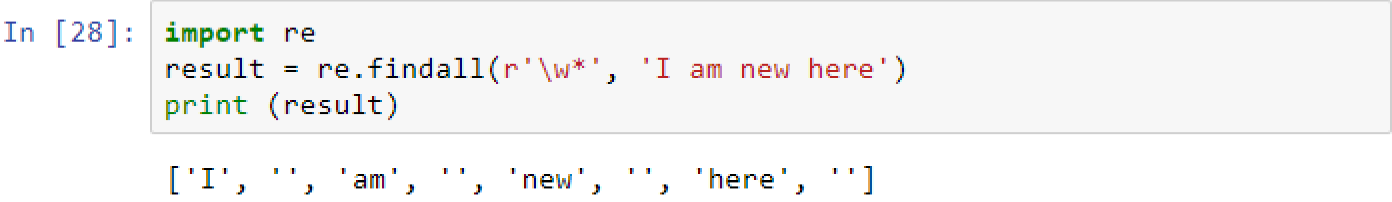
Let’s try using these pattern operators:



*	0 or more occurrences of the pattern to its left
\w	Matches with a alphanumeric character whereas \W (upper case W) matches non alphanumeric character.
\d	Matches with digits [0-9] and /D (upper case D) matches with non-digits.
\s	Matches with a single white space character (space, newline, return, tab, form) and \S (upper case S) matches any non-white space character.



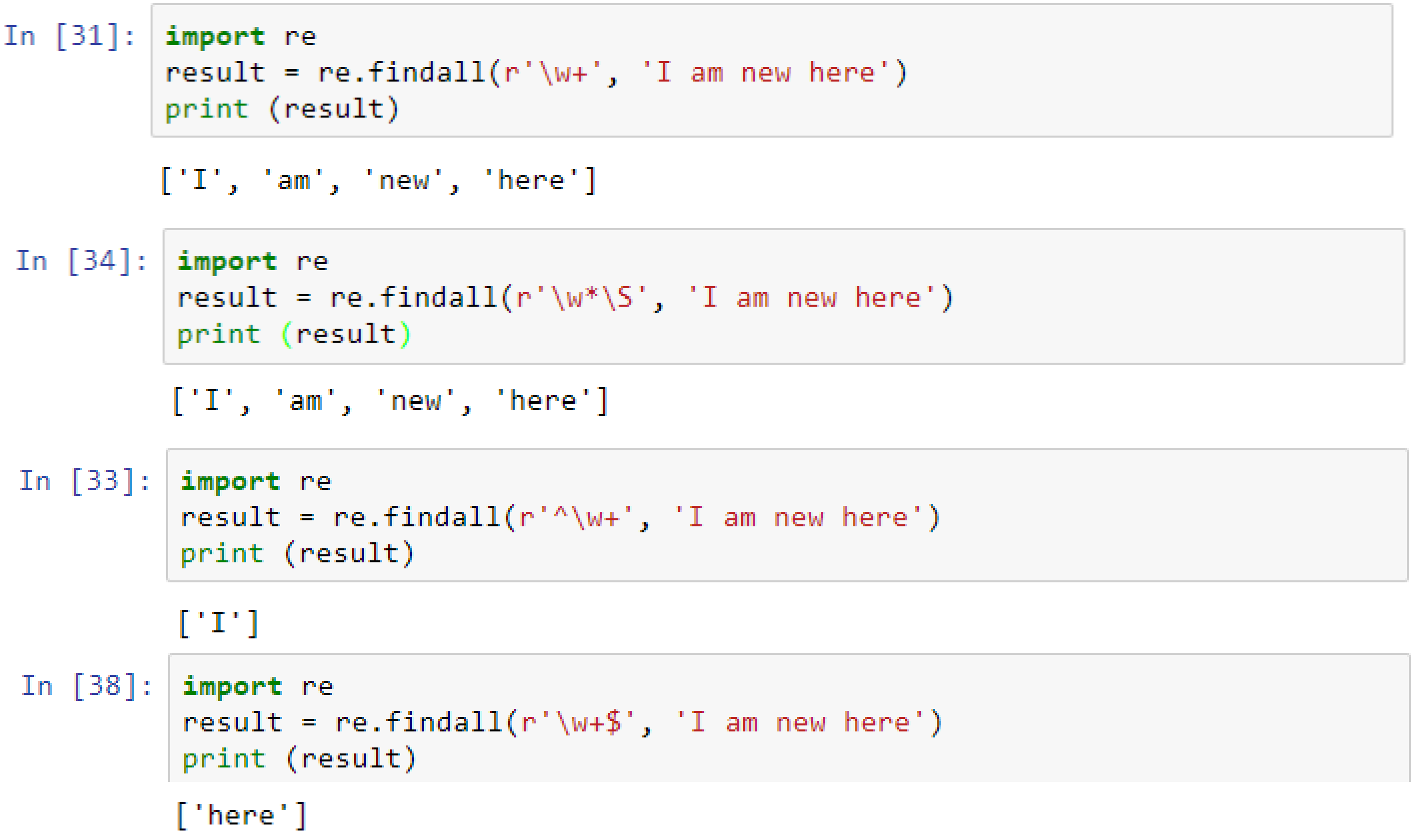
```
In [26]: import re
         result = re.findall(r'\W', 'I am new here')
         print (result)

         [' ', ' ', ' ']
```



```
In [28]: import re
         result = re.findall(r'\w*', 'I am new here')
         print (result)

         ['I', '', 'am', '', 'new', '', 'here', '']
```

Using RE– Pattern Operators

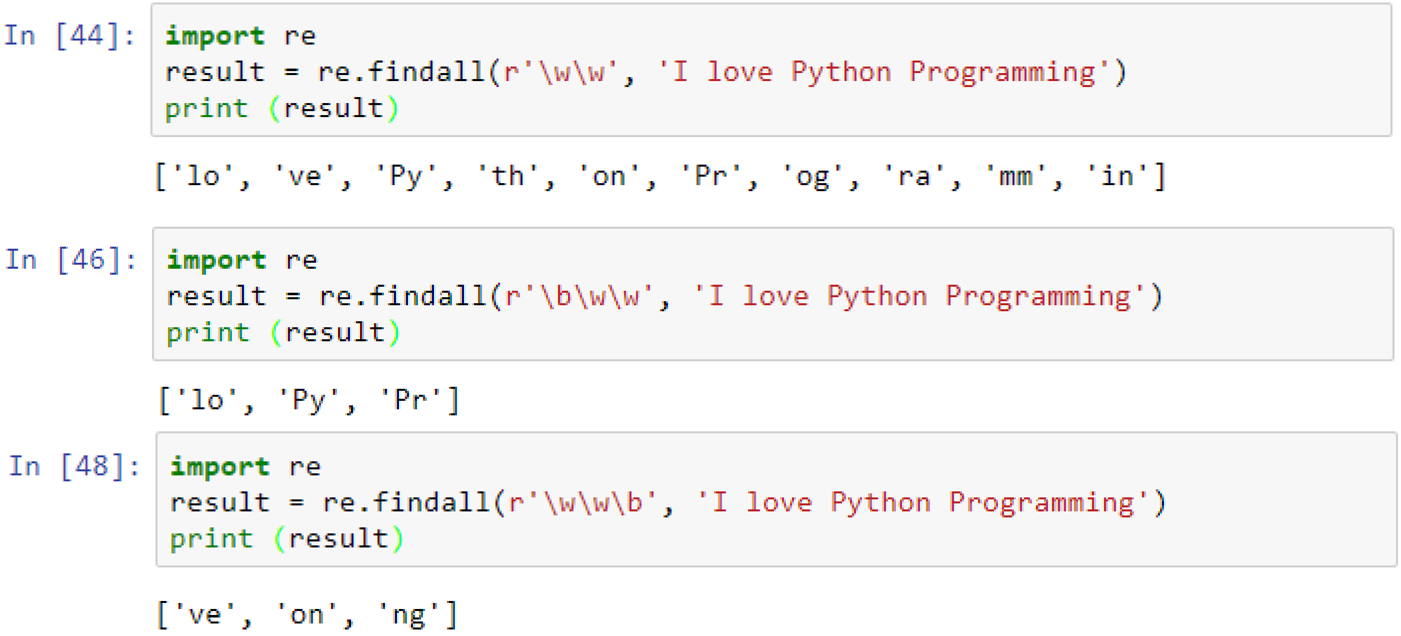
Let’s try using these pattern operators:



```
In [31]: import re
         result = re.findall(r'\w+', 'I am new here')
         print (result)

         ['I', 'am', 'new', 'here']

In [34]: import re
         result = re.findall(r'\w*\S', 'I am new here')
         print (result)

         ['I', 'am', 'new', 'here']

In [33]: import re
         result = re.findall(r'^\w+', 'I am new here')
         print (result)

         ['I']

In [38]: import re
         result = re.findall(r'\w+$', 'I am new here')
         print (result)

         ['here']
```

Using RE– Pattern Operators

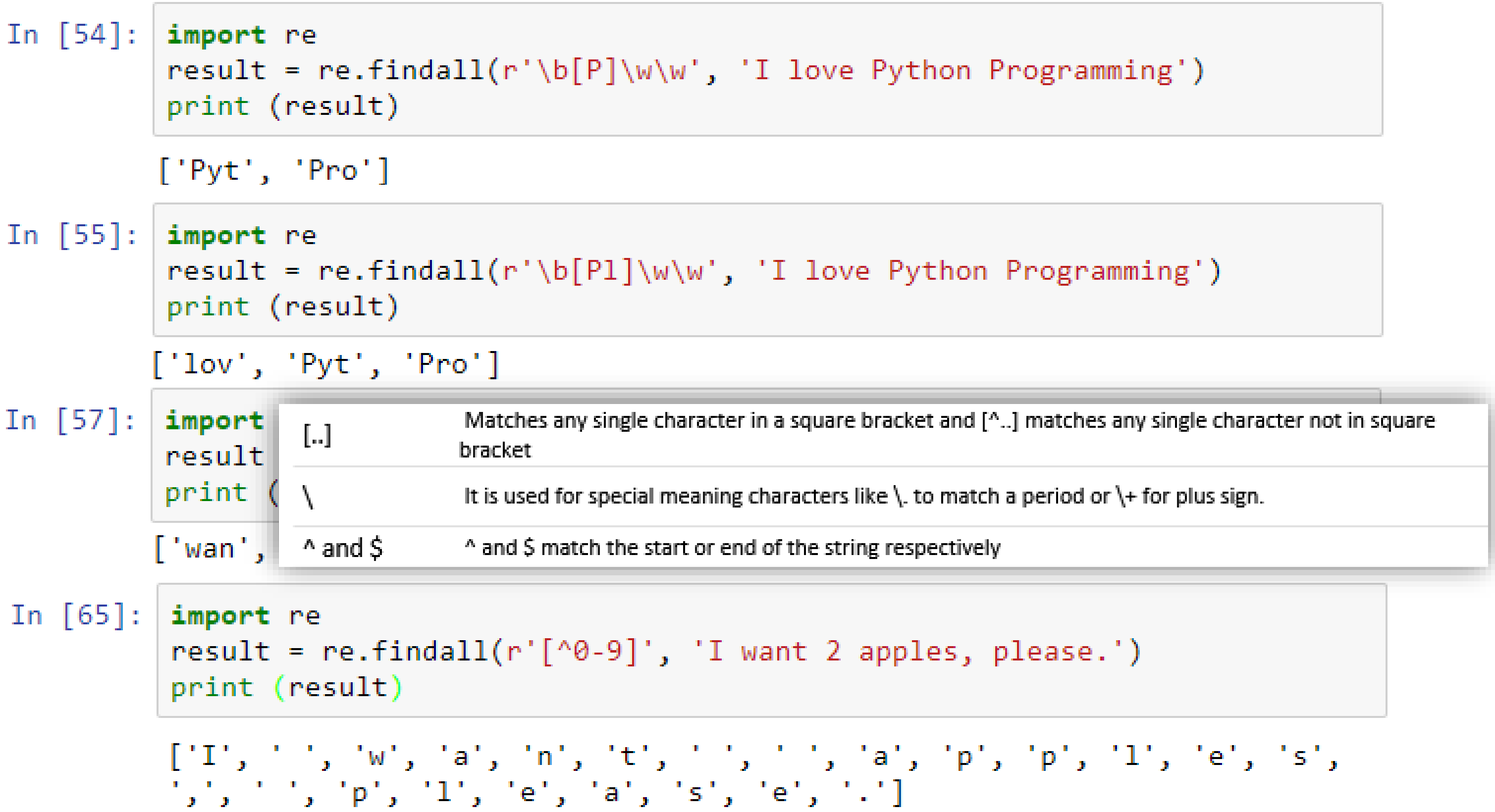
Let’s try using these pattern operators:



```
In [44]: import re
         result = re.findall(r'\w\w', 'I love Python Programming')
         print (result)

         ['lo', 've', 'Py', 'th', 'on', 'Pr', 'og', 'ra', 'mm', 'in']

In [46]: import re
         result = re.findall(r'\b\w\w', 'I love Python Programming')
         print (result)

         ['lo', 'Py', 'Pr']

In [48]: import re
         result = re.findall(r'\w\w\b', 'I love Python Programming')
         print (result)

         ['ve', 'on', 'ng']
```

Using RE– Pattern Operators

Let’s try some more:



```
In [54]: import re
         result = re.findall(r'\b[P]\w\w', 'I love Python Programming')
         print (result)

['Pyt', 'Pro']
```

```
In [55]: import re
         result = re.findall(r'\b[Pl]\w\w', 'I love Python Programming')
         print (result)

['lov', 'Pyt', 'Pro']
```

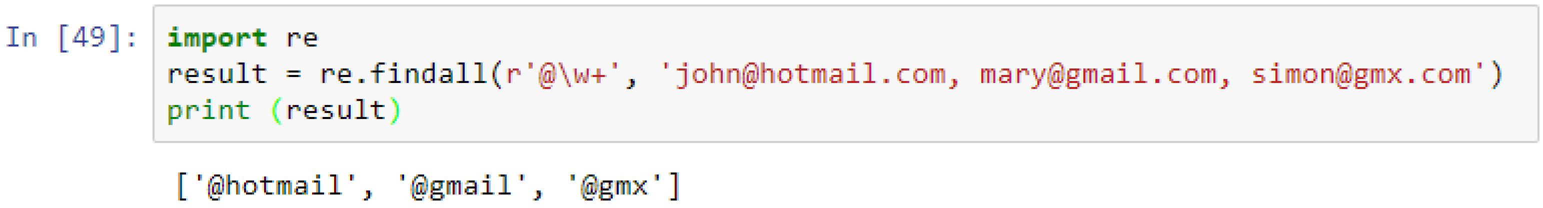
```
In [57]: import
         result
         print (

['wan',
```

[..]	Matches any single character in a square bracket and [^..] matches any single character not in square bracket
\	It is used for special meaning characters like \. to match a period or \+ for plus sign.
^ and $	^ and $ match the start or end of the string respectively

```
In [65]: import re
         result = re.findall(r'[^0-9]', 'I want 2 apples, please.')
         print (result)

['I', ' ', 'w', 'a', 'n', 't', ' ', ' ', 'a', 'p', 'p', 'l', 'e', 's',
',', ' ', 'p', 'l', 'e', 'a', 's', 'e', '.']
```

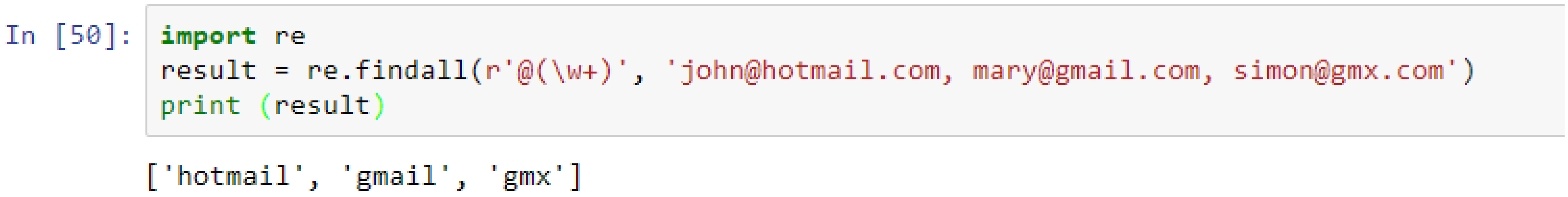
Using RE– Pattern Operators

Let’s try finding email-ids:



```
In [49]: import re
         result = re.findall(r'@\w+', 'john@hotmail.com, mary@gmail.com, simon@gmx.com')
         print (result)

         ['@hotmail', '@gmail', '@gmx']
```

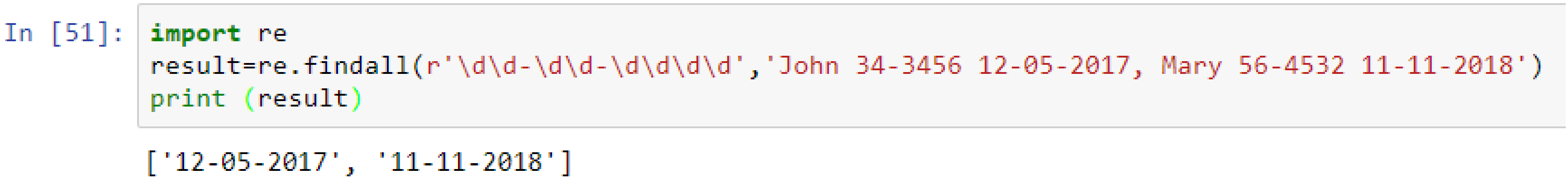
Extract only email-id names, using ‘()’ to indicate which parts you want:



```
In [50]: import re
         result = re.findall(r'@(\w+)', 'john@hotmail.com, mary@gmail.com, simon@gmx.com')
         print (result)

        ['hotmail', 'gmail', 'gmx']
```

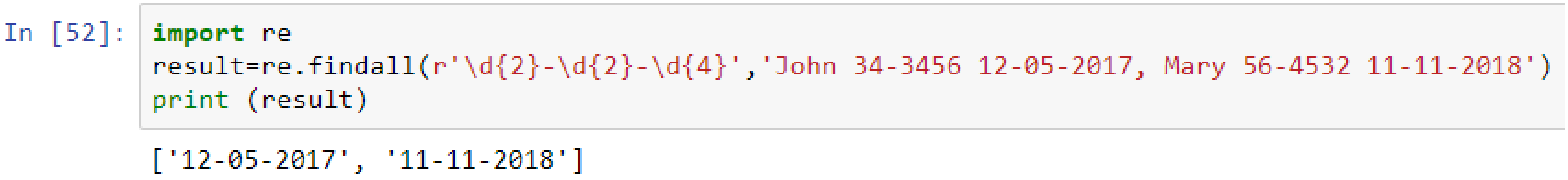
Using RE– Pattern Operators

Return date from a given string:



```
In [51]: import re
         result=re.findall(r'\d\d-\d\d-\d\d\d\d','John 34-3456 12-05-2017, Mary 56-4532 11-11-2018')
         print (result)

         ['12-05-2017', '11-11-2018']
```

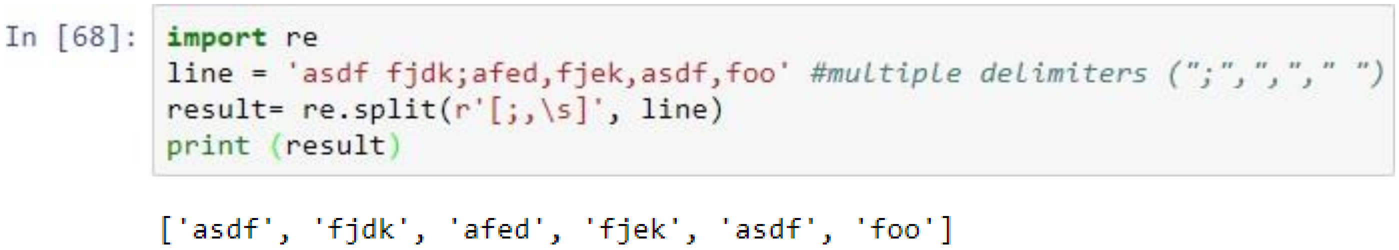
Use {number} to simplify:



```
In [52]: import re
         result=re.findall(r'\d{2}-\d{2}-\d{4}','John 34-3456 12-05-2017, Mary 56-4532 11-11-2018')
         print (result)

         ['12-05-2017', '11-11-2018']
```

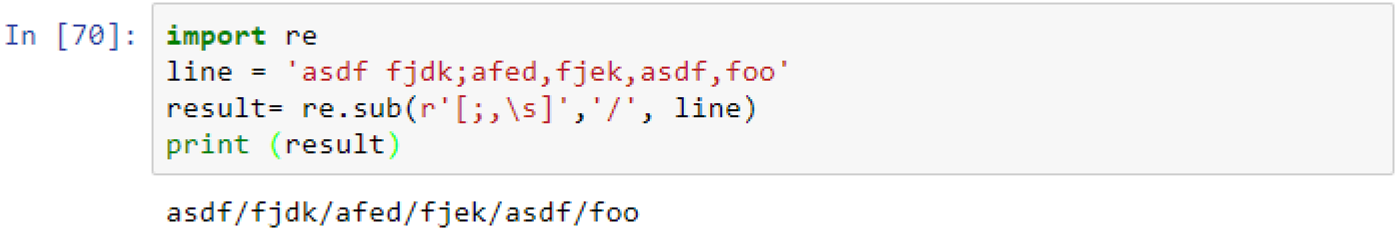
Using RE– Pattern Operators

Splitting a string with multiple delimiters:



```
In [68]: import re
         line = 'asdf fjdk;afed,fjek,asdf,foo' #multiple delimiters (";",","," ")
         result= re.split(r'[;,\s]', line)
         print (result)

         ['asdf', 'fjdk', 'afed', 'fjek', 'asdf', 'foo']
```

Replacing these delimiters with ‘/’ instead:



```
In [70]: import re
         line = 'asdf fjdk;afed,fjek,asdf,foo'
         result= re.sub(r'[;,\s]','/', line)
         print (result)

         asdf/fjdk/afed/fjek/asdf/foo
```

Exercise 5 – Regular Expressions

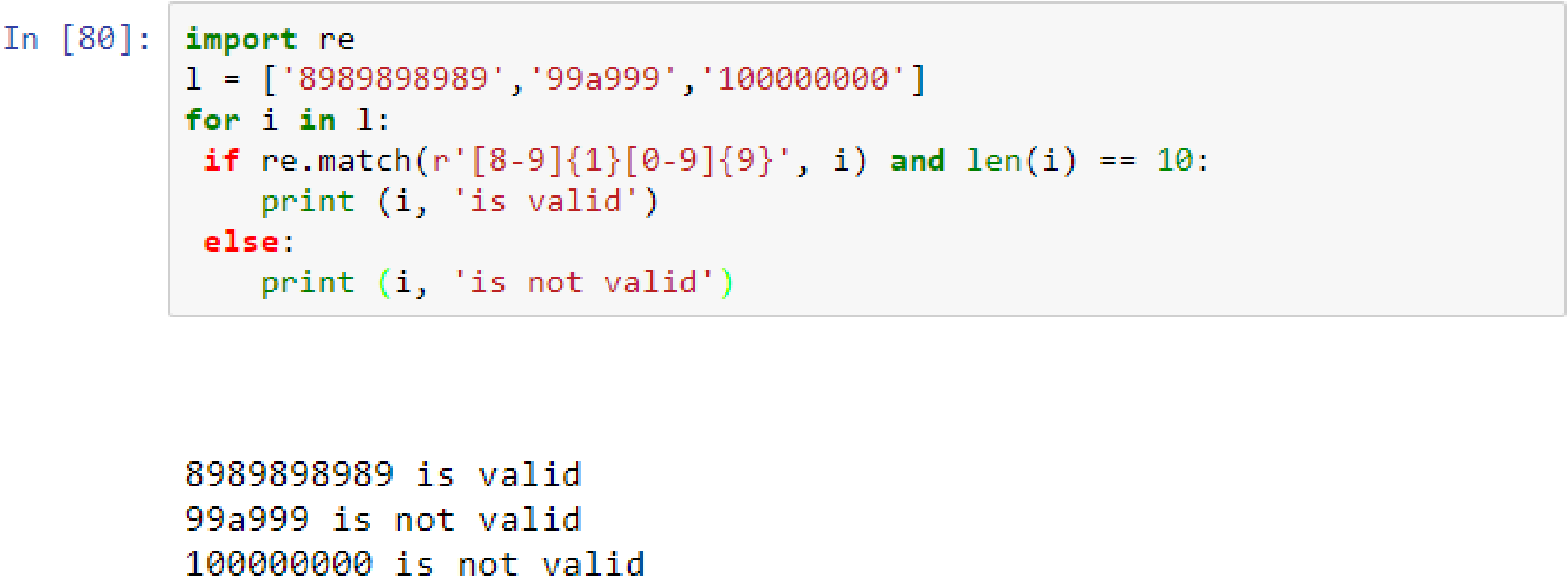
Validate if all the numbers in the list below are:

1. **Starts with 8 or 9**
2. **Must be 10 digits long**

Please print your answer to the screen (valid or not valid)

l = [‘8989898989’, ’99a999’, ‘100000000’]

Exercise 5 – Solution (Regular Expressions)



```
In [80]: import re
         l = ['8989898989','99a999','100000000']
         for i in l:
          if re.match(r'[8-9]{1}[0-9]{9}', i) and len(i) == 10:
             print (i, 'is valid')
          else:
             print (i, 'is not valid')
```

```
8989898989 is valid
99a999 is not valid
100000000 is not valid
```

Program Structure

Python Program Structure

- **Structured programming** is a programming paradigm
- It is aimed at improving the <u>clarity, quality, and development time</u> of a computer program
- Extensive use of the structured control flow constructs of selection (if/then/else) and repetition (while and for), block structures, and subroutines.
- To ensure programs are well written and easy to use

For a Better Python Program

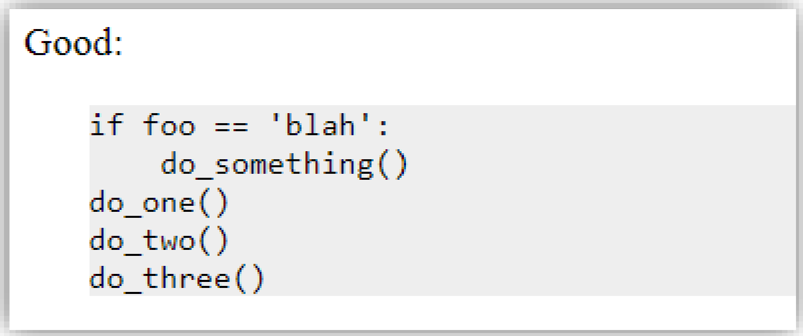
- Readability Counts
- Style Guide for Python Code (PEP 8)
- Watch your Whitespaces and Indentations
- Naming Conventions
- Practicality Beats Purity
- Be Consistent
- Let Python by Python

*PEP 8: http://www.python.org/dev/peps/pep-0257/

About Whitespaces

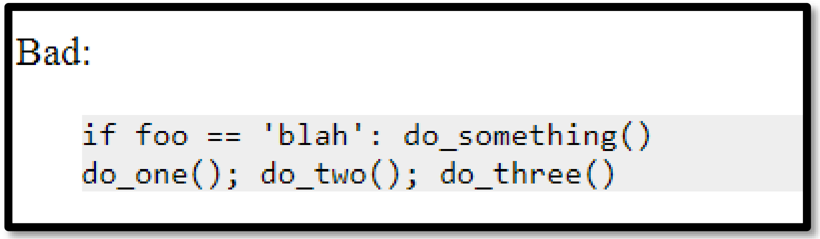
- **4 Spaces per Indentation Level**
- **Never Mix Tab and Spaces**
- **One Blank Line Between Functions**
- **Put spaces around assignments and comparisons**
- **No Spaces Just Inside Parentheses**

Dos and Don’ts – Compound Statements



Good:

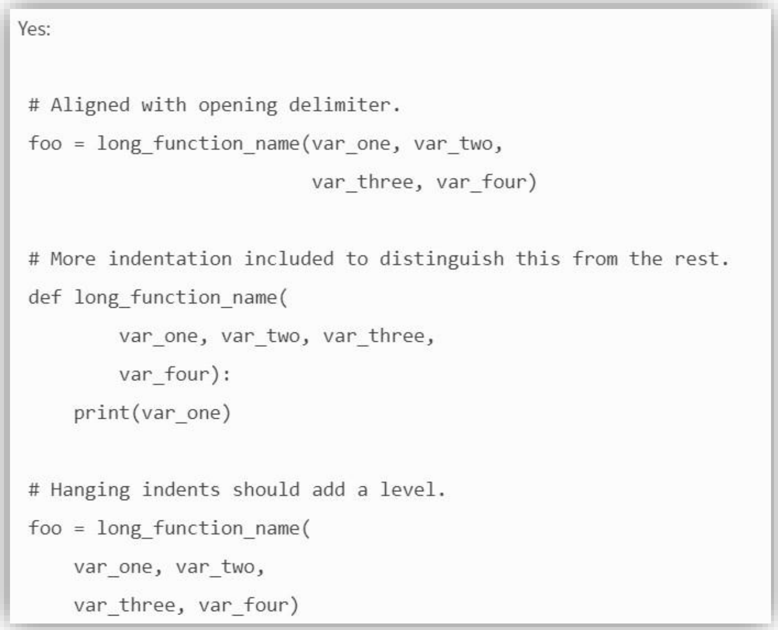
```
if foo == 'blah':
    do_something()
do_one()
do_two()
do_three()
```



Bad:

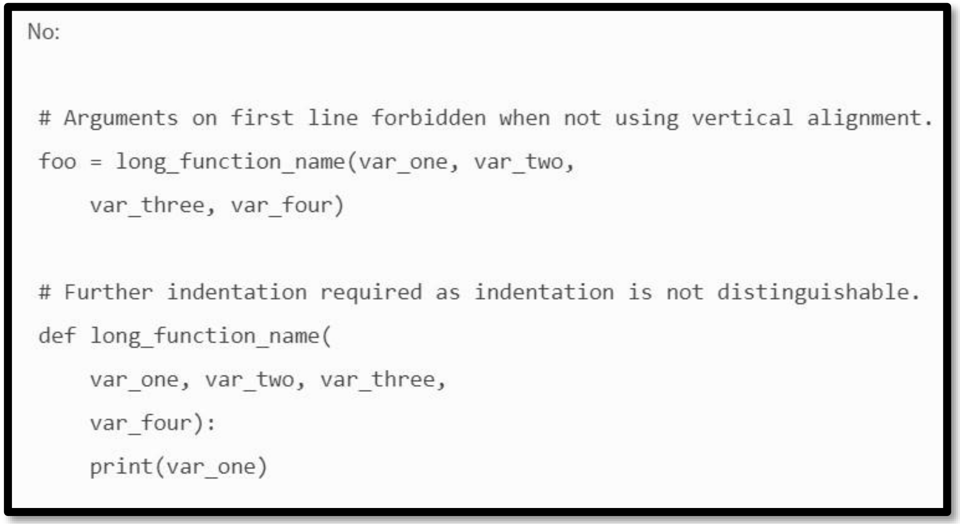
```
if foo == 'blah': do_something()
do_one(); do_two(); do_three()
```

Dos and Don’ts – Code Layout (Indentation)



Yes:

```
# Aligned with opening delimiter.
foo = long_function_name(var_one, var_two,
                         var_three, var_four)

# More indentation included to distinguish this from the rest.
def long_function_name(
        var_one, var_two, var_three,
        var_four):
    print(var_one)

# Hanging indents should add a level.
foo = long_function_name(
    var_one, var_two,
    var_three, var_four)
```



No:

```
# Arguments on first line forbidden when not using vertical alignment.
foo = long_function_name(var_one, var_two,
    var_three, var_four)

# Further indentation required as indentation is not distinguishable.
def long_function_name(
    var_one, var_two, var_three,
    var_four):
    print(var_one)
```

Style Guide Dos and Don’ts – Tabs and Spaces

- **Spaces** are the preferred indentation method.
- **Tabs** should be used solely to remain consistent with code that is already indented with tabs.
- **Python 3 disallows mixing** the use of tabs and spaces for indentation.
- Python 2 code indented with a mixture of tabs and spaces should be converted to using spaces exclusively.

Q & A

THANK YOU