NumPy

Python for Physics
Lab -2

Outline

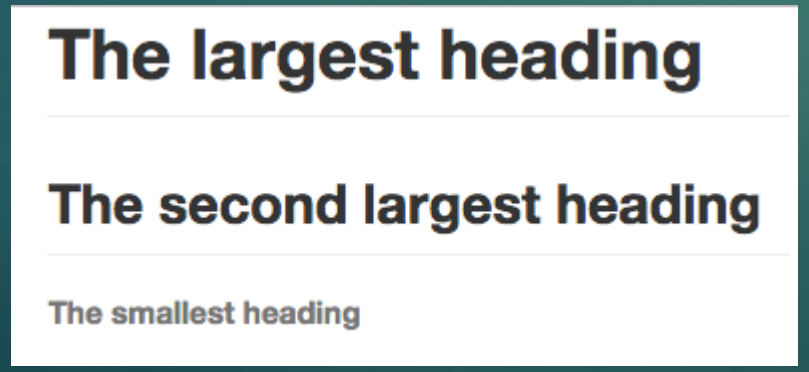
- Markdown
- Data Types
- Functions
- Python for loops

Markdown

Headings

To create a heading, add one to six `#` symbols before your heading text. The number of `#` you use will determine the size of the heading.

```
# The largest heading
## The second largest heading
###### The smallest heading
```



Text

Output	Syntax
emphasis	`*emphasis*`
strong	`**strong**`
`code`	`` `code` ``

```
Emphasis, aka italics, with *asterisks* or _underscores_.

Strong emphasis, aka bold, with **asterisks** or __underscores__.

Combined emphasis with **asterisks and _underscores_**.

Strikethrough uses two tildes. ~~Scratch this.~~
```

Emphasis, aka italics, with *asterisks* or *underscores*.

Strong emphasis, aka bold, with **asterisks** or **underscores**.

Combined emphasis with **asterisks and *underscores***.

Strikethrough uses two tildes. ~~Scratch this.~~

Styling text

You can indicate emphasis with bold, italic, or strikethrough text.

Style	Syntax	Keyboard shortcut	Example	Output
Bold	`** **` or `__ __`	command/control + b	`**This is bold text**`	**This is bold text**
Italic	`* *` or `_ _`	command/control + i	`*This text is italicized*`	*This text is italicized*
Strikethrough	`~~ ~~`		`~~This was mistaken text~~`	~~This was mistaken text~~
Bold and nested italic	`** **` and `_ _`		`**This text is _extremely_ important**`	**This text is *extremely* important**
All bold and italic	`*** ***`		`***All this text is important***`	***All this text is important***

Lists

Create an ordered list using numbers:

```
1. Number theory
2. Algebra
3. Partial differential equations
4. Probability
```

1. Number theory
2. Algebra
3. Partial differential equations
4. Probability

Create an unordered list using an asterisk * for each item:

```
* Number theory
* Algebra
* Partial differential equations
* Probability
```

- Number theory
- Algebra
- Partial differential equations
- Probability

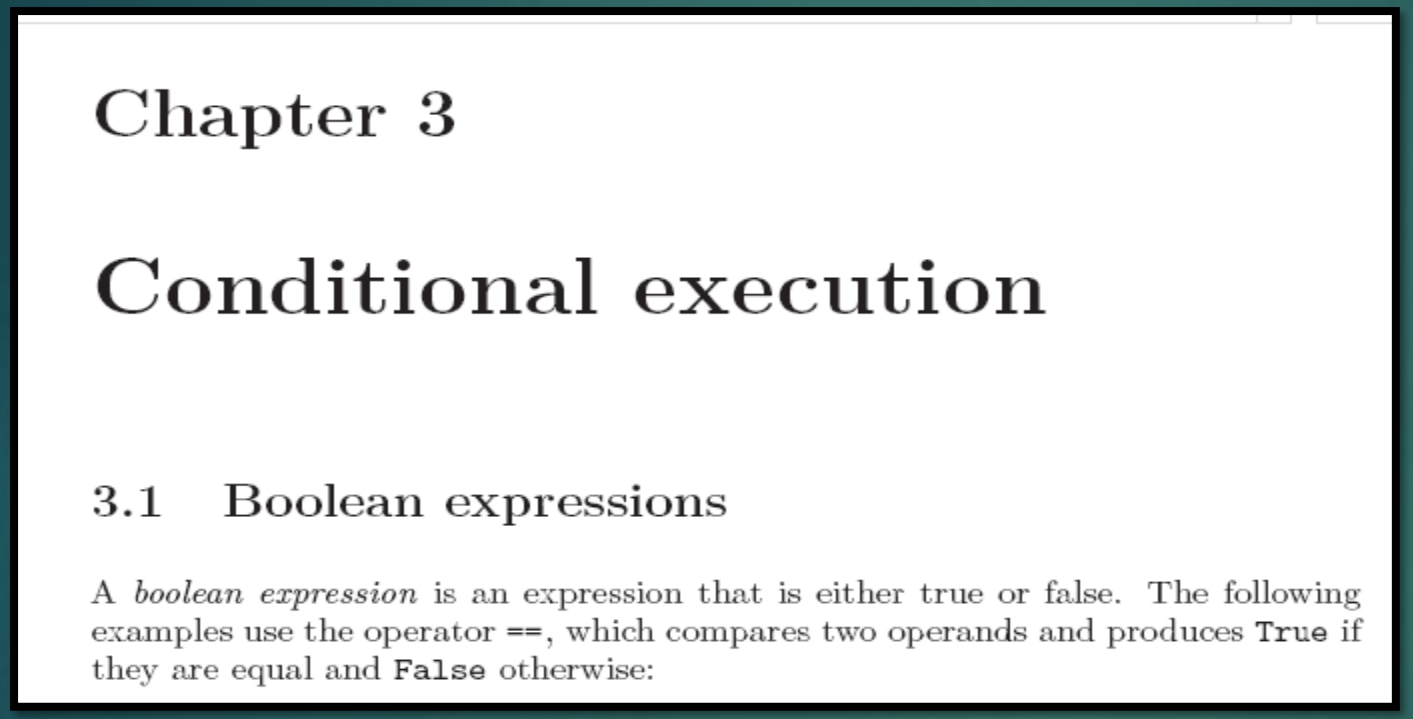
Use indentation to create nested lists:

```
1. Mathematics
  * Calculus
  * Linear Algebra
  * Probability
2. Physics
  * Classical Mechanics
  * Relativity
  * Thermodynamics
3. Biology
  * Diffusion and Osmosis
  * Homeostasis
  * Immunology
```

1. Mathematics
    - Calculus
    - Linear Algebra
    - Probability
2. Physics
    - Classical Mechanics
    - Relativity
    - Thermodynamics
3. Biology
    - Diffusion and Osmosis
    - Homeostasis
    - Immunology

Lab Task 1:

Prepare the code of the following three output on Jupyter notebook using markdown

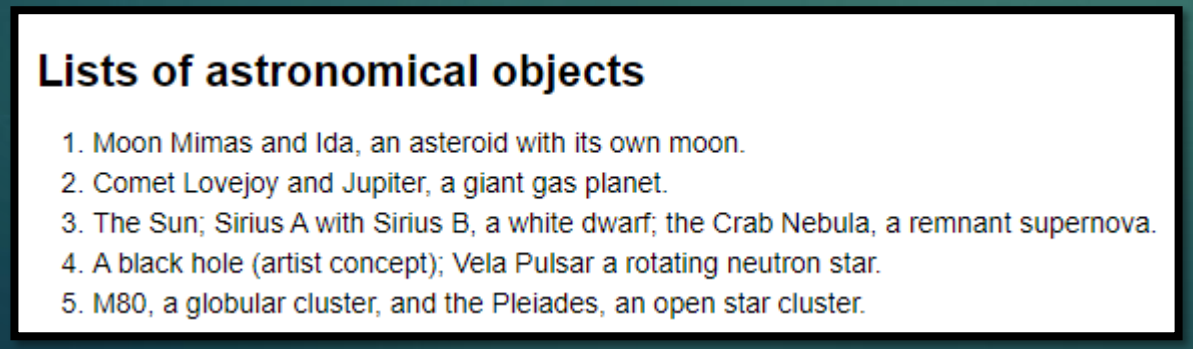


Chapter 3

Conditional execution

3.1 Boolean expressions

A *boolean expression* is an expression that is either true or false. The following examples use the operator ==, which compares two operands and produces `True` if they are equal and `False` otherwise:



Lists of astronomical objects

1. Moon Mimas and Ida, an asteroid with its own moon.
2. Comet Lovejoy and Jupiter, a giant gas planet.
3. The Sun; Sirius A with Sirius B, a white dwarf; the Crab Nebula, a remnant supernova.
4. A black hole (artist concept); Vela Pulsar a rotating neutron star.
5. M80, a globular cluster, and the Pleiades, an open star cluster.

Galaxy groups and clusters

- List of galaxy groups
- List of galaxy clusters
  - List of Abell clusters
- List of galaxy superclusters

Black holes

- Lists of black holes
  - List of black holes
  - List of most massive black holes
  - List of nearest black holes

Python Data Types

- Data types are nothing but variables you use to reserve some space in memory. Python variables do not need an explicit declaration to reserve memory space. The declaration happens automatically when you assign a value to a variable.

String data type

- String are identified as a contiguous set of characters represented in the quotation marks. Python allows for either pairs of single or double quotes. Strings are immutable sequence data type, i.e each time one makes any changes to a string, completely new string object is created.
-

```
In [6]: a_str = "Hello World"

In [7]: print (a_str)   #  Output will be whole string. Hello World
        Hello World

In [8]: print (a_str[0])  # Output will be first character. H
        H

In [11]: print (a_str[0:5])  # Output will be first five characters. Hello
         Hello

In [12]: print (a_str[3:])  # Output will be start from fourth characters till end . lo World
         lo World
```

String Operation

- Subsets of strings can be taken using the slice operator ([ ] and [:]) with indexes starting at 0 in the beginning of the string and working their way from -1 to the end.
- The plus (+) sign is the string concatenation operator and the asterisk (*) is the repetition operator.

```
In [4]: s1 = "Applied"
        s2 = "Physics"
        s1+s2

Out[4]: 'AppliedPhysics'

In [5]: print(s1 + " " + s2) # for space b/w s1 and s2

        Applied Physics

In [6]: s1[0] , s1[1]

Out[6]: ('A', 'p')

In [7]: s1[0:2], s1[3:]

Out[7]: ('Ap', 'lied')
```

```
In [8]: s1[0::+3] , s2[0::+2]
Out[8]: ('Ald', 'Pyis')

In [9]: s1[::-1] , s2[::-1]
Out[9]: ('deilppA', 'scisyhP')

In [10]: s3 = 'Applied'
         s1 == s2 , s1 == s3 , s2 ==s3
Out[10]: (False, True, False)
```

Number Data Types

- Numbers have four types in Python. Integer, float, complex, and long.

```
In [15]: int_num = 10          #integer  value
         print (int_num)

            10

In [16]: float_num = 10.2      #float value
         print (float_num)

            10.2

In [17]: complex_num = 3.14j   #complex value
         print (complex_num)

            3.14j

In [22]: long_num = 1234567435  #long value
         print (long_num)

            1234567435
```

Set Data Type

- A set is a collection of elements with no repeats and without insertion order but sorted order. They are used in situations where it is only important that some things are grouped together, and not what order they were included. For large groups of data, it is much faster to check whether or not an element is in a set than it is to do the same for a list.
- Sets are unordered collections of unique objects, there are two types of set:
  - Set
  - Frozen Set

Sets

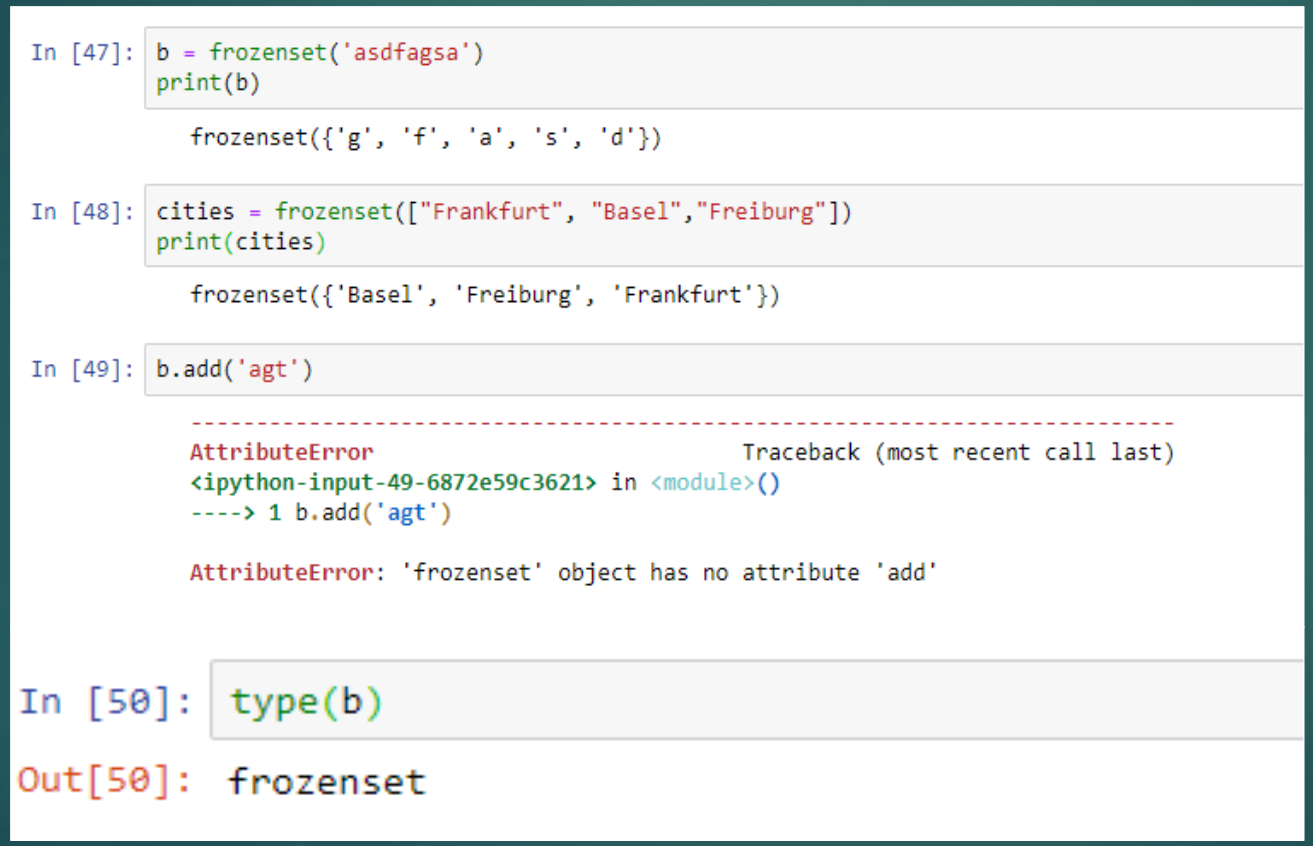
- They are mutable and new elements can be added once sets are defined

```
In [28]: basket = {'apple', 'orange', 'apple', 'pear', 'orange', 'banana'}
         print(basket)   # duplicates will be removed

         {'banana', 'pear', 'apple', 'orange'}

In [36]: a = set('abracadabra')
         print(a) # unique letters in a

         {'a', 'b', 'c', 'd', 'r'}

In [37]: a.add('z')
         print (a)

         {'a', 'b', 'c', 'd', 'r', 'z'}
```

```
In [51]: type(a)

Out[51]: set
```

Frozen Sets

- They are immutable and new elements cannot added after its defined.



```
In [47]: b = frozenset('asdfagsa')
         print(b)

         frozenset({'g', 'f', 'a', 's', 'd'})

In [48]: cities = frozenset(["Frankfurt", "Basel","Freiburg"])
         print(cities)

         frozenset({'Basel', 'Freiburg', 'Frankfurt'})

In [49]: b.add('agt')

         ---------------------------------------------------------------------------
         AttributeError                            Traceback (most recent call last)
         <ipython-input-49-6872e59c3621> in <module>()
         ----> 1 b.add('agt')

         AttributeError: 'frozenset' object has no attribute 'add'

In [50]: type(b)

Out[50]: frozenset
```

Boolean Data Type

```
In [11]: b1 = True
         b2 = False
         type(b1) , type(b2)

Out[11]: (bool, bool)

In [12]: zero_int = 0  #An int, float or complex number set to zero returns as False. An integer,
                       #float or complex number set to any other number, positive or negative, returns as True.
         bool(zero_int)

Out[12]: False

In [13]: pos_int = 1
         f = -0
         neg = -2.3
         bool(pos_int) , bool(s1) , bool(b1), bool(b2), bool(f), bool(neg)

Out[13]: (True, True, True, False, False, True)
```

List Data Type

- A list contains items separated by commas and enclosed within square brackets [ ]. Lists are almost similar to arrays in C. One difference is that all the items belonging to a list can be of different data type.

```
In [17]: list1 = ["physics", "Chemistry", "Math", "Statistics"]   # indexing strat from  0 and then , 1, 2, 3
         list1[0] , list1[3], list1[3]

Out[17]: ('physics', 'Statistics', 'Statistics')

In [18]: list1[2:] , list1[:2] , list1[:], list1[-3:], list1[:-3]

Out[18]: (['Math', 'Statistics'],
          ['physics', 'Chemistry'],
          ['physics', 'Chemistry', 'Math', 'Statistics'],
          ['Chemistry', 'Math', 'Statistics'],
          ['physics'])
```

- Lists are mutable:

```
In [639]: list1[2] = 22

In [640]: list1
Out[640]: ['physics', 'Chemistry', 22, 'Statistics']

In [641]: type(list1[2])
Out[641]: int

In [642]: list1
Out[642]: ['physics', 'Chemistry', 22, 'Statistics']
```

- Appending to a list using “ append and extent”

```
In [643]: list2 = [1,2,3]
          list1.extend(list2)
          list1

Out[643]: ['physics', 'Chemistry', 22, 'Statistics', 1, 2, 3]

In [644]: list1.append(list2)
          list1

Out[644]: ['physics', 'Chemistry', 22, 'Statistics', 1, 2, 3, [1, 2, 3]]

In [645]: list1.extend("ITC")
          print(list1)

          ['physics', 'Chemistry', 22, 'Statistics', 1, 2, 3, [1, 2, 3], 'I', 'T', 'C']

In [646]: list1.append("Islamiat")
          print (list1)

          ['physics', 'Chemistry', 22, 'Statistics', 1, 2, 3, [1, 2, 3], 'I', 'T', 'C', 'Islamiat']

In [647]: list1.append(list2)
          print (list1)

          ['physics', 'Chemistry', 22, 'Statistics', 1, 2, 3, [1, 2, 3], 'I', 'T', 'C', 'Islamiat', [1, 2, 3]]

In [648]: type(list1[-1])

Out[648]: list
```

Tuple Data Type

- Lists are enclosed in brackets [ ] and their elements and size can be changed, while tuples are enclosed in parentheses ( ) and cannot be updated. Tuples are immutable.

- Tuples are immutable

```
In [657]: tuple1 = ('AP', 'PF','Eng')
          tuple1

Out[657]: ('AP', 'PF', 'Eng')

In [658]: tuple1[2]

Out[658]: 'Eng'

In [659]: tuple1[0] = "CP"

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-659-21558ea95455> in <module>()
----> 1 tuple1[0] = "CP"

TypeError: 'tuple' object does not support item assignment
```

Dictionary Data Type

- A **defaultdict** is a dictionary with a default value for keys, so that keys for which no value has been explicitly defined can be accessed without errors. A **defaultdict** is especially useful when the values in the dictionary are collections (lists, dicts, etc) in the sense that it does not need to be initialized every time when a new key is used.
- A **defaultdict** will never raise a **KeyError**. Any key that does not exist gets the default value returned.
- Dictionary consists of key-value pairs. It is enclosed by curly braces { } and values can be assigned and accessed using square brackets [ ].

```
In [78]: dic={'name':'Redrose','age':10}
         print(dic) # will output all the key-value pairs. {'name':'Redrose','age':10}

         {'name': 'Redrose', 'age': 10}

In [79]: print(dic['name']) # will output only value with 'name' key. 'Redrose'

         Redrose

In [80]: print(dic.values()) #will output list of values in dic. ['Redrose',10]

         dict_values(['Redrose', 10])

In [81]: print(dic.keys()) #will output list of keys. ['name','age']

         dict_keys(['name', 'age'])
```

Functions

- A function is a structure that you define. You get to decide if they have arguments or not. You can add keyword arguments and default arguments too. A function is a block of code that starts with the def keyword, a name for the function and a colon. Here's a simple example:

```
In [102]: def a_function():
              print("You just created a function!")

In [105]: a_function()
          You just created a function!
```

Passing Arguments to a function

- Now we're ready to learn about how to create a function that can accept arguments and also learn how to pass said arguments to the function. Let's create a simple function that can add two numbers together

```
In [109]: def add(a, b):
              return a + b

In [110]: add(4,7)

Out[110]: 11
```

Python for loops

- A for loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string).

Example:

- Print each fruit in a fruit list

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  print(x)
```

Examples:

Example 1

- Write a Python program which accepts the radius of a circle from the user and compute the area.

Example 1

- Write a Python program which accepts the radius of a circle from the user and compute the area.

```
In [1]: from math import pi
        r = float(input ("Input the radius of the circle : "))
        print ("The area of the circle with radius " + str(r) + " is: " + str(pi * r**2))

        Input the radius of the circle : 2
        The area of the circle with radius 2.0 is: 12.566370614359172
```

Example 2:

- Write a Python program which accepts a sequence of comma-separated numbers from user and generate a list and a tuple with those numbers.

Example 2:

- Write a Python program which accepts a sequence of comma-separated numbers from user and generate a list and a tuple with those numbers.

```
In [2]: values = input("Input some comma seprated numbers : ")
        list = values.split(",")
        tuple = tuple(list)
        print('List : ',list)
        print('Tuple : ',tuple)
```

```
Input some comma seprated numbers : 2,3,6,7
List :  ['2', '3', '6', '7']
Tuple :  ('2', '3', '6', '7')
```

Lab Task 2:

a) Write a Python program to display the first and last colors from the following list. color_list = ["Red","Green","White" ,"Black"]

b) Write a Python program that accepts an integer (n) and computes the value of n+nn+nnn.

c) Write a Python function to sum all the numbers in a list.