Foundations and Programming for Data Analytics

Week 5 Material

BASICS OF PYTHON PROGRAMMING

- Lists and Tuples
- Modules
- Args and Kwargs

LITERALS COLLECTIONS

- **list**
- **tuple**
- dictionary
- sets

List

- It is a list of elements represented in square brackets with commas in between.
- These variables can be of any data type and can be changed as well.

Example

```
# List

my_list = [23, "geek", 1.2, 'data']
```

Tuple

- It is also a list of comma-separated elements or values in round brackets.
- The values can be of any data type but can’t be changed.

Example

```
# Tuple
my_tuple = (1, 2, 3, 'hello')
```

Dictionary

- It is the unordered set of key-value pairs.

Example

```
# Dictionary
my_dict = {1:'one', 2:'two', 3:'three'}
```

Sets

- It is the unordered collection of elements in curly braces '{}'.

Example

```
# Set
my_set = {1, 2, 3, 4}
```

Comparison between list and tuple

List	**Tuple**

Creating list
`# Creating list - employee id, name, age, position, salary`
`emp_list=[89,'Karan',45,'Lecturer',5500.75]`
`emp_list`
Out
	Creating tuple
	`# Creating tuple - employee id, name, age, position, salary`
	`emp_tuple=(89,'Karan',45,'Lecturer',5500.75)`
	`emp_tuple`
	Out

Find type of data structure using type() function	

`# type() function - List`
`type(emp_list)`
Out
	`# type() function - List`
	`type(emp_tuple)`
	Out

Printing employee detail using for loop	
Code **Out**	**Code** **Out**
Access Items (Using indexing)	
`[89, 'Karan', 45, 'Lecturer', 5500.75]` 0 1 2 3 4 `emp_list [0]` 89 `emp_list [1]` 'Karan' `emp_list [2]` 45 `emp_list [3]` 'Lecturer' `emp_list [4]` 5500.75	`(89, 'Karan', 45, 'Lecturer', 5500.75)` 0 1 2 3 4 `emp_tuple [0]` 89 `emp_tuple [1]` 'Karan' `emp_tuple [2]` 45 `emp_tuple [3]` 'Lecturer' `emp_tuple [4]` 5500.75

# print employee details - list	# print employee details - tuple
Code **Out** Method 2: (using for loop) **Code** **Out**	

Access Items (Using negative indexing)

```
[89, 'Karan', 45, 'Lecturer', 5500.75]
 -5     -4     -3     -2          -1
```

emp_list [-5]	89
emp_list [-4]	'Karan'
emp_list [-3]	45
emp_list [-2]	'Lecturer'
emp_list [-1]	5500.75

# print employee name and salary (Using negative indexing)

Code

Out

```
(89, 'Karan', 45, 'Lecturer', 5500.75)
 -5     -4     -3     -2          -1
```

emp_tuple [-5]	89
emp_tuple [-4]	'Karan'
emp_tuple [-3]	45
emp_tuple [-2]	'Lecturer'
emp tuple [-1]	5500.75

# print employee name and salary (Using negative indexing)

Code

Out

Modify an item	
# Change the salary into 5665 **Code** **Out** **Mutable List**	# Change the salary into 5665 **Code** **Out** **Immutable Tuples**

How to change tuple values?

1. Convert the tuple into a list.
2. Assign the value to be changed
3. Again, convert the list into the tuple

In the `emp_tuple`, change the salary into 5665

Code

Out

LIST

```
# List creation
my_list_1 = [1, 2, 3, 4]
my_list_1
```
Out

```
type(my_list_1)
```
Out

```
my_list_2 = [1, 2.4, 'a string', ['a string in another list', 5]]
my_list_2
```
Out

```
my_list_3 = [2+3, 5*3, 4**2]
my_list_3
```
Out

```
my_str = 'A string.'
list(my_str)
```
Out

# List operators

```
# '+' means list concatenation
[1, 2, 3] + [4, 5, 6]
```

Out

```
# '*' means list replication and concatenation
[1, 2, 3] * 3
```

Out

```
# Membership operators: 'in' and 'not in'

my_list_2 = [1, 2.4, 'a string', ['a string in another list', 5]]

1 in my_list_2
```

Out

```
['a string in another list', 5] in my_list_2
```

Out

```
's string in another list' in my_list_2
```

Out

```
['a string in another list', 7] in my_list_2
```

Out

```
7 not in my_list_2
```

Out

<u># List indexing. Indexing in Python starts at zero.</u>

```
my_list = [1, 2.4, 'a string', ['a string in another list', 5]]
my_list[1]
```

Out

```
my_list = [1, 2.4, 'a string', ['a string in another list', 5]]
my_list[1]
```

Out

```
print(my_list[0])
print(my_list[1])
print(my_list[2])
print(my_list[3])
```

Out

```
my_list[3][0]
```
Out

```
my_list = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
my_list[5]
```
Out

```
my_list = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
my_list[0]
```
Out

```
my_list = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
my_list[11]
```
Out

```
my_list[-1]
```
Out

```
my_list[-3]
```
Out

# List slicing

```
my_list[0:5]
```
Out

```
my_list[3:1000]
```
Out

```
my_list[0:-3]
```

Out

<u>`# to list down even numbers from the given list`</u>

```
my_list[0::2]
```

Out

```
my_list[::2]
```

Out

<u>`# use negative stride to reverse the list.`</u>

```
my_list[::-1]
```

Out

```
print(my_list[2::2])
```

Out

```
print(my_list[2:-1:2])
```

Out

```
print(my_list[-2::-2])
```

Out

```
print(my_list[-2:2:-2])
```

Out

```
print(my_list[2:2:-2])
```

Out

```
# Indexing scheme can be conclude as my_list[start:end:stride]

# Mutability: you can change list values without creating a new list.

my_list = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Out

```
my_list[3] = 'four'

my_list
```

Out

```
my_list[3] = 'apple'
```

```
my_list
```

Out

TUPLES

# Tuple creation

```
my_tuple = (0,)
not_a_tuple = (0) # this is just the number zero (normal use of parentheses)
type(my_tuple), type(not_a_tuple)
```

Out

```
# Convert a list to a tuple
my_list = [1, 2.4, 'a string', ['a string in another list', 5]]
my_tuple = tuple(my_list)
my_tuple
```

Out

```
# change the item in the list
my_tuple[3][0] = 'a string in a list in a tuple'
my_tuple
```

Out

```
# change the item in the tuple
my_tuple[1] = 7
```
Out

```
# Slicing of tuples
my_tuple = (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
```
Out

```
# Reverse
my_tuple[::-1]
```
Out

```
# Odd numbers
my_tuple[1::2]
```
Out

```
# The '+' operator with tuples
my_tuple + (11, 12, 13, 14, 15)
```
Out

```
emp_tuple=(89,'Karan',45,'Lecturer',5500.75)
emp_tuple + (100,)
```
Out

```
# Can we concatenate/'+' tuple and list?
my_tuple + my_list
```

Out

```
# Membership operators
5 in my_tuple
```
Out

```
'Fifi' not in my_tuple
```
Out

```
0 not in my_tuple
```
Out

```
# Tuple unpacking
my_tuple = (1, 2, 3)
(a, b, c) = my_tuple
a
```
Out

```
b
```
Out

```
c
```
Out

```
d
```

Out

```
# Parentheses are dispensable
a, b, c = my_tuple
print(a, b, c)
```

Out

Available Operations

List

```
dir (emp_list)
```

Out

```
dir (emp_tuple)
```

Out

List Methods

Method	Description	Syntax
append()	Adds an element at the end of the list	`list.append(elmnt)` elmnt Required. An element of any type (string, number, object etc.)
clear()	Removes all the elements from the list	`list.clear()` No parameters
copy()	Returns a copy of the list	`list.copy()` No parameters
count()	Returns the number of elements with the specified value	`list.count(value)` value Required. Any type (string, number, list, tuple, etc.). The value to search for.
extend()	Add the elements of a list (or any iterable), to the end of the current list	`list.extend(iterable)` iterable Required. Any iterable (list, set, tuple, etc.)
index()	Returns the index of the first element with the specified value	`list.index(elmnt)` elmnt Required. Any type (string, number, list, etc.). The element to search for
insert()	Adds an element at the specified position	`list.insert(pos, elmnt)` pos Required. A number specifying in which position to insert the value elmnt Required. An element of any type (string, number, object etc.)
pop()	Removes the element at the specified position	`list.pop(pos)` pos Optional. A number specifying the position of the element you want to remove, default value is -1, which returns the last item
remove()	Removes the item with the specified value	`list.remove(elmnt)` elmnt Required. Any type (string, number, list etc.) The element you want to remove
reverse()	Reverses the order of the list	`list.reverse()` No parameters
sort()	Sorts the list	`list.sort(reverse=True|False, key=myFunc)` reverse Optional. reverse=True will sort the list descending. Default is reverse=False key Optional. A function to specify the sorting criteria(s)

List methods

Append Items

To add an item to the end of the list, use the `append()` method:

```
thislist = ["apple", "banana", "cherry"]
thislist.append("orange")
print(thislist)
```

Out

Insert Items

To insert a list item at a specified index, use the `insert()` method.

The insert() method inserts an item at the specified index:

```
thislist = ["apple", "banana", "cherry"]
thislist.insert(1, "orange")
print(thislist)
```

Out

Extend List

To append elements from another list to the current list, use the `extend()` method.

```
thislist = ["apple", "banana", "cherry"]
tropical = ["mango", "pineapple", "papaya"]
thislist.extend(tropical)
print(thislist)
```

Out

Add Any Iterable

The `extend()` method does not have to append lists, you can add any iterable object (tuples, sets, dictionaries etc.).

```
thislist = ["apple", "banana", "cherry"]
thistuple = ("kiwi", "orange")
thislist.extend(thistuple)
print(thislist)
```

Out

Remove Specified Item

The `remove()` method removes the specified item.

```
thislist = ["apple", "banana", "cherry"]
thislist.remove("banana")
print(thislist)
```

Out

If there are more than one item with the specified value, the `remove()` method removes the first occurance:

```
thislist = ["apple", "banana", "cherry", "banana", "kiwi"]
thislist.remove("banana")
print(thislist)
```

Out

Remove Specified Index

The `pop()` method removes the specified index.

```
thislist = ["apple", "banana", "cherry"]
thislist.pop(1)
print(thislist)
```

Out

If you do not specify the index, the `pop()` method removes the last item.

```
thislist = ["apple", "banana", "cherry"]
thislist.pop()
print(thislist)
```

Out

Clear the List

The clear() method empties the list.

The list still remains, but it has no content.

```
thislist = ["apple", "banana", "cherry"]
thislist.clear()
print(thislist)
```

Out

```
fruits = ["apple", "banana", "cherry", "kiwi", "mango"]
newlist = []

for x in fruits:
  if "a" in x:
    newlist.append(x)

print(newlist)
```

Sort List Alphanumerically

List objects have a sort() method that will sort the list alphanumerically, ascending, by default:

Sort the list alphabetically:

```
thislist = ["orange", "mango", "kiwi", "pineapple", "banana"]
thislist.sort()
print(thislist)
```

Out

Sort the list numerically:

```
thislist = [100, 50, 65, 82, 23]
thislist.sort()
print(thislist)
```

Out

Sort Descending

To sort descending, use the keyword argument `reverse = True`:

```
thislist = ["orange", "mango", "kiwi", "pineapple", "banana"]
thislist.sort(reverse = True)
print(thislist)
```

Out

Copy a List

You cannot copy a list simply by typing list2 = list1. There are ways to make a copy, one way is to use the built-in List method `copy()`.

Make a copy of a list with the `copy()` method:

```
thislist = ["apple", "banana", "cherry"]
mylist = thislist.copy()
print(mylist)
```

Out

Another way to make a copy is to use the built-in method `list()`.

```
thislist = ["apple", "banana", "cherry"]
mylist = list(thislist)
print(mylist)
```

Out

Count the given element

The `count()` method returns the number of elements with the specified value.

Return the number of times the value "cherry" appears in the fruits list:

```
fruits = ['apple', 'banana', 'cherry']
x = fruits.count("cherry")
```

Out

Return the number of times the value 9 appears int the list:

```
points = [1, 4, 2, 9, 7, 8, 9, 3, 1]
x = points.count(9)
```

Out

The index of the first element

The `index()` method returns the position at the first occurrence of the specified value.

What is the position of the value 32:

```
fruits = [4, 55, 64, 32, 16, 32]
x = fruits.index(32)
```

Out

Tuple Methods

Method	Description	syntax
count()	Returns the number of times a specified value occurs in a tuple	`tuple.count(value)` *value Required. The item to search for*
index()	Searches the tuple for a specified value and returns the position of where it was found	`tuple.index(value)` *value Required. The item to search for*

Tuple `count()` Method

The `count()` method returns the number of times a specified value appears in the tuple.

Return the number of times the value 5 appears in the tuple:

```
thistuple = (1, 3, 7, 8, 7, 5, 4, 6, 8, 5)
x = thistuple.count(5)
print(x)
```

Out

Tuple `index()` Method

The `index()` method finds the first occurrence of the specified value.

Search for the first occurrence of the value 8, and return its position:

```
thistuple = (1, 3, 7, 8, 7, 5, 4, 6, 8, 5)
x = thistuple.index(8)
print(x)
```

Out

Summary

List Items

List items are ordered, changeable, and allow duplicate values.

Tuple Items

Tuple items are ordered, unchangeable, and allow duplicate values.

Quiz

1. Which of the following is not a property of a list?
   a) Ordered
   b) Mutable
   c) Contain only same type of elements
   d) Can contain duplicate values

2. Which of the following functions cannot be used to add an element to the list?
   a) add()
   b) insert()
   c) append()
   d) extend()

3. What is the output of the following code?

```
list1=[1,’a’,5.4]
list1.extend([-3,0.8])
list1
```

a) [1, 'a', 5.4, [-3, 0.8]]
b) [1, 'a', 5.4, -3, 0.8]
c) [[1, 'a', 5.4],[-3, 0.8]]
d) None of the above

4.
```
list1=[9,8,7,6,5,4,3,2,1]
 list1.pop(4)
```

a) 4
b) 5
c) 6
d) None

5. Which of the following is the show error on using sort() function?

a) list1=[1,2.5,-6,0.7]
b) list2=[1,'a',5.6,9]
c) list3=['a','R','u','H']
d) All the above

6. Which of the following gives the output as [2,3,2,3,4,5] If list1=[2,3] and list2=[4,5]
a) list1+list2*2
b) (list1*2)+list2
c) list2+list1*2
d) Both b and c

7. Which of the following is the show error on using sort() function?

a) list1=[1,2.5,-6,0.7]
b) list2=[1,'a',5.6,9]
c) list3=['a','R','u','H']
d) All the above

8. Which of the following is/are immutable in Python?

a) Tuples
b) Dictionaries
c) Sets
d) List

9. What will be the output of the following code?

```
var=(4)
print(type(4))
```

a) <class 'list'>
b) <class 'int'>
c) <class 'tuple'>
d) None of the above

10. What will be the output of the following code?

```
tup=(1,2,3,1,1.0,4)
print(tup.count(1))
```

a) 2
b) 3
c) TypeError
d) ValueError

Arguments

- Information can be passed into functions as arguments.
- Arguments are specified after the function name, inside the parentheses. You can add as many arguments as you want, just separate them with a comma.

Types of Python Function Arguments

- Positional arguments
- Default argument
- **Keyword arguments (named arguments)**
- **Arbitrary arguments (variable-length arguments *args and **kwargs)**

Keyword Arguments

- You can also send arguments with the `key = value` syntax.
- This way the order of the arguments does not matter.
- The phrase Keyword Arguments are often shortened to kwargs in Python documentations.

```
def participation (name2, name1, name3):
  print("The WINNER is " + name2)

participation (name1 = "Emil", name2 = "Lim", name3 = "Nur")
```

Out

Arbitrary arguments (variable-length arguments *args and **kwargs)

- Args will help to pass multiple arguments, and
- Kwargs will allow us to pass keyword arguments to a function.
- *args receives arguments as a tuple.
- **kwargs receives arguments as a dictionary.

Arbitrary Arguments
***args**

Arbitrary Arguments are often shortened to *args in Python documentations.
If you do not know how many arguments that will be passed into your function, add a * before the parameter name in the function definition.
This way the function will receive a tuple of arguments, and can access the items accordingly:

```
def participation (*name):
  print("The WINNER is " + name[2})

participation ("Emil", "Lim", "Nur")
participation (“Jo”, "Joan", "Lee", "Nur")
```

Out

Arbitrary Keyword Arguments
****kwargs**

If you do not know how many keyword arguments that will be passed into your function, add two asterisk: ** before the parameter name in the function definition.

This way the function will receive a dictionary of arguments, and can access the items accordingly:

```
def kid_name(**kid):
   print("The last name is " + kid["lname"])

kid_name(fname = "Eben", lname = "Christy")
```

Out

Revision

Identify the types of python function arguments

1.

```
def get_net_price(price, discount):
    return price * (1-discount)

net_price = get_net_price(100, 0.1)
print(net_price)
```

Type of python function arguments

2.

```
def get_net_price(price, discount):
    return price * (1-discount)

net_price = get_net_price(discount=0.1, price=100)
print(net_price)
```

Type of python function arguments

3.

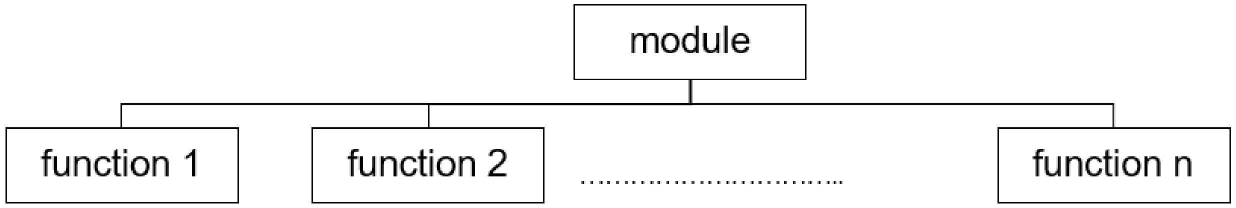
```
def get_net_price(price, tax=0.07, discount=0.05):
    return price * (1 + tax - discount)

net_price = get_net_price(100)
print(net_price)
```

Types of python function arguments

MODULES

What is a Module?

A module is a file with the extension .py and contains executable Python code.
A module contains a set of functions.
It is same as a code library.



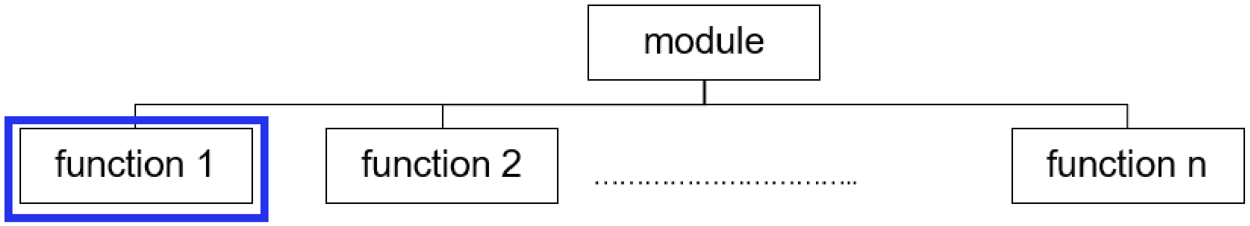
Example 1: Module name: `numpy`

- `array`
- `random`

Example2 : Module name: `pandas`

- `DataFrame`
- `read_csv`
- `dropna()`
- `fillna`
- `duplicated()`
- `drop_duplicates()`
- `corr()`

Refer more: https://www.w3schools.com/python/pandas/pandas_cleaning.asp

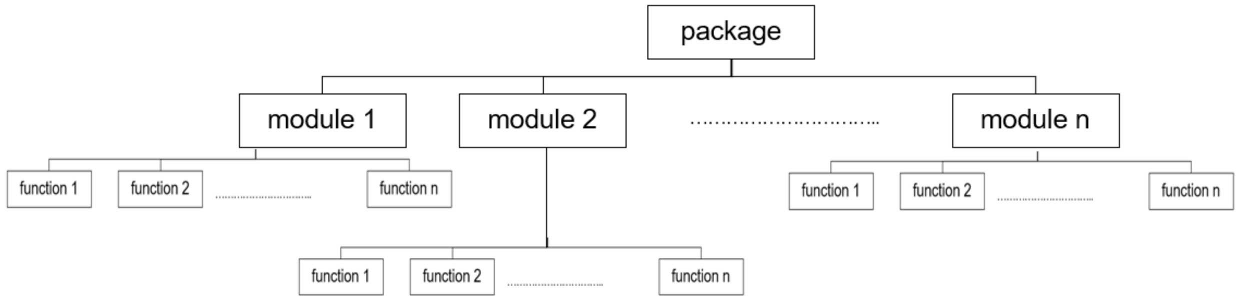


How to access function 1?

```
import module
module.function1
```

PACKAGES

A Python package contains one or more modules.
Each package has many functions.
Packages organize modules in the hierarchical structure.



Example 1: `matplotlib`

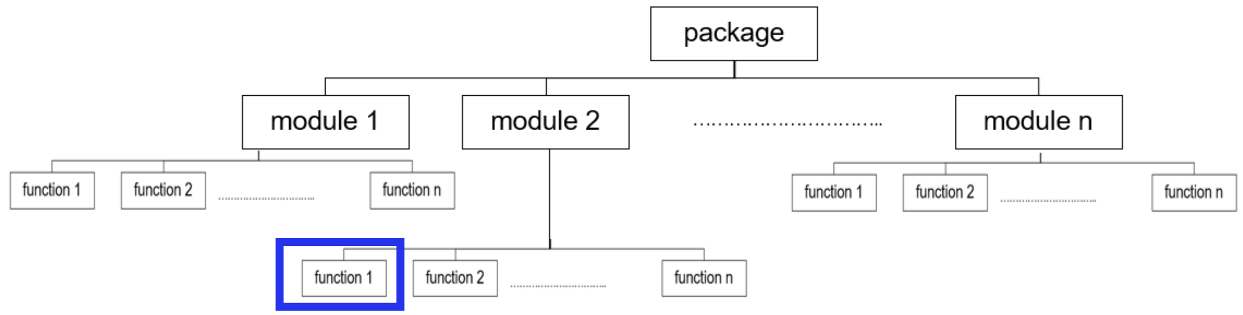
`pyplot`

- `bar()` - Create a Vertical bar plot
- `barh()` - Create a Horizontal bar plot
- `plot()` - Create a line chart.

- `scatter()` – plot a dot

✓ Bar charts are used when a variable is qualitative and takes a number of discrete values.
✓ A line chart can be used to compare values between groups across time by plotting one line per group.
✓ The scatter plot is the standard way of showing the relationship between two variables.

Example 2: `bokeh`
`plotting`
- `figure`
- `show`



How to access function 1?
```
import package.module2
package.module2.function1
```

MODULES

There are two types modules

- Built-in-modules
- User-defined-modules

Built-in-modules

Modules that are pre-defined are called built-in modules. We don’t have to create these modules in order to use them. All built-in modules are available in the Python Standard Library.

Few examples of built-in modules are:

- math module
- random module
- datetime module

For data analysis

- Pandas
- NumPy

User-defined Modules in Python

User-defined modules are modules that we create. These are custom-made modules created specifically to cater to the needs of a certain project.

Built-in-modules

<u>For data analysis</u>

- **Pandas**
- NumPy

Pandas

- Import Pandas
- Read CSV Files
- Data Cleaning

<u>Import Pandas</u>

```
import pandas as pd
```

<u>Read CSV Files</u>

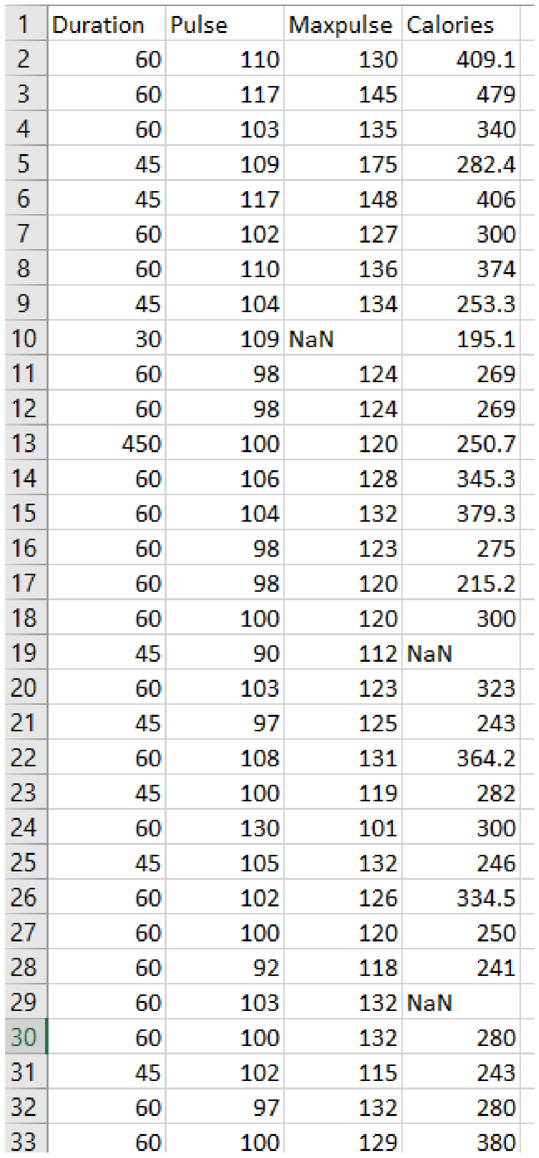
```
df= pd.read_csv("data.csv")
```

<u>Data Cleaning</u>

Data cleaning means fixing bad data in your data set.
Bad data could be:

- Empty cells
- Data in wrong format
- Wrong data
- Duplicates

Data.csv



1	Duration	Pulse	Maxpulse	Calories
2	60	110	130	409.1
3	60	117	145	479
4	60	103	135	340
5	45	109	175	282.4
6	45	117	148	406
7	60	102	127	300
8	60	110	136	374
9	45	104	134	253.3
10	30	109	NaN	195.1
11	60	98	124	269
12	60	98	124	269
13	450	100	120	250.7
14	60	106	128	345.3
15	60	104	132	379.3
16	60	98	123	275
17	60	98	120	215.2
18	60	100	120	300
19	45	90	112	NaN
20	60	103	123	323
21	45	97	125	243
22	60	108	131	364.2
23	45	100	119	282
24	60	130	101	300
25	45	105	132	246
26	60	102	126	334.5
27	60	100	120	250
28	60	92	118	241
29	60	103	132	NaN
30	60	100	132	280
31	45	102	115	243
32	60	97	132	280
33	60	100	129	380

Empty cells

Wrong data

Duplicates

Run the code in Jupyter notebook

Duplicates

Discovering Duplicates

```
df.duplicated()

df.duplicated().sum()
```

Removing Duplicates

```
df.drop_duplicates(inplace = True)
```

Empty Cells

Empty cells can potentially give you a wrong result when you analyze data.

```
df.isnull()

df.isnull().sum()
```

Remove Rows

One way to deal with empty cells is to remove rows that contain empty cells.
This is usually OK, since data sets can be very big, and removing a few rows will not have a big impact on the result.

```
df.dropna()
```

Note: By default, the `dropna()` method returns a new DataFrame, and will not change the original.

```
df.dropna(inplace = True)
```

Note: Now, the `dropna(inplace = True)` will NOT return a new DataFrame, but it will remove all rows containing NULL values from the original DataFrame.

Replace Empty Values

The `fillna()` method allows us to replace empty cells with a value:

```
df = pd.read_csv('data.csv')
df["Calories"].fillna(300, inplace = True)
```

The example above replaces all empty cells in the whole Data Frame.

Replace Only For Specified Columns

A common way to replace empty cells, is to calculate the mean, median or mode value of the column.
Mean = the average value (the sum of all values divided by number of values).
Median = the value in the middle, after you have sorted all values ascending.
Mode = the value that appears most frequently.

```
# Replace Using Mean
df = pd.read_csv('data.csv')
x = df["Calories"].mean()
df["Calories"].fillna(x, inplace = True)

# Replace Using Median
df = pd.read_csv('data.csv')
x = df["Calories"].median()
df["Calories"].fillna(x, inplace = True)

# Replace Using Mode
df = pd.read_csv('data.csv')
x = df["Calories"].mode()[0]
df["Calories"].fillna(x, inplace = True)
```

Wrong Data

"Wrong data" does not have to be "empty cells" or "wrong format", it can just be wrong, like if someone registered "199" instead of "1.99".

Sometimes you can spot wrong data by looking at the data set, because you have an expectation of what it should be.

If you take a look at our data set, you can see that in row 13, the duration is 450, but for all the other rows the duration is between 30 and 60.

It doesn't have to be wrong, but taking in consideration that this is the data set of someone's workout sessions, we conclude with the fact that this person did not work out in 450 minutes.

Replacing Values

One way to fix wrong values is to replace them with something else.

```
df.loc[11, 'Duration'] = 45
```

Loop through all values in the "Duration" column.
If the value is higher than 60, set it to 60:

```
df = pd.read_csv('data.csv')
for x in df.index:
  if df.loc[x, "Duration"] > 60:
    df.loc[x, "Duration"] = 60
```

Removing Rows

Another way of handling wrong data is to remove the rows that contains wrong data.

This way you do not have to find out what to replace them with, and there is a good chance you do not need them to do your analyses.

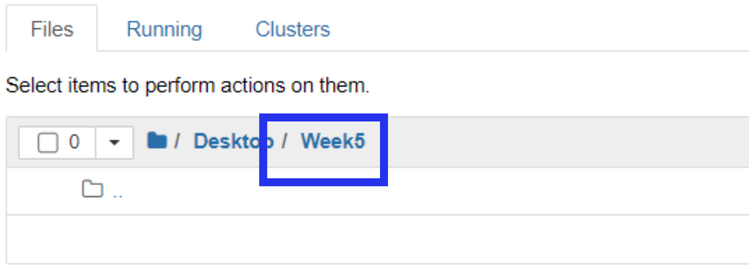
```
df = pd.read_csv('data.csv')
for x in df.index:
  if df.loc[x, "Duration"] > 60:
    df.drop(x, inplace = True)
```

Clean Dataset

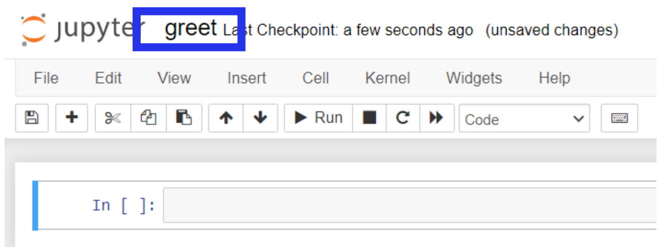
```
# Import Python Libraries
import pandas as pd

# Reading Dataset
df = pd.read_csv('data.csv')

# Removing Duplicates
df.drop_duplicates(inplace = True)

# Empty Cells - Replace Only For Specified Columns
x = df["Calories"].mean()
df["Calories"].fillna(x, inplace = True)
x = df["Maxpulse"].mode()[0]
df["Maxpulse"].fillna(x, inplace = True)

# Wrong Data - Replacing Values
for x in df.index:
  if df.loc[x, "Duration"] > 60:
    df.loc[x, "Duration"] = 60

df
```

User-defined-modules

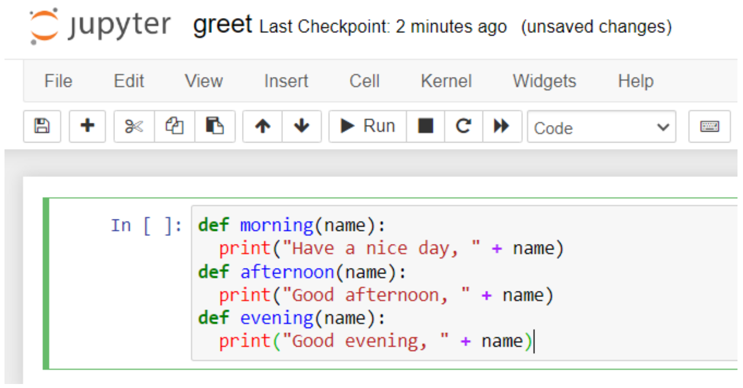
1. Create a folder `Week 5` and open the folder.



2. Create a new jupyter notebook `greet` (`greet` is a module) in the folder.

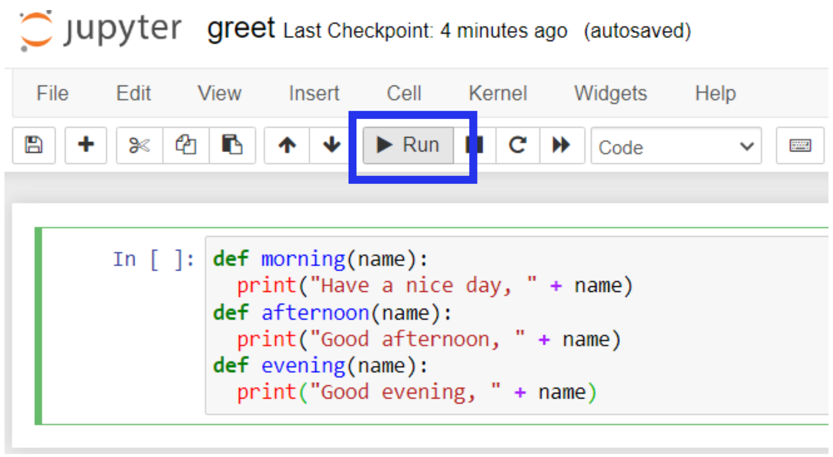

jupyter greet Last Checkpoint: a few seconds ago (unsaved changes)
File Edit View Insert Cell Kernel Widgets Help
Run
Code
In [ ]:

3. Create 3 functions - `morning`, `afternoon`, and `evening`


jupyter greet Last Checkpoint: 2 minutes ago (unsaved changes)
File Edit View Insert Cell Kernel Widgets Help
Run
Code
In [ ]:

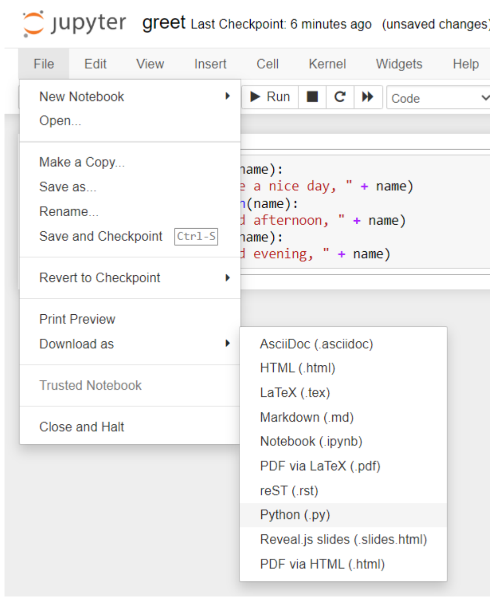
```python
def morning(name):
  print("Have a nice day, " + name)
def afternoon(name):
  print("Good afternoon, " + name)
def evening(name):
  print("Good evening, " + name)
```

4. Run the program

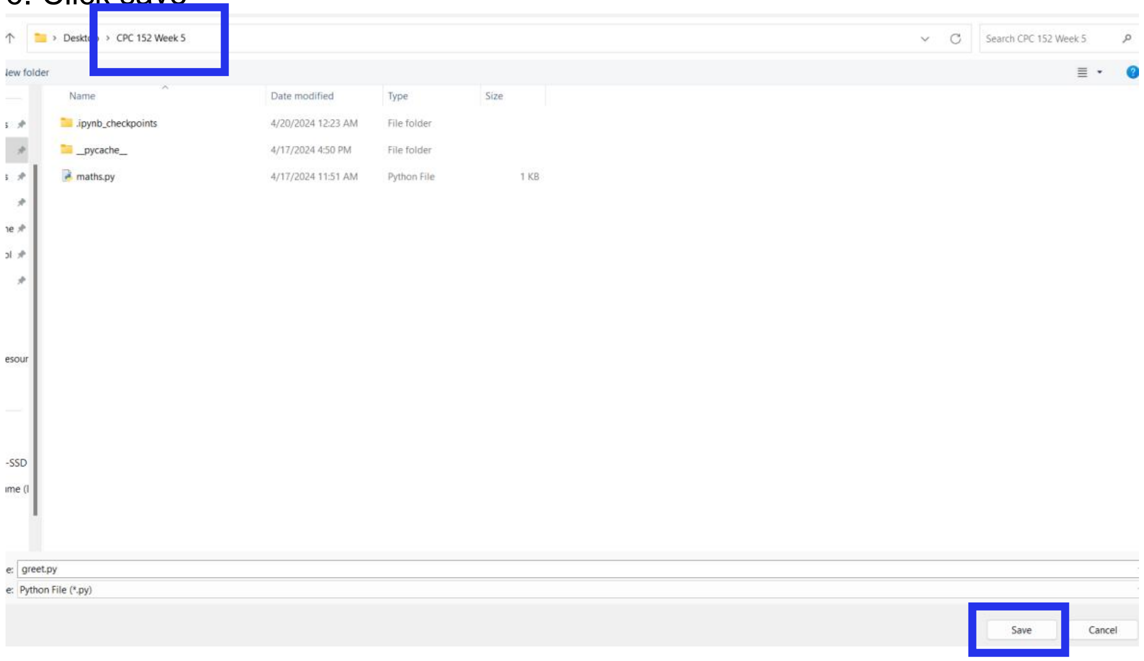

jupyter greet Last Checkpoint: 4 minutes ago (autosaved)
File Edit View Insert Cell Kernel Widgets Help
Run
Code

```
In [ ]: def morning(name):
          print("Have a nice day, " + name)
        def afternoon(name):
          print("Good afternoon, " + name)
        def evening(name):
          print("Good evening, " + name)
```

5. Click File → Download as → Python(.py)



6. Click save



Desktop › CPC 152 Week 5

Search CPC 152 Week 5

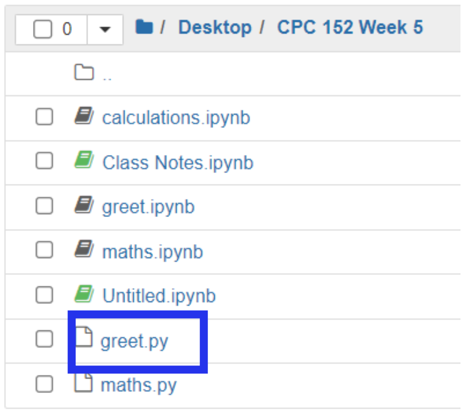
New folder

Name	Date modified	Type	Size
.ipynb_checkpoints	4/20/2024 12:23 AM	File folder	
__pycache__	4/17/2024 4:50 PM	File folder	
maths.py	4/17/2024 11:51 AM	Python File	1 KB

File name: greet.py

Save as type: Python File (*.py)

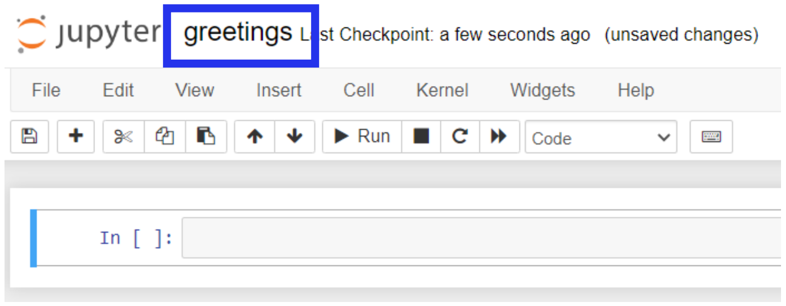
7. Close the file



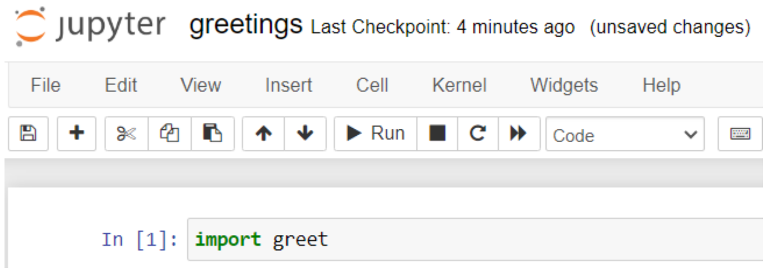
Desktop / CPC 152 Week 5

..
calculations.ipynb
Class Notes.ipynb
greet.ipynb
maths.ipynb
Untitled.ipynb
greet.py
maths.py

8. Create an another jupyter notebook in the same folder and rename into `greetings`

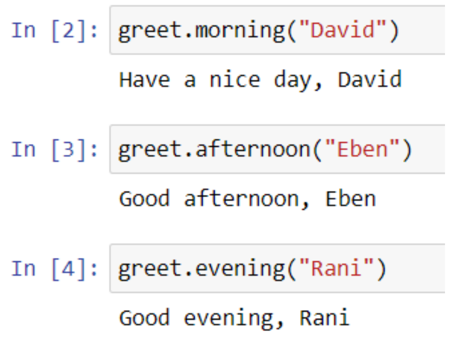

jupyter
greetings
Last Checkpoint: a few seconds ago (unsaved changes)
File Edit View Insert Cell Kernel Widgets Help
Run
Code
In [ ]:

9. Import the newly created module `greet`


jupyter
greetings Last Checkpoint: 4 minutes ago (unsaved changes)
File Edit View Insert Cell Kernel Widgets Help
Run
Code

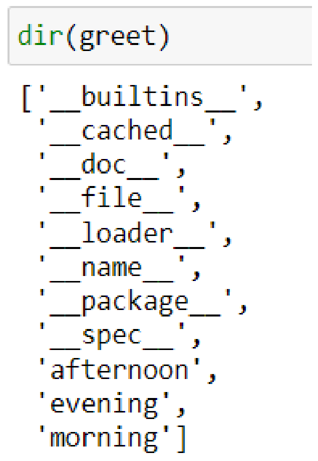
```
In [1]: import greet
```

10. Then access the 3 functions (`morning, afternoon,` and `evening)` created in `greet` module.



```
In [2]: greet.morning("David")

        Have a nice day, David

In [3]: greet.afternoon("Eben")

        Good afternoon, Eben

In [4]: greet.evening("Rani")

        Good evening, Rani
```

11. In Python, we can use the `dir()` function to list all the function names in a module.

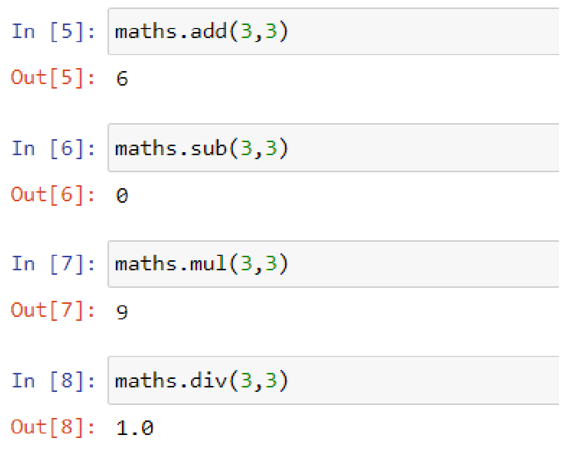


```
dir(greet)
```

```
['__builtins__',
 '__cached__',
 '__doc__',
 '__file__',
 '__loader__',
 '__name__',
 '__package__',
 '__spec__',
 'afternoon',
 'evening',
 'morning']
```

Program

1. Create the following



```
In [5]: maths.add(3,3)

Out[5]: 6

In [6]: maths.sub(3,3)

Out[6]: 0

In [7]: maths.mul(3,3)

Out[7]: 9

In [8]: maths.div(3,3)

Out[8]: 1.0
```