Session 6: Control statements & table joining

Module BUSN9690

Business Statistics with Python

Nested loops

For-nested-loop

```
for iterating_var in sequence:
  for iterating_var in sequence:
    statements(s)
  statements(s)
```

While-nested-loop

```
while expression:
  while expression:
    statement(s)
  statement(s)
```

Example: to find the prime numbers from 2 to 100 –

```
i = 2
while(i < 100):
  j = 2
  while(j <= (i/j)):
    if not(i%j): break
    j = j + 1
  if (j > i/j) : print(i, " is prime")
  i = i + 1
print("Good bye! ")
```

Continue statement

- Loop control statements change execution from its normal sequence. When execution leaves a scope, all automatic objects that were created in that scope are destroyed. Python supports the following control statements.
- Continue Statement: It returns the control to the beginning of the loop.

- # Prints all letters except ‘f' and 's'

```
for letter in 'Then I’ll huff and I’ll puff and I’ll blow your house in':
    if letter == 'f' or letter == 's':
        continue
    print("Current Letter :", letter)
```

Break statement

- It brings control out of the loop

- for letter in “Then I’ll huff and I'll puff and I’ll blow your house in”:
- # break the loop as soon it sees 'f' or 's'
- if letter == 'f' or letter == 's':
- break
- print("Current Letter :", letter)

Pass statement

- We use pass statement to write empty loops. Pass is also used for empty control statement and function, which can be used when a loop or a function is not implemented yet, but will be implemented in the future.

- # An empty loop
- for letter in 'Then I'll huff and I'll puff and I'll blow your house in':
- pass

- def function(args):
- pass

Joining tables

Example

- Assume you are working for a university. Your line manager gave you two tables: Table A and Table B. She asked you to combine the two tables together

Table A

```
In [2]: students
Out[2]:
  StudentLogin  Lastname          Programme
0       A00010   POLLARD  BusinessAnalytics
1       B00020     SIMMS        SupplyChain
2       C00030     CHAN         SupplyChain
3       D00110  WHITEING            Banking
4       E00120     MEYER  BusinessAnalytics
5       F00130    HAMNAM  BusinessAnalytics
```

Table B

```
In [3]: marks
Out[3]:
            Module StudentLogin  Marks
0    BusinessStats       A00010     70
1  MachineLearning       C00030     65
2       Simulation       E00120     58
3          BigData       F00130     81
4  MachineLearning       E00120     73
5  MachineLearning       G00120     65
```

  - Table A does not have StudentLogin G00120
  - Table B does not have StudentLogin B00020 or D00110

Inner join

Syntax:

import pandas as pd

pd.merge(A,B,on=c)

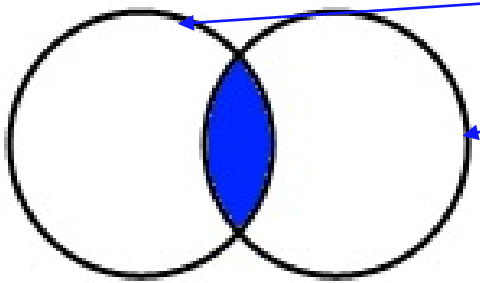
Return all rows that from A where there are matching values in B

Table A

Table B

```
In [2]: students
Out[2]:
  StudentLogin  Lastname          Programme
0        A00010   POLLARD  BusinessAnalytics
1        B00020     SIMMS        SupplyChain
2        C00030     CHAN         SupplyChain
3        D00110  WHITEING            Banking
4        E00120     MEYER  BusinessAnalytics
5        F00130    HAMNAM  BusinessAnalytics
```

```
In [3]: marks
Out[3]:
            Module StudentLogin  Marks
0    BusinessStats        A00010     70
1  MachineLearning        C00030     65
2       Simulation        E00120     58
3          BigData        F00130     81
4  MachineLearning        E00120     73
5  MachineLearning        G00120     65
```



```
In [4]: innerJoin=pd.merge(students,marks,on='StudentLogin')

In [5]: innerJoin
Out[5]:
  StudentLogin Lastname          Programme            Module  Marks
0        A00010  POLLARD  BusinessAnalytics      BusinessStats     70
1        C00030     CHAN        SupplyChain  MachineLearning     65
2        E00120    MEYER  BusinessAnalytics        Simulation     58
3        E00120    MEYER  BusinessAnalytics  MachineLearning     73
4        F00130   HAMNAM  BusinessAnalytics           BigData     81
```

To output all StudentLogin appearing in both Table A (left table) and Table B (right table); but StudentLogin B00020 (only in the left table), G00120, and D00110 (only in the right table) will be excluded

Left outer join:

Syntax: import pandas as pd; pd.merge(A,B,on=c,how='left')

Every row in A must appear in result. If no matching rows, padded with NULL values for variables of B

Table A

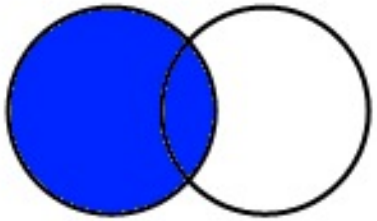
Table B

```
In [2]: students
Out[2]:
  StudentLogin  Lastname          Programme
0       A00010    POLLARD  BusinessAnalytics
1       B00020      SIMMS        SupplyChain
2       C00030       CHAN        SupplyChain
3       D00110  WHITEING            Banking
4       E00120      MEYER  BusinessAnalytics
5       F00130     HAMNAM  BusinessAnalytics
```

```
In [3]: marks
Out[3]:
            Module StudentLogin  Marks
0     BusinessStats       A00010     70
1  MachineLearning       C00030     65
2       Simulation       E00120     58
3          BigData       F00130     81
4  MachineLearning       E00120     73
5  MachineLearning       G00120     65
```

```
In [7]: leftOuterJoin=pd.merge(students,marks,on='StudentLogin',how='left')

In [8]: leftOuterJoin
Out[8]:
  StudentLogin  Lastname          Programme            Module  Marks
0       A00010    POLLARD  BusinessAnalytics     BusinessStats   70.0
1       B00020      SIMMS        SupplyChain               NaN    NaN
2       C00030       CHAN        SupplyChain  MachineLearning   65.0
3       D00110  WHITEING            Banking               NaN    NaN
4       E00120      MEYER  BusinessAnalytics        Simulation   58.0
5       E00120      MEYER  BusinessAnalytics  MachineLearning   73.0
6       F00130     HAMNAM  BusinessAnalytics          BigData   81.0
```



To output all StudentLogin appearing in Table A only; that is, G00120 and D00110, which only appear in Table B (i.e., the right table), will be excluded.

Right outer join:

Syntax: import pandas as pd; pd.merge(A,B,on=c,how='right')

Every row in B must appear in result.

If no matching rows, padded with NULL values for variables of A

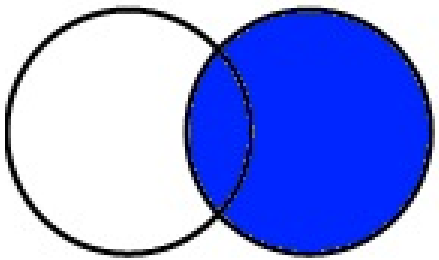


Table A

Table B

```
In [2]: students
Out[2]:
  StudentLogin  Lastname          Programme
0       A00010   POLLARD  BusinessAnalytics
1       B00020     SIMMS        SupplyChain
2       C00030      CHAN        SupplyChain
3       D00110  WHITEING            Banking
4       E00120     MEYER  BusinessAnalytics
5       F00130    HAMNAM  BusinessAnalytics
```

```
In [3]: marks
Out[3]:
            Module StudentLogin  Marks
0     BusinessStats       A00010     70
1  MachineLearning       C00030     65
2       Simulation       E00120     58
3          BigData       F00130     81
4  MachineLearning       E00120     73
5  MachineLearning       G00120     65
```

```
In [9]: rightOuterJoin=pd.merge(students,marks,on='StudentLogin',how='right')

In [10]: rightOuterJoin
Out[10]:
  StudentLogin Lastname          Programme            Module  Marks
0       A00010  POLLARD  BusinessAnalytics     BusinessStats     70
1       C00030     CHAN        SupplyChain  MachineLearning     65
2       E00120    MEYER  BusinessAnalytics        Simulation     58
3       F00130   HAMNAM  BusinessAnalytics          BigData     81
4       E00120    MEYER  BusinessAnalytics  MachineLearning     73
5       G00120      NaN                NaN  MachineLearning     65
```

To output all StudentLogin appearing in table B only; that is, B00020, which only appears in Table A, will be excluded.

Full join:

Syntax: import pandas as pd; pd.merge(A,B,on=c,how='outer')

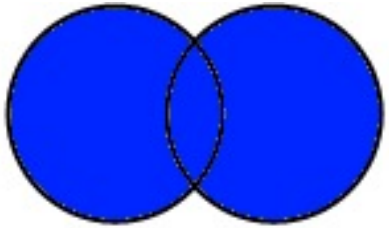
Joins data and returns all rows and columns

Table A

Table B

```
In [2]: students
Out[2]:
  StudentLogin  Lastname          Programme
0       A00010   POLLARD  BusinessAnalytics
1       B00020     SIMMS        SupplyChain
2       C00030      CHAN        SupplyChain
3       D00110  WHITEING            Banking
4       E00120     MEYER  BusinessAnalytics
5       F00130    HAMNAM  BusinessAnalytics
```

```
In [3]: marks
Out[3]:
              Module StudentLogin  Marks
0      BusinessStats       A00010     70
1  MachineLearning         C00030     65
2         Simulation       E00120     58
3            BigData       F00130     81
4  MachineLearning         E00120     73
5  MachineLearning         G00120     65
```



```
In [12]: fullJoin=pd.merge(students,marks,on='StudentLogin',how='outer')

In [13]: fullJoin
Out[13]:
  StudentLogin  Lastname          Programme          Module  Marks
0       A00010   POLLARD  BusinessAnalytics    BusinessStats   70.0
1       B00020     SIMMS        SupplyChain             NaN    NaN
2       C00030      CHAN        SupplyChain  MachineLearning   65.0
3       D00110  WHITEING            Banking             NaN    NaN
4       E00120     MEYER  BusinessAnalytics      Simulation   58.0
5       E00120     MEYER  BusinessAnalytics  MachineLearning   73.0
6       F00130    HAMNAM  BusinessAnalytics         BigData   81.0
7       G00120       NaN                NaN  MachineLearning   65.0
```

To output all StudentLogin appearing in Table A or Table B

An example: to grade student marks

```
>>>students=pd.DataFrame({'StudentLogin': ['A00010','B00020','C00030','D00110','E00120','F00130'],
'Lastname':['POLLARD','SIMMS','CHAN','WHITEING','MEYER','HAMNAM'],
'Programme': ['BusinessAnalytics','SupplyChain','SupplyChain','Banking',' BusinessAnalytics',' BusinessAnalytics']
}) #to create a data frame named students
>>>marks=pd.DataFrame({'Module': ['BusinessStats','MachineLearning','Simulation','BigData', 'MachineLearning',
'MachineLearning'],
'StudentLogin':['A00010', 'C00030', 'E00120','F00130', 'E00120', 'G00120'],
'Marks':[70,65,58,81,73,65]
}) #to create a data frame named marsk
#to conduct an inner join between the two tables: students and marks
>>>innerJoin=pd.merge(students,marks,on='StudentLogin')

#to conduct a left outer join between the two tables: students and marks
>>>leftOuterJoin=pd.merge(students,marks,on='StudentLogin',how='left')
```

An example: to grade student marks

```
#to conduct a right outer join between the two tables: students and marks
rightOuterJoin=pd.merge(students,marks,on='StudentLogin',how='right')

#to conduct a full join between the two tables: students and marks
fullJoin=pd.merge(students,marks,on='StudentLogin',how='outer')

numRows = len(innerJoin) #to find the number of rows in the data frame and assign it to numRows
temp = innerJoin.Marks #to assign the marks to temp, which is a vector
grade = [0 for i in range(numRows)] #to to initialize by assigning 0's to the array Grade
```

An example--continued

```
for i in range(numRows): #to iterate each student marks
    if temp[i] >= 70:
        grade[i] = "distinction"#if a student's grade is greater than 70, his/her grade is distinction
    elif temp[i] < 70 and temp[i] >= 60:
        grade[i] = "merit"#if a student's grade is between 60 an 70, his/her grade is merit
    else:
        grade[i] = "pass"#if a student's grade is less than 60, his/her grade is pass

innerJoin['grade']=grade #insert the final marks back to the resulting_table
innerJoin #print out the results
```

Recap

Matrix

- Let

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & a_{2m} \\ \cdots & \cdots & \ddots & \cdots \\ a_{n1} & a_{n2} & \cdots & a_{nm} \end{pmatrix}$$

then A is an n by m matrix, or an $n \times m$ matrix.

- If

$$B = \begin{pmatrix} a_{11} & a_{21} & \cdots & a_{n1} \\ a_{12} & a_{22} & \cdots & a_{n2} \\ \cdots & \cdots & \ddots & \cdots \\ a_{1m} & a_{2m} & \cdots & a_{nm} \end{pmatrix}$$

Then B is an $m \times n$ matrix, or the transpose of the above matrix A, or $B = A^T$, of course, $A = B^T$

Simply put: put the k-th row of A to the k-th column of B

Matrix

- Denote

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \cdots & \cdots & \ddots & \cdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix}$$

where $n = m$, then A is a square matrix of order n

- Let

$$I_n = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & 1 \end{pmatrix}$$

Then I_n is an $n \times n$ identity matrix. That is, the identity matrix I_n of size n is the $n \times n$ matrix in which all the elements on the main diagonal are equal to 1 and all other elements are equal to 0,

- Example
- Let $A = \begin{pmatrix} 65 & 60 & 70 & 80 \\ 72 & 65 & 55 & 65 \end{pmatrix}$ and $B = \begin{pmatrix} 65 & 72 \\ 60 & 65 \\ 70 & 55 \\ 80 & 65 \end{pmatrix}$, then $A = B^T$, or $B = A^T$
- $M_6 = \begin{pmatrix} 65 & 60 \\ 72 & 65 \end{pmatrix}$ is a square matrix of order 2
- $I_3 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$ is called a 3 x 3 identity matrix, or an identity matrix of order 3

Matrix addition/subtraction/multiplication/division

- If **A** is an $n \times m$ matrix and **B** is an $n \times m$ matrix,

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & a_{2m} \\ \cdots & \cdots & \ddots & \cdots \\ a_{n1} & a_{n2} & \cdots & a_{nm} \end{pmatrix}, \qquad B = \begin{pmatrix} b_{11} & b_{12} & \cdots & b_{1m} \\ b_{21} & b_{22} & \cdots & b_{2m} \\ \cdots & \cdots & \ddots & \cdots \\ b_{n1} & b_{n2} & \cdots & b_{nm} \end{pmatrix}$$

then ***matrix addition*** **C = A+B**: $C = \begin{pmatrix} c_{11} & c_{12} & \cdots & c_{1m} \\ c_{21} & c_{22} & \cdots & c_{2m} \\ \cdots & \cdots & \ddots & \cdots \\ c_{n1} & c_{n2} & \cdots & c_{nm} \end{pmatrix}$ is an $n \times$ m matrix, where $c_{ij} = a_{ij} + b_{ij}$

- Example

$$\begin{pmatrix} 1 & 2 & 0 \\ 0 & 7 & 6 \\ 4 & 0 & 1 \end{pmatrix} + \begin{pmatrix} 5 & 2 & 1.2 \\ 2 & 0 & 9 \\ 2 & 5 & 12 \end{pmatrix} = \begin{pmatrix} 1+5 & 2+2 & 0+1.2 \\ 0+2 & 7+0 & 6+9 \\ 4+2 & 0+5 & 1+12 \end{pmatrix} = \begin{pmatrix} 6 & 4 & 1.2 \\ 2 & 7 & 15 \\ 6 & 5 & 13 \end{pmatrix}$$

- Similarly, subtraction $(A - B)$, multiplication $(A \circ B)$, and division $(A \oslash B)$ can be done, all of them are element-wise operators. That is, each element of A is subtracted/multiplied/divided by the corresponding element of B

Scalar Multiplication

- If **A** is an $n \times m$ matrix,

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & a_{2m} \\ \cdots & \cdots & \ddots & \cdots \\ a_{n1} & a_{n2} & \cdots & a_{nm} \end{pmatrix},$$

The product $\alpha \boldsymbol{A}$ of a number α (also called a scalar) and a matrix A is computed by multiplying every entry of A by α: $\mathbf{C} = \alpha \boldsymbol{A}$: $C = \begin{pmatrix} c_{11} & c_{12} & \cdots & c_{1m} \\ c_{21} & c_{22} & \cdots & c_{2m} \\ \cdots & \cdots & \ddots & \cdots \\ c_{n1} & c_{n2} & \cdots & c_{nm} \end{pmatrix}$ is an $n \times$ m matrix,

where $c_{ij} = \alpha a_{ij}$

- An example

$$3 \times \begin{pmatrix} 1 & 2 & 0 \\ 0 & 7 & 6 \\ 4 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 3 & 6 & 0 \\ 0 & 21 & 18 \\ 12 & 0 & 3 \end{pmatrix}$$

Dot product between two matrices

- If **A** is an $n \times m$ matrix and **B** is an $m \times p$ matrix,

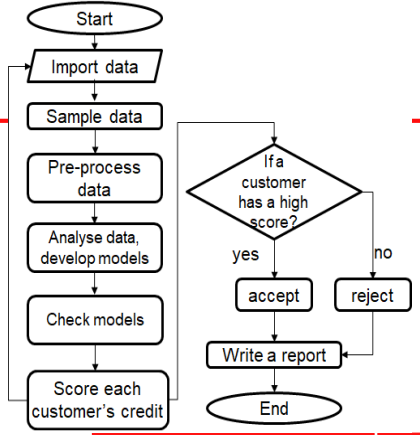
$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & a_{2m} \\ \cdots & \cdots & \ddots & \cdots \\ a_{n1} & a_{n2} & \cdots & a_{nm} \end{pmatrix}_{n \times m}, \qquad B = \begin{pmatrix} b_{11} & b_{12} & \cdots & b_{1p} \\ b_{21} & b_{22} & \cdots & b_{2p} \\ \cdots & \cdots & \ddots & \cdots \\ b_{m1} & b_{m2} & \cdots & b_{mp} \end{pmatrix}_{m \times p}$$

- then *matrix product* $C = A \cdot B$ (denoted $C = \begin{pmatrix} c_{11} & c_{12} & \cdots & c_{1p} \\ c_{21} & c_{22} & \cdots & c_{2p} \\ \cdots & \cdots & \ddots & \cdots \\ c_{n1} & c_{n2} & \cdots & c_{np} \end{pmatrix}$) is defined to be the $n \times p$ matrix

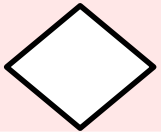
$$c_{ij} = a_{i1}b_{1j} + \cdots + a_{im}b_{mj} = \sum_{k=1}^{m} a_{ik}b_{kj} = \sum_{k=1}^{m} a_{ik}b_{kj}$$

- **Distinguish $A \cdot B$ from $A \circ B$. $A \cdot B$ is normally denoted by AB** (without multiplication sign or dot sign in-between)
- **Rules:**
  - **c_{ij} is the sum of the elements in the i-th row in A multiplying the elements in the j-th column in B**
  - **The dimensions of the resulting matrix are $n \times p$**

Basic flowchart symbols



Different shapes are used➔flowchart symbols: what do they stand for?

Symbol	Symbol Name	Purpose
	Start/Stop	Used at the beginning and end of the algorithm to show start and end of the program.
	Process	Indicates processes like mathematical operations.
	Input/ Output	Used for denoting program inputs and outputs.
	Decision	Stands for decision statements in a program, where answer is usually Yes or No.
	Arrow	Shows relationships between different shapes.

Working directory

```
>>>from os import *      #import the package into python

>>> getcwd()            #to show the current working directory

>>> chdir("c:\Wutemp\Python") #use c:/Wutemp/Python as the current directory

>>> getcwd()            #to re-show the current working directory
```

Alternatively, if you are using Spyder, click **Tools** → **Preferences** → **Current working directory**→enter your working directory to the cell in **The following directory**

Variable naming convention

- A valid variable name can only consist of letters, numbers and/or the underscore character;
- A valid variable name must start with a letter or the underscore character, but it cannot start with a number; a variable starting with the underscore has special meaning.
- Variable names are case-sensitive (age, Age and AGE are three different variables)

Variable Name	Validity	Reason
var_name2	valid	Starting with a letter
.var_name, var.name	invalid	Starting with a dot (.), or containing a dot (.)
var_name%	invalid	Has the character '%'. Only the underscore character is allowed.
2var_name	invalid	As it starts with a number
Var1, var1	valid	They are two different variables
_var_name	valid	Starts with _ which is valid

- if, else, elif, while, for, False, True, etc. are reserved and therefore cannot be used

Strings

- The quotes at the beginning and end of a string should be both double quotes or both single quote. They can not be mixed.
- Double quotes can not be inserted into a string starting and ending with double quotes.
- Single quote can not be inserted into a string starting and ending with single quote.
- Double quotes with \ as prefix can be inserted into a string starting and ending with double (or single) quotes.
- Single quote with \ as prefix can be inserted into a string starting and ending with double (or single) quotes.

Data types—String operations: cont'd

- Other examples

```
>>>str = 'It is a nice day today!'
>>>str.upper()
>>>str.lower()
>>>print(str[5:-2])    # Prints characters the 6th to 2nd last character
```

- You cannot delete or remove characters from a string

```
>>>str = 'It is a nice day today!'
>>>str[5] = 'a'
TypeError: 'str' object does not support item assignment
```

- Concatenation of Two or More Strings

```
>>>str1 = 'Hello'; str2 ='World!'
>>>print('str1 + str2 = ', str1 + str2)
>>>print(str * 2)            # Prints string two times
>>>print(str + ", do you like it?")   # Prints concatenated string
```

Data types—String operations: cont'd

- String Membership Test

  >>>str = 'It is a nice day today!'

  >>>'w' in str #to check whether the letter w is in the string str. the answer is True or False. Note: w differs from W

  >>>'w' not in str

- Built-in functions to work with Python

  len(): it returns the length (number of characters) of the string

  >>>len(str) #it will return value 23

Casting

- int(): constructs an integer number from an integer literal, a float literal (by removing all decimals), or a string literal (providing the string represents a whole number)
  - z=int("123") # z will be 123
  - x=int(2); # x will be 2
  - y=int(3.2); # y will be 3
- float(): constructs a float number from an integer literal, a float literal or a string literal (providing the string represents a float or an integer)
  - x=float(2); # x will be 2.0
  - y=float(3.2); # y will be 3.2
  - z=float("123") # z will be 123.0

- str(): constructs a string from a wide variety of data types, including strings, integer literals and float literals
  - x=str(2); # x will be "2"
  - y=str(3.2); # y will be "3.2"
  - z=str("123") # z will be "123"

Access elements in a list

- List items are indexed, the first item has index [0], the second item has index [1] etc.

  >>>Modules=["Business Statistics", "Simulation", "Machine Learning", "Big data", "Dissertation"]

- Then
  - Modules[1] #the 2nd item in the list
  - Modules[-2] #the second lst item in the list
  - Modules[1:3] # the 2nd to the 3th elements in the list
  - Modules[:3] # the first 3 elements in the list
  - Modules[2:] # the elements from the 3rd to the end in the list
-

Update and insert items

```
>>>Modules=["Business Statistics", "Simulation", "Machine Learning", "Big data", "Dissertation"]
```

- Change and insert items
  - Modules[0]="Linear programming" #change the first item in Modules to "linear programming"
  - Modules[1:3]=["Nonlinear programming","Genetic Algorithm"] #change the 2nd and 3rd items in Modules to "linear programming". The modules now become ['Business Statistics', 'Nonlinear programming', 'Genetic Algorithm', 'Big data', 'Dissertation']
- Insert a new item into the 3rd position
  - Modules.insert(2, "Python") #The list becomes ['Business Statistics', 'Nonlinear programming', 'Python', 'Genetic Algorithm', 'Big data', 'Dissertation']
- Append items onto the end of a list
  - Modules.append("placement") #append "placement" onto the list
  - moreModules=["Simulation","Machine Learning"] #these modules have been deleted, we need to add them back
  - Modules.extend(moreModules) #what is the output?

Remove, clear, and sort

```
>>>Modules=["Business Statistics", "Simulation", "Machine Learning", "Big data", "Dissertation"]
```

- Remove an item, delete the entire list
  - Modules.remove("Simulation") #remove "simulation" from the list
  - Modules.pop(1) #remove the 2nd item from the list
  - delete Modules #delete the entire list

```
>>>Modules=["Business Statistics", "Simulation", "Machine Learning", "Big data", "Dissertation"]
```

- Sort the elements in a list
  - Modules.sort() #The list "Modules" becomes ['Big data', 'Business Statistics', 'Dissertation', 'Machine Learning', 'Simulation']
  - Modules.sort(reverse=True) #The list "Modules" becomes ['Big data', 'Business Statistics', 'Dissertation', 'Machine Learning', 'Simulation']

Data types: list, tuple, set and dictionary

Four built-in data types in Python used to store collections of data: list, tuple, set and dictionary

Data Type	Example	Verify
List **Definition: []**	list1 = ["apple", "peach", "mellon"] list2 = [1, 15, 23, 91, 93] list3 = [True, False, False] list4 = ["peach", 21, True, 9, "cat"]	>>>list4 = ["peach", 21, True, 9, "cat"] >>>type(list4)
Tuple **Definition: ()**	tuple1 = ("apple", "peach", "mellon") tuple2 = (1, 15, 23, 91, 93) tuple3 = (True, False, False) tuple4 = ("peach", 21, True, 9, "cat")	>>>tuple4 = ("peach", 21, True, 9, "cat") >>>type(tuple4)
Set **Definition: {} without keys**	set1 = {"apple", "peach", "mellon"} set2 = {1, 15, 23, 91, 93} set3 = {True, False, False} set4 = {"peach", 21, True, 9, "cat"}	>>>set4 = {"peach", 21, True, 9, "cat"} >>>type(set4)

Dictionary
Definition: {} with keys
	thisdict = {
	"course": "MSC Business Analytics",
	"module": "Business Statistics with Python",
	"year": 2021
	}
		>>> thisdict = { "course": "MSC Business Analytics",
		"module": "Business Statistics with Python",
		"year": 2021}
		>>>type(thisdict)

Differences between list, tuple, set, and dictionary

	List	Tuple	Set	Dictionary
Represented by	[ ]	( )	{ }	{ }
Allows duplicate elements?	Yes	Yes	No	Not on keys
Function used to create	list()	tuple()	set()	dict()
Indexing /Slicing	Yes	Yes	No	Yes
Mutable or not?	Mutable	Immutable	Mutable, but elements are not duplicated.	Mutable, but Keys are not duplicated.
Ordered	ordered	ordered	unordered	unordered

Some commonly used built-in functions

Function in R	Description
abs(x)	absolute value: $\|x\|$
math.sqrt(x)	square root: $\sqrt{x}$
math.ceil(x)	math.ceil(3.475) is 4, math.ceil$(-1.9) = -1$
math.floor(x)	math.floor(3.475) = 3, math.floor$(-1.9) = -2$
math.trunc(x)	math.trunc(5.99) is 5, math.trunc$(-1.9) = -1$
round(x, n)	Round a number to only two decimals: round(3.475, 2) is 3.48
math.cos(x), math.sin(x), math.tan(x)	also math.acos(x), math.cosh(x), math.acosh(x), etc.
math.log(x)	natural logarithm, i.e, ln(x)
math.log10(x)	common logarithm
math.exp(x)	e^x

numpy.arange()

- numpy.arange(): Create a Numpy Array of evenly spaced numbers in Python
- Syntax:
  - numpy.arange([start,]stop, [step,]dtype=**None**)
  - # If step is not specified, step=1
- Example 1:
  - import numpy as np
  - arr1= np.arange(5, 30, 2) # Start = 5, Stop = 30, Step Size = 2
  - print(arr1) # it will output: [5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29]
- Example 2:
  - import numpy as np
  - arr1 = np.array([1.1, 2.1, 3.1])
  - newarr1 = arr1.astype('i') # convert the elements in arr1 into integer type
  - print(newarr1) #it will output: [1 2 3]

Array shape and reshape

- Shape: the number of elements in each dimension.
  - import numpy as np
  - arr1 = np.array([[1, 2, 3, 4], [5, 6, 7, 8]])
  - print(arr1.shape) #output: (2, 4)
- From 1-d to 2-d
  - import numpy as np
  - arr1 = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])
  - newarr2 = arr1.reshape(4, 3)
  - print(newarr2)

```
#output: [[ 1  2  3]
          [ 4  5  6]
          [ 7  8  9]
          [10 11 12]]
```

- Flattening the arrays
  - import numpy as np
  - arr2 = np.array([[1, 2, 3], [4, 5, 6]])
  - newarr1 = arr2.reshape(-1)
  - print(newarr1)
- You can also reshape from 1-d to 3-d, or flat from 3-d to 1-d

Joining NumPy Arrays

- Joining means putting contents of two or more arrays in a single array
  - Join two 1-d arrays
    - import numpy as np
    - arr1 = np.array([1, 2, 3])
    - arr2 = np.array([4, 5, 6])
    - arr = np.concatenate((arr1, arr2))
    - print(arr) #[1 2 3 4 5 6]
  - Join two 2-d arrays
    - import numpy as np
    - arr1 = np.array([[1, 2], [3, 4]])
    - arr2 = np.array([[5, 6], [7, 8]])
    - arr = np.concatenate((arr1, arr2), axis=1)
    - print(arr)

- Joining Arrays Using Stack Functions
  - import numpy as np
  - arr1 = np.array([1, 2, 3])
  - arr2 = np.array([4, 5, 6])
  - arr = np.stack((arr1, arr2), axis=1)
  - print(arr)

- There are other joining methods

Array splitting

- import numpy as np
- arr1 = np.array([1, 2, 3, 4, 5, 6])
- arr2 = np.array([[1, 2], [3, 4], [5, 6], [7, 8], [9, 10], [11, 12]])
- newarr1 = np.array_split(arr1, 3)
- newarr2 = np.array_split(arr2, 3)
- newarr3 = np.array_split(arr2, 3,axis=1)
- print(newarr1) #[array([1, 2]), array([3, 4]), array([5, 6])]
- print(newarr2) #[array([[1, 2], [3, 4]]), array([[5, 6],[7, 8]]), array([[9, 10], [11, 12]])]
- print(newarr3) #array([[1], [3], [5], [7], [9], [11]]), array([[2], [4], [6],[8], [10], [12]]), array([], shape=(6, 0), dtype=int32)]

Searching Arrays

- To find a certain value in an array, and return the indexes that get a match, using keyword where().
  - import numpy as np
  - arr = np.array([1, 2, 3, 4, 5, 4, 4])
  - x = np.where(arr == 4)
  - print(x)

- The searchsorted() method is assumed to be used on sorted arrays.
  - import numpy as np
  - arr = np.array([6, 7, 8, 9])
  - x = np.searchsorted(arr, 7)
  - print(x)) #output: 1

Sorting Arrays

- Sort an array using sort()
  - import numpy as np
  - arr1 = np.array([6, 8, 7, 9])
  - arr2 = np.array(['banana', 'cherry', 'apple'])
  - arr3 = np.array([[3, 2, 4], [5, 0, 1]])
  - x1 = np.sort(arr1)
  - x2 = np.sort(arr2)
  - x3 = np.sort(arr3)
  - print(x1)) #output: [6,7,8,9]
  - print(x2)) #output: ['apple','banana', 'cherry']
  - print(x3)) #output: [[2,3,4],[0, 1,5]]

Other useful methods in numpy: Arithmetic operation

- import numpy as np #np is just an alias, an acronym of numpy

- arr1 = np.array([10, 20, 30, 40, 50, 60])
- arr2 = np.array([3, 5, 6, 8, 2, 1])

- arrAdd = np.add(arr1, arr2)#elements in arr1 add those in arra2. output: [13, 25, 36, 48, 52, 61])
- arrSub = np.subtract(arr1, arr2) #output: [7, 15, 24, 32, 48, 59]
- arrTimes = np.multiply(arr1, arr2) # output: [30, 100, 180, 320, 100, 60]
- arrDiv = np.divide(arr1, arr2) # output: ([3.33333333, 4., 5., 5., 25.,60.])
- arrPow = np.power(arr1, arr2) #output: ([1000,3200000,729000000,-520093696,2500,60],dtype=int32)
- arrMod = np.mod(arr1, arr2) # mode of output: [1, 0, 0, 0, 0, 0]
- arrLog = np.log2(arr1)#log at base 2 of all elements of arr1. output:

Other useful methods: basic statistics

- Basic statistics
  - import numpy as np

  - arr1 = np.array([10, 20, 30, 40, 50, 60])
  - arr2 = np.array([3, 5, 6, 8, 2, 1])

  - arrSum = np.sum([arr1,arr2])#sum of all elements of arr1 and arr2. output: 235
  - arrMin = np.amin(arr1) #the minimum value of the elements in arr1. output: 10
  - arrMax = np.amax(arr1) #the maximum value of the elements in arr1. output: 60
  - arrMean = np.mean(arr1) #the mean value of the elements in arr1. output: 35

Other useful methods: frequency & indices

- find the unique elements in a numpy array. Syntax:

numpy.unique(arr, return_index=False, return_inverse=False, return_counts=False, axis=None)

- arr : Numpy array in which we want to find the unique values.
- return_index : optional bool flag. If True returns an array of indices of first occurrence of each unique value.
- return_counts : optional bool flag. If True returns an array of occurrence count of each unique value.
- axis : If not provided then will act on flattened array. If 0 or 1 then acts on row or column wise.

```
import numpy as np
arr1 = np.array([11, 11, 12, 13, 14, 15, 16, 17, 12, 13, 11, 14, 18])
arrUnique = np.unique(arr1) #arrUnique==[11, 12, 13, 14, 15, 16, 17, 18]
uniqueValues, indicesList = np.unique(arr1, return_index=True) #to find a tuple of unique values & their first index location from a numpy array. Unique Values : [11,12,13,14,15,16,17,18]. Indices of Unique Values : [ 0 2 3 4 5 6 7 12]
uniqueValues, occurCount= np.unique(arr1, return_counts =True) #to find a tuple of unique values & their first index location from a numpy array. Unique Values : [11 12 13 14 15 16 17 18]. Counts of Unique Values : [3 2 2 2 1 1 1 1]
```

Shuffling, random number generating

- Shuffling: Shuffling aims to change the arrangement of elements in-place. i.e. in the array itself.
  - from numpy import random
  - import numpy as np
  - arr = np.array([1, 2, 3, 4, 5])
  - random.shuffle(arr)
  - print(arr)

- To generate a random normal distribution of size
  - from numpy import random
  - x = random.normal(size=(2, 30)) #generate two series of random numbers, each with 30 values
  - print(x)

Pandas DataFrame

- Pandas DataFrame is two-dimensional, size-mutable, potentially heterogeneous tabular data structure with labelled axes (rows and columns)

Name	Business statistics	Simulation	Machine Learning	Big Data	Dissertation
John	65	60	70	80	65
Anna	72	65	71	65	62
Emma	56	64	67	63	65
Nigel	78	70	76	80	75
Ben	72	68	70	76	72

Creating a DataFrame directly

```
# import pandas as pd
    import pandas as pd

# dictionary of students
    marks=pd.DataFrame({'Module': ['BusinessStats','MachineLearning','Simulation','BigData',
    'MachineLearning', 'MachineLearning'], 'StudentLogin':['A00010', 'C00030', 'E00120','F00130',
    'E00120', 'G00120'],'Marks':[70,65,58,81,73,65]})

    print(marks)
```

```
In [13]: marks
Out[13]:
             Module StudentLogin  Marks
0    BusinessStats        A00010     70
1  MachineLearning        C00030     65
2       Simulation        E00120     58
3          BigData        F00130     81
4  MachineLearning        E00120     73
5  MachineLearning        G00120     65
```

Accessing, selecting, deleting, adding, and renaming

- Selecting
  - marks[['Module', 'Marks']]# select two columns
  - marks.loc[1,] # select the second row
  - marks.iat[1,2]#
  - marks.loc[1,'Marks']
- Deleting
  - marks.drop(['Module','Marks'], axis=1) # delete two columns named Module and Marks
  - marks.drop([0,1]) # delete the 1st two rows

- Adding
  - marks['Grade'] = ['Dis', 'Merit', 'Pass', 'Dis', 'Dis','Merit'] #add a new column called 'Grades' to marks.
  - # add a row onto the dataframe
  - new_row={'Module':'BigData','StudentLogin':'H00120','Marks':56,'Grade':'Pass'}
  - marks = marks.append(new_row, ignore_index=True)
- Renaming
  - marks.columns=['Module1', 'StudentLogin1', 'Marks1', 'Grade1']
  - marks1=marks.rename(columns={'Marks1':'Marks2'}) #rename mark1 to mark2: only rename one of the column names

Dot product between matrices using pandas DataFrames

- Multiplying two matrices of same dimensions

```
import pandas as pd
#to define two matrices
matrix1 = [(1, 1, 2),
        (0, 2, 1),
        (2, 0, 1)];

matrix2 = [(2, 0, 2),
        (1, 1, 1),
        (2, 2, 2)];

# Data loaded into pandas DataFrames
dataFrame1 = pd.DataFrame(data=matrix1);
dataFrame2 = pd.DataFrame(data=matrix2);
```

```
#to find the dimensions of dataFrame1, 2
Dim1 = dataFrame1.shape;
Dim2 = dataFrame2.shape;

# to find the dot product between Matrix1 and Matrix2
dotResult = dataFrame1.dot(dataFrame2)

#to find the addition and substraction netween dataFrame 1 and dataFrame2
addResult = dataFrame1+dataFrame2
subResult = dataFrame1-dataFrame2


#to find the transpose of matrix2
dataFrame2.transpose()
```

Read a cvs file into Python

- If you are going to read an existing file into Python, you first need to
  - Know the folder the file is located in;
  - Know the name of the file;
  - Know whether you will need to read the variable names, normally from the first row of the existing file; and
  - Give a name of the data frame to which the dataset will be assigned.

- **#Basic syntax: use** read_csv() to read a comma-separated values (csv) file into DataFrame.

- Dataframe0=pandas.read_csv(file, header =) #you have read the csv file into Dataframe0
  - file: the path to the file to read
  - header: a logical value. If header=0, the first row is used as the names of the variables, header =1, the first row is not treated as the names of the variables

Write a dataframe to a cvs file

- If you are going to write a dataframe to a csv file, you first need to
  - Know the folder the file is located in;
  - Know the name of the dataframe;
  - Give a name of the csv file to which the dataframe will be written to.

- **#Basic syntax: use** to_csv() to write a dataframe to a comma-separated values (csv) file.
- df.to_csv(file) #you have written a dataframe called df to a csv file
  - file: the path to the file to read
- Example:
  - import pandas as pd #import library pandas
  - marks=pd.DataFrame({'Module': ['BusinessStats','MachineLearning','Simulation','BigData', 'MachineLearning', 'MachineLearning'], 'StudentLogin':['A00010', 'C00030', 'E00120','F00130', 'E00120', 'G00120'],'Marks':[70,65,58,81,73,65]})
  - marks.to_csv("C:\Wutemp\Python\example.csv")

Data cleansing

- Useful functions. Suppose a data frame called DF is given
  - DF.info() #to find the basic information of DF
  - DF.describe() #to find the descriptive statistics of DF
  - DF.shape() #to find the numbers of rows and columns of DF
  - DF.head() #show the first 5 rows
  - DF.tail() #show the last 5 rows

  - DF.dropna() #to drop missing data
  - DF.duplicated()#to check duplicates.
  - DF.drop_duplicates(inplace = True)#this will remove the duplicate

Fix the inconsistent value of No 6

- import pandas as pd
- Adult_df = pd.read_csv('adult.csv') #it contains 21 records
- Adult_df.loc[5, 'age'] = 35 #replace age in No.6 with 35.
  Note: Python starts from 0, No 6 therefore becomes row 5
- Note: you can also use Adult_df.iat[5,1] to access the value at the 6th row and 2nd column, where iat is a function name
  - DataFrame.iat: Access a single value for a row/column pair by integer position: Adult_df.iat[5,1]
  - DataFrame.loc: Access a group of rows and columns by label(s): Adult_df.loc[5, 'age']

No	age	date	education	occupation	Income
1	39	01/01/1995	Bachelors	Adm-clerical	<=50K
2	50	02/01/1995	Bachelors	Exec-managerial	<=50K
3	38	03/01/1995		Handlers-cleaners	<=50K
4	53	04/01/1995	11th	Handlers-cleaners	<=50K
5	28	05/01/1995	Bachelors	Prof-specialty	<=50K
6	177	06/01/1995	Masters	Exec-managerial	<=50K
7	49	07/01/1995	9th	Other-service	<=50K
7	49	07/01/1995	9th	Other-service	<=50K
8	52	08/01/1995	HS-grad	Exec-managerial	>50K
9	31	09/01/1995	Masters	Exec-managerial	>50K
10	42	10/01/1995	Bachelors	Exec-managerial	>50K
11	37	11/01/1995	Some-college	Exec-managerial	>50K
12	30	12/01/1995	Bachelors	Prof-specialty	>50K
13	23	13/01/1995	Bachelors	Adm-clerical	<=50K
14	32	14/01/1995	Assoc-acdm	Sales	<=50K
15	40	15/01/1995	Assoc-voc	Craft-repair	>50K
16	34	16/01/1995	7th-8th	Transport-moving	<=50K
17	25	17/01/1995	HS-grad	Farming-fishing	<=50K
18	32	18/01/1995	HS-grad	Machine-op-inspct	<=50K
19	38	19/01/1995	11th	Sales	<=50K
20	43	20/01/1995	Masters	Exec-managerial	>50K

User-defined function

- Basic syntax #function without return value

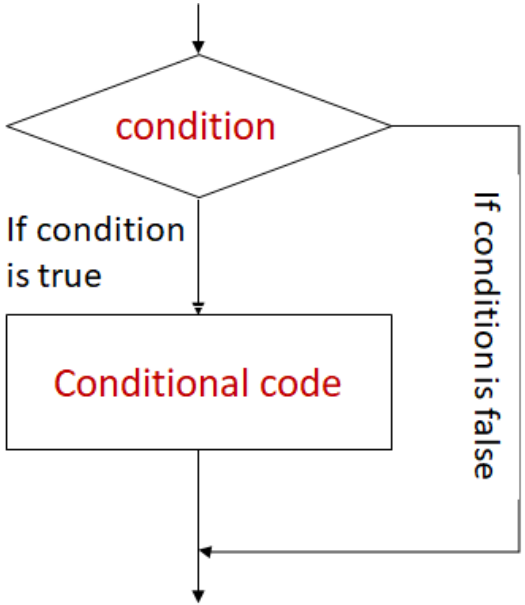
```
def functionName(par_1,...,par_n): #par_1,...,par_n are arguments, which are optional
    statements #function body: remember the indentation
```

- Basic syntax #function without return value

```
def functionName(par_1,...,par_n): #par_1,...,par_n are arguments, which are optional
    statements #function body
    return value1
```

if-else

- The if-else structure allows you to execute statements depending on whether a given condition is true or false



#Syntax:

```
if  <condition>:
   statement # do something
else:
    statement## do something
```

#Syntax:

```
if <condition1>:
      statement# do something
elif <condition2>:
      statement# do something different
else:
      statement#do something different
```

- Basic rule:
  - Do not forget the **indentation** and the **colon**

for-loop

- for loops take an iterator variable and assign it successive values from a sequence or vector.

```
#Basic syntax:
for val in sequence or vector:
    statement
```

while-loop

- Repeats a statement or group of statements while a given condition is true. It tests the condition before executing the loop body.

```
#Basic syntax:
while test_expression:
  statement
```