

Session 2: Basic Operators & Data Type I

Module BUSN9690
Business Statistics with Python

Professor Shaomin Wu

A real dataset

Missing data

Outlier

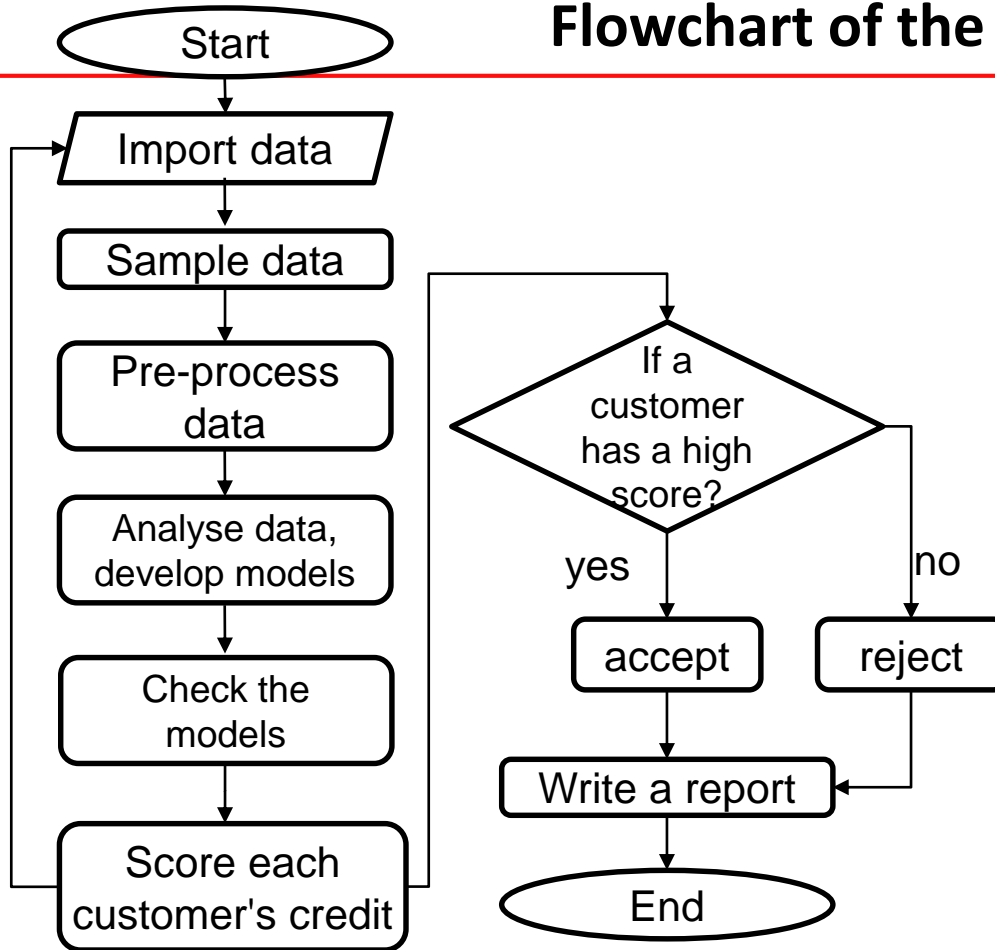
Inconsistent data

Obs	age	workclass	education	marital_status	occupation	relationship	race	gender	hours_per_week	Income
1	39	State-gov	Bachelors	Never-married	Adm-clerical	Not-in-family	White	Male	40	<=50K
2	50	Self-emp-not-inc	Bachelors	Married-civ-spouse	Exec-managerial	Husband	White	Male	13	<=50K
3	38		HS-grad	Divorced	Handlers-cleaners	Not-in-family	White	Male	40	<=50K
4	53	Private	11th	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	40	<=50K
5	28	Private	Bachelors	Married-civ-spouse	Prof-specialty	Wife	Black	Female	40	<=50K
6	37	Private	Masters	Married-civ-spouse	Exec-managerial	Wife	White	Female	40	<=50K
7	49	Private	9th	Married-spouse-absent	Other-service	Not-in-family	Black	Female	16	<=50K
8	154	Self-emp-not-inc	HS-grad	Married-civ-spouse	Exec-managerial	Husband	White	Male	45	>50K
9	31	Private	Masters	Never-married	Prof-specialty	Not-in-family	White	Female	50	>50K
10	42	Private	Bachelors	Married-civ-spouse	Exec-managerial	Husband	White	Male	40	>50K
11	37	Private	Some-college	Married-civ-spouse	Exec-managerial	Husband	Black	Male	80	>50K
12	30	State-gov	Bachelors	Married-civ-spouse	Prof-specialty	Husband	Asian-Pac-Islander	Male	40	>50K
13	23	Private	Bachelors	Never-married	Adm-clerical	Own-child	White	Female	30	<=50K
14	32	Private	Assoc-acdm	I am 21!	Sales	Not-in-family	Black	Male	50	<=50K
15	40	Private	Assoc-voc	Married-civ-spouse	Craft-repair	Husband	Asian-Pac-Islander	Male	40	>50K
16	34	Private	7th-8th	Married-civ-spouse	Transport-moving	Husband	Amer-Indian-Eskimo	Male	45	<=50K
17	25	Self-emp-not-inc	HS-grad	Never-married	Farming-fishing	Own-child	White	Male	35	<=50K
18	32	Private	HS-grad	Never-married	Machine-op-inspct	Unmarried	White	Male	40	<=50K
19	38	Private	11th	Married-civ-spouse	Sales	Husband	White	Male	50	<=50K
20	43	Self-emp-not-inc	Masters	Divorced	Exec-managerial	Unmarried	White	Female	45	>50K
21	40	Private	Doctorate	Married-civ-spouse	Prof-specialty	Husband	White	Male	60	>50K
22	54	Private	HS-grad	Separated	Other-service	Unmarried	Black	Female	20	<=50K

An example

- Suppose you are a business analyst in a bank. Your manager, Emma, is planning a marketing campaign that aims to attract valuable customers to apply for their credit cards. You are given a dataset of 1 million customers. Emma asked you to analyse these data and give her a list of customers she should send the application forms to.
- To undertake this project, you will need
 - 1) To read the 1 million observations into your computer ---(**BUSN9690**);
 - 2) To sample a subset of the data---(**BUSN9690**)
 - 3) To pre-process / pre-analyse the data ---(**BUSN9040**);
 - 4) To analyse the data and build models---(**BUSN9040**);
 - 5) To validate the models ---(**BUSN9690, BUSN9040**);
 - 6) To score each customer's credit ---(**BUSN9040**); and
 - 7) To write a report detailing the modelling process and providing your manager with the list of customers whose scores are high ---(**BUSN9690, BUSN9040**)

Flowchart of the task

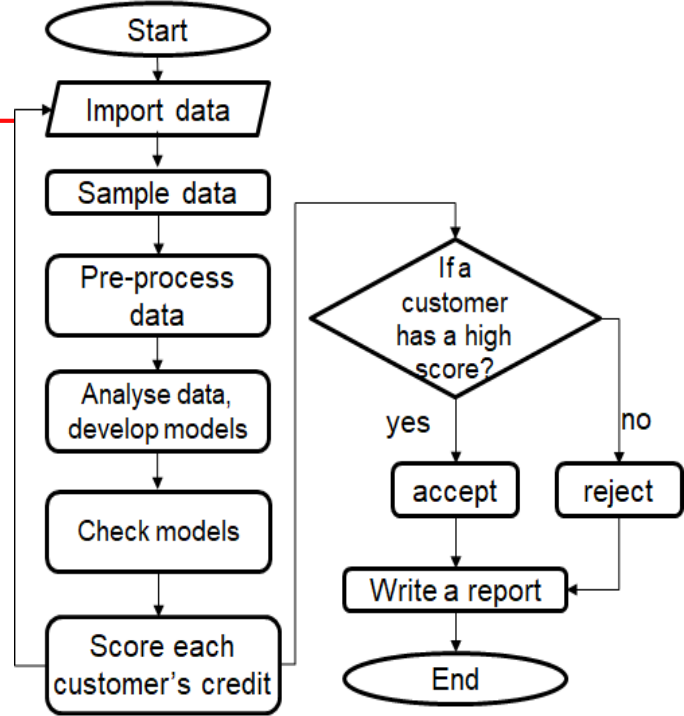


❖ Different shapes are used → flowchart symbols: [how many different shapes?](#)



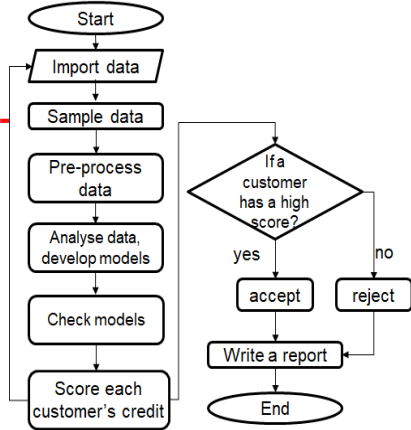
Flowchart of the task




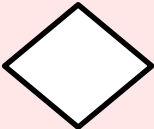

- ❖ Your data may include **age**, **address**, income, etc. → data type
 - ❑ age: integer
 - ❑ address: string
 - ❑ Income: float
- ❖ Each customer's record forms a vector → data type
- ❖ if-statement → A condition clause
- ❖ Import/export data
- ❖ Check each customer's score → loop-clause



Basic flowchart symbols

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

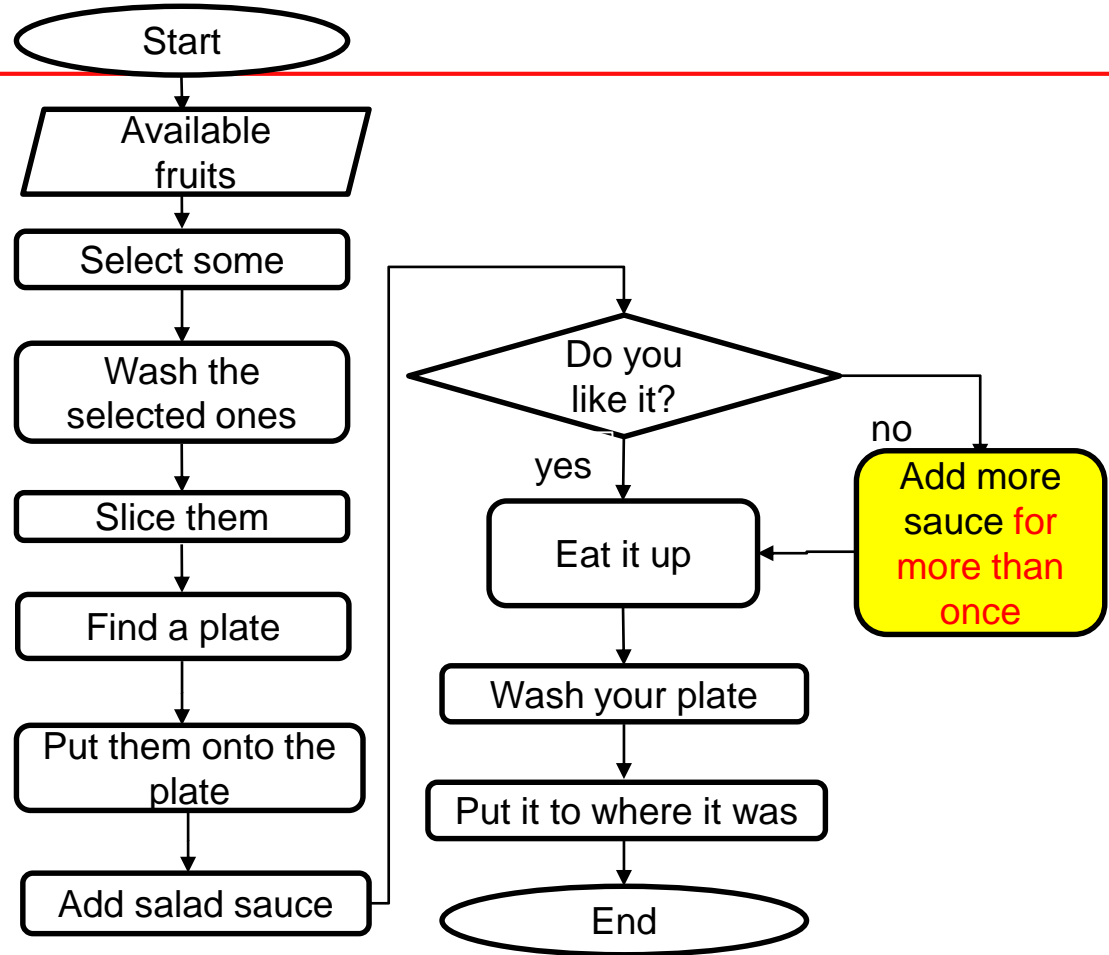


Symbol	Symbol Name	Purpose
	Start/Stop	Used at the beginning and end of the algorithm to show start and end of the program.
	Process	Indicates processes like mathematical operations.
	Input/ Output	Used for denoting program inputs and outputs.
	Decision	Stands for decision statements in a program, where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

Question



- Sophie would like to make some salad. Can you create a flowchart to show the process, using the symbols shown on Slide 6?



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](#)

First program in Python

- April attends module BUSN9690. She is 22. She likes month April and being 22. But she does not want to obtain a mark 22 on BUSN9690

```
>>> print("April attends module BUSN9690. She is 22. She likes month April and  
being 22. But she does not want to obtain a mark 22 on BUSN9690")
```

- to assign "April" to a variable called name.

```
>>> name="April"
```

- This should not be read as *name equals April*

```
>>> print(name + " attends module BUSN9690. She is 22. She likes month "+name +"  
and being 22. But she does not want to obtain a mark 22 on BUSN9690 ")
```

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

Essentials

- The operator `=` can be used, to assign something to a variable

```
>>>name="April"
```

```
>>>X = 5
```

```
>>>y = 4
```

```
>>>X+y #Python is case-sensitive, e.g., x and X are different
```



Q: What is the value of `x+y`?

- Other legal commands

```
>>>name="April"; X = 5; y = 4 #you may write these assignments in one line
```

```
>>>name,X,y="April", 5, 4 #You may assign multiple objects to multiple variables
```

```
>>>a=b=c=1 #You may also assign a single value to several variables simultaneously
```

- Delete variables

```
>>>del X,y,name #You have delete X, y, and name
```

Question



Question: can you replace the value 22 and module BUSN9690 with age and module, respectively?

```
>>> name="April"
```

```
>>> age="22"
```

```
>>> module="BUSN9690"
```

```
>>> print(name +" attends module "+ module+". She is "+ age+" . She likes month"+ name+"  
and being "+age+" . But she does not want to obtain a mark "+age+" on "+module)
```

(or

```
print(name," attends module ", module,". She is ", age, ". She likes month", name," and being ", age, ".  
But she does not want to obtain a mark ", age, " on ", module)
```

)

Python Basic Syntax

- End of Statements

- To end a statement in Python, you simply press Enter, nothing else;
- To ensure readability, you are suggested to limit the length of any single line of code to a number of characters, 79 characters, say;
- If a line of code is incomplete, you may use the backslash `\` to indicate that a statement is continued on the next line, e.g.

```
>>>print("April attends module BUSN9690. She is 22. \  
    She likes month April and being 22. \  
    But she does not want to obtain a mark 22 on BUSN9690")
```

Python Basic Syntax

- Line continuation is automatic when the split comes while a statement is inside parenthesis (, square brackets [or curly brackets {.
- Comments
 - Comments in Python begin with a hash mark #
 - Inline comments: you may write some code on a line, followed by some comments, e.g.

```
>>>name="April" #you may replace April with Summer, Winter, etc
```
 - Use three double quotes to """ start and end making paragraphs of comments

indentation is used to logically separated blocks of code.

Comments are important

- To explain the objective of a program
- To separate the beginning and/or end of logical or functional blocks
- To make note about special scenarios or exceptions

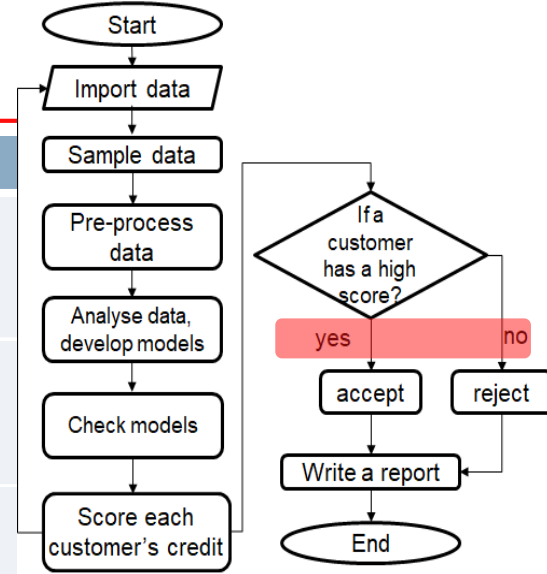
```
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
GHG-Maintenance.bib new 1 get-pip.py new 2 Session-4-Python-code-of-the-example.py new 3
1 import csv #to import the package csv
2 Student_Variables = [['StudentLogin','LastName','Programme']] # to define student variables
3 Mark_Variables = [['Programme','StudentLogin','Mark']] # to define mark variables
4 # data rows of csv file students
5 Student_Rows = [['A00010','POLLARD','Business Analytics'],
6                 ['B00020','SIMMS','SupplyChain'],
7                 ['C00030','CHAN','SupplyChain'],
8                 ['D00110','WHITEING','Banking'],
9                 ['E00120','MEYER','Business Analytics'],
10                ['F00130','HAMNAM','Business Analytics']]
11 # data rows of csv file student mark
12 Mark_Rows = [['Business Stats','A00010','70'],
13              ['Machine Learning','C00030','65'],
14              ['Simulation','E00120','58'],
15              ['BigData','F00130','81'],
16              ['Machine Learning','E00120','73'],
17              ['Machine Learning','G00120','65']] #values of marks
18
19 # name of csv file
20 filename = "student_records.csv"
21 # writing to csv file
22 with open(filename, 'w', newline='') as csvfile1:
23     # creating a csv writer object
24     csvwriter1 = csv.writer(csvfile1)
25
26     # writing the fields
27     csvwriter1.writerow(Student_Variables)
28     csvwriter1.writerow(Student_Rows)
29
30
31
32 filename = "mark_records.csv"
33 # writing to csv file
34 with open(filename, 'w', newline='') as csvfile2:
35     # creating a csv writer object
36     csvwriter2 = csv.writer(csvfile2)
37
```

Data types

- Name and age can be regarded with different types
- Basic data types
 - number: integer, float, Boolean, complex
 - string
 - list
 - tuple
 - dictionary

Data types--numbers

Data Type	Example	Verify
boolean	True, False	>>>x = True >>>print(type(x)) bool
integer	11, 23, 999	>>>x = 232 >>>print(type(x)) int
Float (floating point real values)	11.3, 23.863, 999.001	>>>x = 232.35 >>>print(type(x)) float



- Examples of Boolean type operations

>>>True and True

>>>True or False

- **Caution:** Decimal notation with . and not ,

1,2: Error: unexpected ',' in "1,"

Arithmetic Operators

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division
**	Exponentiation
%	Modulus (Remainder from division)
//	Floor division (the integer part of the result)

```
>>>x = 3
```

```
>>>y = 13
```

```
>>>x+y
```

```
[1] 16
```

```
>>>x-y
```

```
[1] -10
```

```
>>>x*y
```

```
[1] 39
```

```
>>>y/x
```

```
[1] 0.2307692
```

```
>>> y**x
```

```
[1] 2197
```

```
>>>y%x
```

```
[1] 1
```

```
>>>y//x
```

```
[1] 4
```

Relational Operators

Operator	Description
=	Assign
==	Equal to
!=	Not equal to
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to

- **x=6 means to assign 6 to x**
- **y = y+2: it is not y equals to y+2, it is "to assign y+2 to y"**
- **x==y is used to compare whether x and y are equal**

```
>>> x = 3
>>> y = 13
>>> x<y
True
>>> x>y
False
>>> x<=5
True
>>> y>=20
False
>>> y == 13
True
>>> x != 3
False
```

Logical Operators

Operator	Description	Example
not	Logical NOT	not y>13
and	Logical AND	x<5 and y>13
or	Logical OR	x<5 or y>13

- not: Reverse the result, returns False if the result is true
- and: Returns True if both statements are true
- or: Returns True if one of the statements is true

```
>>>y=13
>>>x=3
>>>not x==3
False
>>> y>13 and x<4
False
>>> y>13 or x<4
True
```

Data types—String operations

Data Type	Example	Verify
string	Included by single quotes or double quotes, but not a mixture 1) "it is a nice day" 2) "he likes \"eating\" apples" 3) "it is a nice day'---illegal: as it has " and ' a) ", ` , and ' are invalid "I am " b) 23.4 differs from '23.4' or "23.4"	>>>print("he likes \"eating\" apples") he likes "eating" apples

■ Difference

- a) “, `, and ’ are invalid “I am “
- b) 23.4 differs from '23.4' or "23.4"

Strings

rules	Example
<ul style="list-style-type: none">The quotes at the beginning and end of a string should be both double quotes or both single quote. They can not be mixed.	<pre>>>> a = 'Start and end with single quote' >>> print(a) [1] Start and end with single quote >>> b = "Start and end with double quotes" >>> print(b) [1] Start and end with single quote</pre>
<ul style="list-style-type: none">Double quotes can be inserted into a string starting and ending with single quote.Single quote can be inserted into a string starting and ending with double quotes.	<pre>>>> c = "single quote ' in between double quotes" >>> print(c) [1] single quote ' in between double quote >>> d = 'Double quotes " in between single quote' >>> print(d) [1] Double quote " in between single quote</pre>

Strings

rules	Example
<ul style="list-style-type: none">• 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.	<pre>>>> f = 'Single quote ' inside single quote' SyntaxError: invalid syntax >>> g = "Double quotes " inside double quotes" SyntaxError: invalid syntax</pre>

rules	Example
<ul style="list-style-type: none">• 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.	<pre>>>> h = 'Single quote \' inside single quote' >>> print(h) [1] Single quote ' inside single quote >>> k = "Double quotes \" inside double quotes" >>> print(k) [1] Double quotes " inside double quotes</pre>

Data types—String operations

- Exercise: String operations

```
>>>str = 'Welcome to the world of Python!'
```

```
>>> print(str)           # Prints complete string
```

```
>>> print(str[0])        # Prints first character of the string
```

```
>>> print(str[1:6])      # Prints characters starting from 2nd to 6th
```

```
>>> print(str[2:])       # Prints string starting from 3rd character
```


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
```

- methods in print()

```
>>> x = 12.3456789
```

```
>>> print('The value of x is %3.2f' %x)
```



Question: what is the result of: `print('The value of x is %3.4f' %x)`

Casting

- If you want to specify a type on to a variable, you may use
 - `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)
- `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=int(2);           # x will be 2
```

```
y=int(3.2);         # y will be 3
```

```
z=int("123")        # z will be 123
```

```
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
```

Casting

- `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"
```

 Question: what will happen if you enter

- `W1=int("apple")`
- `W2=float("apple")`

Vectors

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

- Vectors/matrices:

- Lists
- Tuples
- Sets
- Dictionaries

Lists

- Lists are used to store multiple items in a single variable. For example,

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

- List items are ordered:

```
>>>L1=[1,2,3];L2=[2,1,3]
```

```
>>>L1==L2
```

False #which suggest that L1 differs from L2

- List items can have duplicate values

```
>>>L3=[1,2,2,3]
```

```
>>>Anna=["Anna",72,65,55,65,62] #two 65 in the list
```

- Find the number of items in a list

```
>>>list4 = ["peach", "banana", "mellon"]
```

```
>>>print(len(list4))
```

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[0] #the 1st item in the list
- Modules[1] #the 2nd item in the list
- Modules[-1] #the last item in the list
- Modules[-2] #the second last 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
- Modules[2:] # the elements from the 3rd to the end in the list

 Question: what is Modules[-3:-1]?

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']