

MATH1166 Problem Solving and Mathematical Thinking Programming for Mathematics¹

Kayvan Nejabati Zenouz²

University of Greenwich

April 27, 2020

“This, therefore, is Mathematics; she reminds you of the invisible form of the soul; she gives life to her own discoveries; she awakens the mind and purifies the intellect; she brings light to our intrinsic ideas...”

PROCLUS 414 - 485 AD

¹ Use these notes in conjunction with Excel and Python demos.

² Office: QM315, Email: K.NejabatiZenouz@greenwich.ac.uk,
Student Drop-in Hours: MONDAYS 12:00-13:00 (MATHS ARCADE) AND FRIDAYS
14:00-15:00 (QM315)

- 1 **Introduction**
 - Module Aims and Assessment
 - Topics to be Covered
 - Reading List and References
- 2 **Week 5: Data and Visualisation with Excel**
 - What is Excel?
 - Data Entry and Functions
 - Visualisation Methods
 - Tables and Pivots
- 3 **Week 6: Introduction to Python 3**
 - What is Python?
 - Installing and Running
 - Basic Programming and Mathematics
 - Interfaces for Python: IDLE, Jupyter, Spyder
 - Packages: math, cmath, numpy
- 4 **Week 7: Data Types, Methods, and Programming**
 - Code and Data
 - Data Types: int, str, bool, float, complex
 - Variables and Assignments
 - Collection of Data: tuple, list, set, dict
 - Logical and Comparison Operations
 - First Programme
- 5 **Week 8: Conditional Statements and Loops**
 - if, else, and elif Statements
 - for and while Loops
 - break and continue Statements
- 6 **Week 9: Built-in and User-Defined Functions**
 - Functions in Mathematics
 - Built-in Functions
 - User-Defined Functions
 - Python Anonymous Functions lambda
- 7 **Week 10: Matrices, Dataframes, and Data Manipulation**
 - Matrices with numpy and sympy
 - Linear Algebra, Symbolic Mathematics, Calculus, and Number Theory with sympy
 - Data Manipulation and Visualisation with pandas
 - Import Data pandas.read_excel()
- 8 **Week 11: Statistics and Visualisation Methods**
 - Statistics with scipy
 - Plotting with matplotlib
 - Interactive Plots with plotly
- 9 **Week 12: Numerical Algorithms**
 - Introduction to Numerical Analysis
 - Roots of Nonlinear Equations
 - The Bisection Method
 - Error for Bisection Method

Lecture Contents

1

Introduction

- Module Aims and Assessment
- Topics to be Covered
- Reading List and References

2

Week 5: Data and Visualisation with Excel

- What is Excel?
- Data Entry and Functions
- Visualisation Methods
- Tables and Pivots

3

Week 6: Introduction to Python 3

- What is Python?
- Installing and Running
- Basic Programming and Mathematics
- Interfaces for Python: IDLE, Jupyter, Spyder
- Packages: math, cmath, numpy

4

Week 7: Data Types, Methods, and Programming

- Code and Data
- Data Types: int, str, bool, float, complex
- Variables and Assignments
- Collection of Data: tuple, list, set, dict
- Logical and Comparison Operations
- First Programme

5

Week 8: Conditional Statements and Loops

- if, else, and elif Statements
- for and while Loops
- break and continue Statements

6

Week 9: Built-in and User-Defined Functions

- Functions in Mathematics
- Built-in Functions
- User-Defined Functions
- Python Anonymous Functions lambda

7

Week 10: Matrices, Dataframes, and Data Manipulation

- Matrices with numpy and sympy
- Linear Algebra, Symbolic Mathematics, Calculus, and Number Theory with sympy
- Data Manipulation and Visualisation with pandas
- Import Data pandas.read_excel()

8

Week 11: Statistics and Visualisation Methods

- Statistics with scipy
- Plotting with matplotlib
- Interactive Plots with plotly

9

Week 12: Numerical Algorithms

- Introduction to Numerical Analysis
- Roots of Nonlinear Equations
- The Bisection Method
- Error for Bisection Method

Aims

The aim of this part of the module is to provide an overview of modern computer skills essential for mathematicians. In particular, by the end of this part you will be able to...

- 1 **Manipulate data** and **produce graphics** for visualisation using **Excel**.
- 2 Learn **programming** with **Python**, **data type**, create **functions**, **control flow**, **manipulate data**, and produce **visualisations**.
- 3 Design **programmes** for solving **mathematical** problems and **data analysis**.

Assessment

Computing Assignment (Electronic submission), weight 30%,
release 10/12/2019, due 06/02/2020.

Topics to be Covered...

- 1 Data and Visualisation with Excel
- 2 Introduction to Python 3
- 3 Data Types, Methods, and Programming
- 4 Conditional Statements and Loops
- 5 Built-in and User-Defined Functions
- 6 Matrices, Dataframes, and Data Manipulation
- 7 Statistics and Visualisation Methods
- 8 Numerical Algorithms

Guidance for Success

Attend Lectures, Engage with Tutorials, Ask Questions, Read Books, Use Online Resources (Google, YouTube, etc...), Keep Your Work Organised, and Always Ask for Help.

Suggested Reading List and References

For reading list see Guerrero (2018); Alexander et al. (2018); Kalb (2018); Summerfield (2008); Johansson (2015).

Alexander, M., R. Kusleika, and J. Walkenbach

2018. *Excel 2019 Bible*, Bible. Wiley.

Guerrero, H.

2018. *Excel Data Analysis: Modeling and Simulation*. Springer International Publishing.

Johansson, R.

2015. *Numerical Python: A Practical Techniques Approach for Industry*. Apress.

Kalb, I.

2018. *Learn to Program with Python 3: A Step-by-Step Guide to Programming*. Apress.

Summerfield, M.

2008. *Programming in Python 3: A Complete Introduction to the Python Language*. Pearson Education.

Class Activity with www.menti.com

Please **scan** the barcode with your **phone** in order to take part in the class activity.




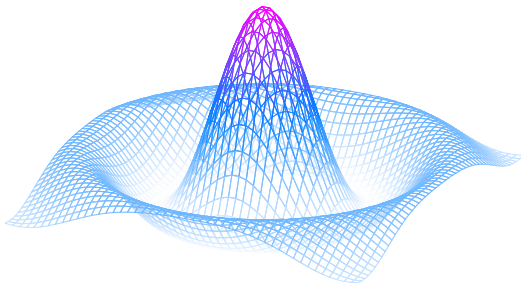
<https://www.menti.com/xt4nbvqhhu>

Alternatively, go to www.menti.com on your electronic devices and enter the access code **74 28 5**.

Week 5

Data and Visualisation with Excel


$$\frac{\sin(r)}{r}$$



Lecture Contents

- 1 **Introduction**
 - Module Aims and Assessment
 - Topics to be Covered
 - Reading List and References
- 2 **Week 5: Data and Visualisation with Excel**
 - What is Excel?
 - Data Entry and Functions
 - Visualisation Methods
 - Tables and Pivots
- 3 **Week 6: Introduction to Python 3**
 - What is Python?
 - Installing and Running
 - Basic Programming and Mathematics
 - Interfaces for Python: IDLE, Jupyter, Spyder
 - Packages: math, cmath, numpy
- 4 **Week 7: Data Types, Methods, and Programming**
 - Code and Data
 - Data Types: int, str, bool, float, complex
 - Variables and Assignments
 - Collection of Data: tuple, list, set, dict
 - Logical and Comparison Operations
 - First Programme
- 5 **Week 8: Conditional Statements and Loops**

- if, else, and elif Statements
 - for and while Loops
 - break and continue Statements
- 6 **Week 9: Built-in and User-Defined Functions**
 - Functions in Mathematics
 - Built-in Functions
 - User-Defined Functions
 - Python Anonymous Functions lambda
 - 7 **Week 10: Matrices, Dataframes, and Data Manipulation**
 - Matrices with numpy and sympy
 - Linear Algebra, Symbolic Mathematics, Calculus, and Number Theory with sympy
 - Data Manipulation and Visualisation with pandas
 - Import Data pandas.read_excel()
 - 8 **Week 11: Statistics and Visualisation Methods**
 - Statistics with scipy
 - Plotting with matplotlib
 - Interactive Plots with plotly
 - 9 **Week 12: Numerical Algorithms**
 - Introduction to Numerical Analysis
 - Roots of Nonlinear Equations
 - The Bisection Method
 - Error for Bisection Method

By the end of this session you will be able to...

- 1 Understand the basic uses of Excel.
- 2 Create graphs of functions and visualise qualitative or quantitative data.
- 3 Learn to modify the appearance of Excel graphics.
- 4 Produce summary tables from data.

What is Excel?

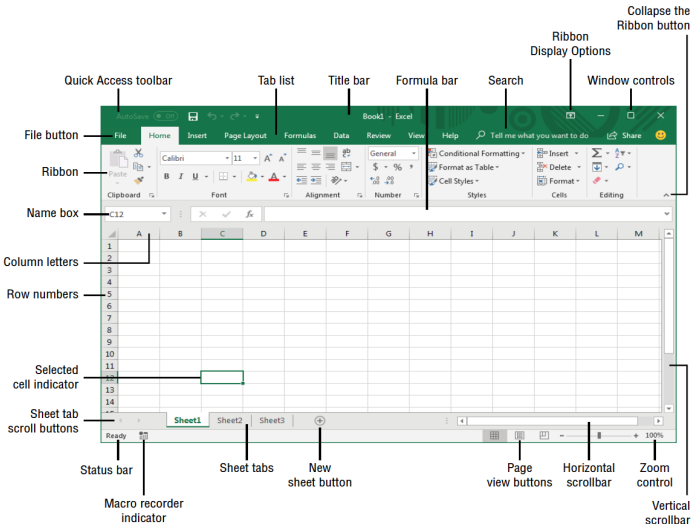
- Excel is **spreadsheet software** which is used for **storage, organisation, and analysis** of **tabular data**.
- First version was released by Microsoft in 1985 - it remains the most **popular** among **business applications** used for
 - ❶ **Number crunching**: create budgets, tabulate expenses, analyse survey results.
 - ❷ **Creating graphics**: customizable graphics and dashboards for business/academic reports.
 - ❸ **Data manipulation**: perform complex calculations on data as well as tools to manipulate text based data.

Remark 1: Why NOT Use Excel?

Unprecedented increase in **data production** has led businesses to use open source **fast performance** programming languages such as **Python** and **R** for visualisation and data analysis.

Excel Interface

We shall look at data entry, functions, graphics, and tables in Excel.

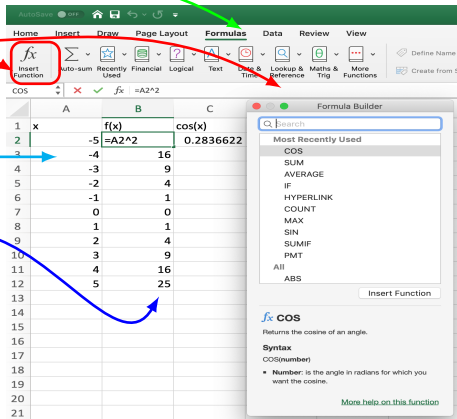


Data Entry and Functions

Use in-built **functions**

Click on “Formulæ”

then “Insert Function”



Data entry in the cells

Perform mathematical operations on data

Some In-Built Functions (here “=” means written as)

$$nx = n * x, x^n = x^{\wedge}n, \pi = \text{pi}()$$

$$\cos(x), \ln(x), e^x = \exp(x), \sqrt{x} = \text{sqrt}(x), \text{ etc...}$$

- In many scenarios we would like to have a **graphical representation** of data.
- Graphics can **summarise data** in a way that tables, or numerical values can never do.
- First we need to think about what **type of data** we are dealing with.

Quantitative Data

This is also referred to as numerical data; it comes as **continuous** or **discrete**.

Qualitative Data

This is also referred to as categorical data; it comes as **binary**, **nominal**, or **ordinal**.

Example

- **Quantitative:** weight or number of people in a room.
- **Qualitative:** true/false, red/blue/yellow, or good/OK/bad.

Exercise 1: Research and Find

- ❶ More **examples** of quantitative and qualitative data.
- ❷ Methods of **visualisation** for each quantitative and qualitative data.
- ❸ Meaning of **exploratory** and **explanatory** analysis.

Submit your findings in <https://www.menti.com/xt4nbvqhhu>.

Quantitative Visualisation

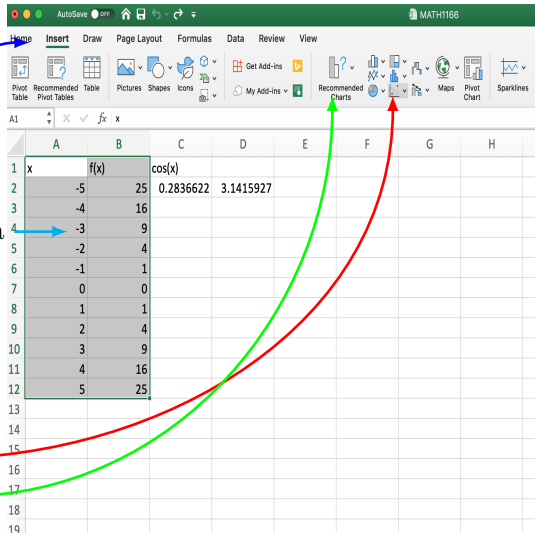
There are three steps:

2. Go to Insert

1. Select Range of Data

3. Choose Scatter

Or Recommended Charts



Quantitative Visualisation

Results:

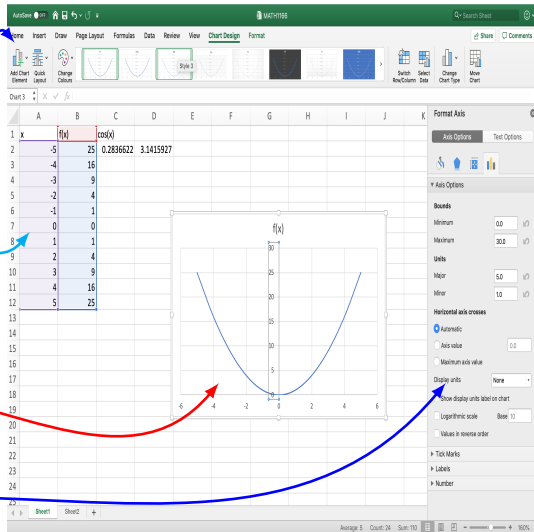
Design your graph

Range of Data

The graph

Customise layout

Double Click on Axis



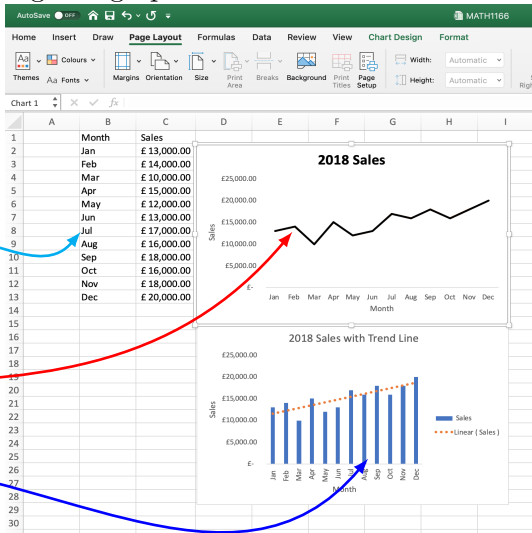
Exercise 2:

Create exactly the following two graphs.

Enter Sales Data

Line graph
with title and axis

Bar graph
add Trendline



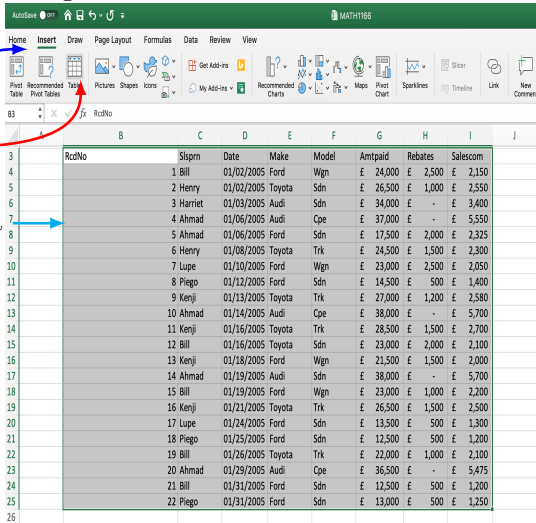
Qualitative Visualisation: Tables

Suppose we have the following data.

2. Go to Insert

3. Choose Table

1. Select Range of Data



The screenshot shows the Microsoft Excel interface with the 'Insert' tab selected. The ribbon includes options for Pivot Tables, Recommended Pivot Tables, Pictures, Shapes, Icons, Charts, My Add-ins, Recommended Charts, Maps, Pivot Chart, Sparklines, Slicer, Timeline, Link, and New Comment. The data table is located in the range B3:I25. The table has 8 columns: RcdNo, Slspn, Date, Make, Model, Amtpaid, Rebates, and Salescom. The data is as follows:

RcdNo	Slspn	Date	Make	Model	Amtpaid	Rebates	Salescom
1	Bill	01/02/2005	Ford	Wgn	£ 24,000	£ 2,500	£ 2,150
2	Henry	01/02/2005	Toyota	Sdn	£ 26,500	£ 1,000	£ 2,550
3	Harriet	01/03/2005	Audi	Sdn	£ 34,000	£ -	£ 3,400
4	Ahmad	01/06/2005	Audi	Cpe	£ 37,000	£ -	£ 5,550
5	Ahmad	01/06/2005	Ford	Sdn	£ 17,500	£ 2,000	£ 2,325
6	Henry	01/08/2005	Toyota	Trk	£ 24,500	£ 1,500	£ 2,300
7	Lupe	01/10/2005	Ford	Wgn	£ 23,000	£ 2,500	£ 2,050
8	Piego	01/12/2005	Ford	Sdn	£ 14,500	£ 500	£ 1,400
9	Kenji	01/13/2005	Toyota	Trk	£ 27,000	£ 1,200	£ 2,580
10	Ahmad	01/14/2005	Audi	Cpe	£ 38,000	£ -	£ 5,700
11	Kenji	01/16/2005	Toyota	Trk	£ 28,500	£ 1,500	£ 2,700
12	Bill	01/16/2005	Toyota	Sdn	£ 23,000	£ 2,000	£ 2,100
13	Kenji	01/18/2005	Ford	Wgn	£ 21,500	£ 1,500	£ 2,000
14	Ahmad	01/19/2005	Audi	Sdn	£ 38,000	£ -	£ 5,700
15	Bill	01/19/2005	Ford	Wgn	£ 23,000	£ 1,000	£ 2,200
16	Kenji	01/21/2005	Toyota	Trk	£ 26,500	£ 1,500	£ 2,500
17	Lupe	01/24/2005	Ford	Sdn	£ 13,500	£ 500	£ 1,300
18	Piego	01/25/2005	Ford	Sdn	£ 12,500	£ 500	£ 1,200
19	Bill	01/26/2005	Toyota	Trk	£ 22,000	£ 1,000	£ 2,100
20	Ahmad	01/29/2005	Audi	Cpe	£ 36,500	£ -	£ 5,475
21	Bill	01/31/2005	Ford	Sdn	£ 12,500	£ 500	£ 1,200
22	Piego	01/31/2005	Ford	Sdn	£ 13,000	£ 500	£ 1,250

Read more on Guerrero 2018, Chapter 2 or Alexander et al. 2018, Part III.

Qualitative Visualisation: Pivot Tables

We can create summary tables first and then use visualisation.

2. Click to Summarise with Pivot Table click OK

1. Table of Data

The screenshot shows the Microsoft Excel interface. The 'Table of Data' is a table with columns RcdNo, Slsprn, Date, Make, and Model. The 'Create Pivot Table' dialog box is open, showing the 'Table/Range' as 'Table7' and the 'New worksheet' option selected. The 'Table of Data' is a table with columns RcdNo, Slsprn, Date, Make, and Model. The 'Create Pivot Table' dialog box is open, showing the 'Table/Range' as 'Table7' and the 'New worksheet' option selected.

RcdNo	Slsprn	Date	Make	Model
1	Bill	01/02/2005	Ford	Wgn
2	Henry	01/02/2005	Toyota	Sdn
3	Harriet	01/03/2005	Audi	Sdn
4	Ahmad	01/06/2005	Audi	Cpe
5	Ahmad	01/06/2005	Ford	Sdn
6	Henry	01/08/2005	Toyota	Trk
7	Lupe	01/10/2005	Ford	Wgn
8	Piegi	01/12/2005	Ford	Sdn
9	Kenji	01/13/2005	Toyota	Trk
10	Ahmad	01/14/2005	Audi	Cpe
11	Kenji	01/16/2005	Toyota	Trk
12	Bill	01/16/2005	Toyota	Sdn
13	Kenji	01/18/2005	Ford	Wgn
14	Ahmad	01/19/2005	Audi	Sdn
15	Bill	01/19/2005	Ford	Wgn
16	Kenji	01/21/2005	Toyota	Trk
17	Lupe	01/24/2005	Ford	Sdn
18	Piegi	01/25/2005	Ford	Sdn
19	Bill	01/26/2005	Toyota	Trk
20	Ahmad	01/29/2005	Audi	Cpe
21	Bill	01/31/2005	Ford	Sdn
22	Piegi	01/31/2005	Ford	Sdn

Qualitative Visualisation: Pivot Tables

Results. Play around with options on the right hand side to obtained desired tables.

Customise and choose

Pivot Table of Data

The screenshot shows the Microsoft Excel interface with a PivotTable and the PivotTable Fields task pane. The PivotTable is located in the range A3:J21 and has the following data:

Row Labels	Sum of RodNo	Count of Date	Count of Model	Sum of Ampaid	Sum of Rebates	Sum of Salescom
Ahmad	53	5	5	167000	2000	24750
Audi	48	4	4	149500	0	22425
Ford	5	1	1	17500	2000	2325
Bill	68	5	5	104500	7000	9750
Ford	37	3	3	59500	4000	5550
Toyota	31	2	2	45000	3000	4200
Harriet	3	1	1	34000	0	3400
Audi	3	1	1	34000	0	3400
Henry	8	2	2	51000	2500	4850
Toyota	8	2	2	51000	2500	4850
Kenji	49	4	4	103500	5700	9780
Ford	13	1	1	21500	1500	2000
Toyota	36	3	3	82000	4200	7780
Lupe	24	2	2	36500	3000	3350
Ford	24	2	2	36500	3000	3350
Piego	48	3	3	40000	1500	3850
Ford	48	3	3	40000	1500	3850
Grand Total	253	22	22	536500	21700	59730

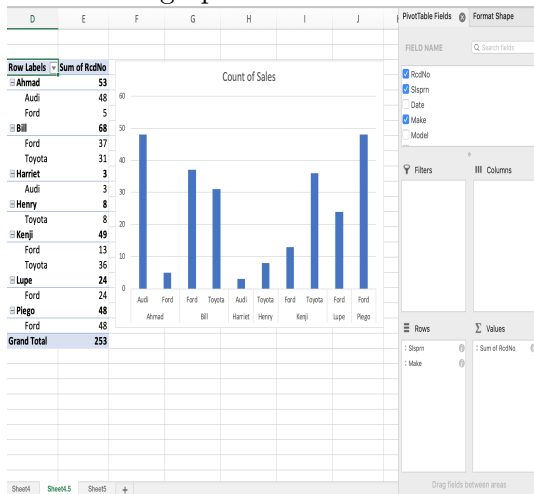
The PivotTable Fields task pane on the right shows the following configuration:

- PivotTable Fields:** RodNo, Status, Date, Make, Model
- Filters:** (Empty)
- Columns:** Values
- Rows:** Status, Make
- Values:** Sum of RodNo, Count of Date, Count of Model, Sum of Ampaid, Sum of Rebates, Sum of Salescom

Exercise 3:

Create the following pivot table and graphics.

Table of Data



Read more on Guerrero 2018, Chapter 4 or Alexander et al. 2018, Part IV.

Important to Remember

Audience

Understand your audience and context. *A graphics can be worth a thousand words*, or confusing if badly produced!

Research

Choose visuals carefully, compare several and go for the best, or learn from experts through prior research.

Clutter and Design

Avoid clutter: remember sometimes less is more. Design your graphics professionally. Avoid using pie charts, doughnut charts, or 3D bar charts.

Storytelling

Think about your audience and what you want them to see. Accompany graphics with a brief story of what they show.

What we did today...

Basics of Excel

History and applications

Data

Types and visualisations

Do's and Dont's

Remember them

Exercises

Go through and check solutions

Need Help?

Email me or ask me any questions

Next Time

Python 3 ♥♥♥

Week 6

Introduction to Python 3

```
# Python 3: Fibonacci Series up to n
def fib(n):
    a, b = 0, 1
    while a < n:
        print(a, end=' ')
        a, b = b, a+b
fib(1000)
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987
```



Lecture Contents

1

Introduction

- Module Aims and Assessment
- Topics to be Covered
- Reading List and References

2

Week 5: Data and Visualisation with Excel

- What is Excel?
- Data Entry and Functions
- Visualisation Methods
- Tables and Pivots

3

Week 6: Introduction to Python 3

- What is Python?
- Installing and Running
- Basic Programming and Mathematics
- Interfaces for Python: IDLE, Jupyter, Spyder
- Packages: math, cmath, numpy

4

Week 7: Data Types, Methods, and Programming

- Code and Data
- Data Types: int, str, bool, float, complex
- Variables and Assignments
- Collection of Data: tuple, list, set, dict
- Logical and Comparison Operations
- First Programme

5

Week 8: Conditional Statements and Loops

- if, else, and elif Statements
- for and while Loops
- break and continue Statements

6

Week 9: Built-in and User-Defined Functions

- Functions in Mathematics
- Built-in Functions
- User-Defined Functions
- Python Anonymous Functions lambda

7

Week 10: Matrices, Dataframes, and Data Manipulation

- Matrices with numpy and sympy
- Linear Algebra, Symbolic Mathematics, Calculus, and Number Theory with sympy
- Data Manipulation and Visualisation with pandas
- Import Data pandas.read_excel()

8

Week 11: Statistics and Visualisation Methods

- Statistics with scipy
- Plotting with matplotlib
- Interactive Plots with plotly

9

Week 12: Numerical Algorithms

- Introduction to Numerical Analysis
- Roots of Nonlinear Equations
- The Bisection Method
- Error for Bisection Method

By the end of this session you will be able to...

- 1 Understand what Python is as a programming language.
- 2 Learn how to install and work with different interfaces of Python.
- 3 Perform basic operations and mathematical calculations.
- 4 Learn about Python packages and how to use some of them.

What is Python?

- Python is an **Object-Oriented, General Purpose, Interpreted**, programming language.
- It was created in 1990s by Guido van Rossum, and it has become the **easiest-to-learn** and **niciest-to-use** programming language.
- Python is used to write **applications** to solve problems in **mathematics, numerical and financial analysis, neurosciences, biology**, and many others areas
<https://www.youtube.com/watch?v=hxGB7LU4i1I>.
- It remains the **top choice** of language for **data science** in industry alongside the more statistical language **R**.
- After learning Python it will be **easier** to learn other programming languages.

Exercise 1: Research to Find Definition and Example

- i An algorithm
- ii A computer programme
- iii Programming language (including higher/lower)
- iv General Purpose language
- v Object-Oriented programming
- vi Interpreted and Compiled programming language

Submit your findings in <https://www.menti.com/sdoiq8pes>.

How to Get Python?

Getting Python

Python is **open source**, freely available, from
www.python.org

there are two versions: old Python 2 and new Python 3. We will learn **Python 3**.

Online Documentations

The above website also contains vast amount of information about Python and **should be your first** point of contact. For example, go to **Tutorial** on

<https://docs.python.org/3>

Remark 1: Google Everything!

Remember **programming** and **Googling** go hand in hand. Make sure you use the extensive online resources available.

Python

Once installed, open a **Python shell**, type python in the **Command Prompt** (Windows) or **Terminal** (MAC/Linux). Or click on any Python icon you find on your system.

The image shows two side-by-side terminal windows. The left window is a Windows Command Prompt, and the right window is a macOS Terminal. Red, blue, and green arrows point from text labels on the left to specific lines of code in the terminal windows.

Write python (Red arrow pointing to `python` in the Windows Command Prompt)

Test Python (Blue arrow pointing to `print("Hello World")` in the Windows Command Prompt)

Use as Calculator (Blue arrow pointing to `2+2` in the Windows Command Prompt)

To close type quit() (Green arrow pointing to `quit()` in the Windows Command Prompt)

Windows Command Prompt Output:

```
Microsoft Windows [Version 10.0.16299.1768]
(c) 2017 Microsoft Corporation. All rights reserved.

G:\>python
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 17:00:18) [MSC v.19
00 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more informa
tion.
>>> print("Hello World")
Hello World
>>> 2+2
4
>>>
>>>
>>> quit()
G:\>
```

macOS Terminal Output:

```
Last login: Fri Aug 9 16:07:26 on ttys001
(base) Kayvans-MacBook-Pro:~ Kayvan$ python
Python 3.7.3 (default, Mar 27 2019, 16:54:48)
[Clang 4.0.1 (tags/RELEASE_401/final)] :: Anaconda, Inc. on
darwin
Type "help", "copyright", "credits" or "license" for more i
nformation.
>>> print("Hello World")
Hello World
>>> 2+2
4
>>>
>>>
>>> quit()
(base) Kayvans-MacBook-Pro:~ Kayvan$
```

- A programme is **set of instructions** asking a computer to preform a task.
- It is a way of **speaking to machines** and you need to be **precise, logical**, and **understand the rules** of each language carefully.
- Computer reads your programme in the order of appearance. For example, the code `print("Hello World")` produces

```
>>> print("Hello World!")  
Hello World!
```

- Function `print` is used to tell Python to display an output.
Note, Python functions are case sensitive!

```
>>> Print("Hello World!")  
Traceback (most recent call last):  
File "<stdin>", line 1, in <module>  
NameError: name 'Print' is not defined
```


Remark 2: Indentation Matters in Python!

Indentation of code helps with the **readability** of programmes, but will produce **errors** if used incorrectly!

```
>>> print("Hello World!")
Hello World!
>>> print("Hello World!")
File "<stdin>", line 1
print("Hello World!")
^
IndentationError: unexpected indent
```

Remark 3: Always Comment Your Code!

Anything that comes after a **#** will **not be executed** by Python, so you can use this to comment your codes

```
>>> print("Hello World!") # This will print
Hello World!
```

- You can use Python as a **calculator** with operations

$$+, -, *, /, **$$

for addition, subtraction, multiplication, division, and exponentiation. For example, $\left(\frac{(2+5)\times 5}{10}\right)^2$ is written

```
>>> ((2+5)*5/10)**2  
12.25
```

- The **order** of mathematical **operations** follows **PEMDAS**:
 - ❶ Parentheses
 - ❷ Exponents
 - ❸ Multiplication
 - ❹ Division
 - ❺ Addition
 - ❻ Subtraction

Exercise 2: Research - Interfaces for Python

- Find from internet if you can get **Python 3** on your **phone**, or smart devices.
- Find suitable online Python Compilers, e.g., https://www.tutorialspoint.com/execute_python_online.php.
- Type the following codes to see what you get.

```
1  print("Hello World")
2  2+2
3  print(2+2)
4  print("2+2")
5  print('2+2')
6  # print("2+2")
7  x=2
8  x+2
9  help(print)
10 Hello World
```

- Write the code on first slide of this session (slide 25).

Interfaces for Python and IDLE

Programming in the terminal can be difficult and time consuming. For this reason Python programmers often use **user friendly interfaces** and **scientific distributions**.

IDLE

The Integrated Development and Learning Environment (IDLE) for Python offers many features; for example, **ability to save codes, code colouring, smart indent, auto completion, interactivity**, debugger, and many others.

Anaconda

Another way to get access to Python interfaces is through scientific distribution **Anaconda**, available from

www.anaconda.com

already equipped with many packages and interfaces such as **Spyder, Jupyter, JupyterLab**.

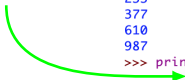
IDLE interface comes with Python installation.

Code colouring



Smart indent

Auto completion

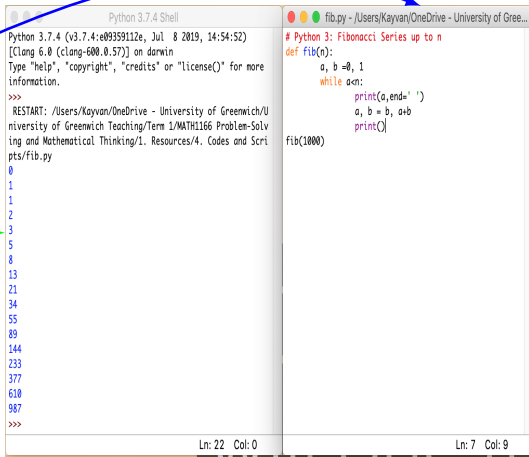
A screenshot of the Python 3.7.4 Shell window. The window title is '*Python 3.7.4 Shell*'. The code editor shows a Fibonacci function definition with syntax highlighting: 'def' and 'while' are orange, 'fib' is blue, 'a, b = 0, 1' is black, and 'print' is purple. A red arrow points from 'Code colouring' to the code. A blue arrow points from 'Smart indent' to the code. The code is executed, and the output is shown in the console: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987. A green arrow points from 'Auto completion' to the console. The console shows the prompt '>>> print(' followed by a dropdown menu with the option 'print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)'. The status bar at the bottom right shows 'Ln: 30 Col: 10'.

You can save your code by creating a *.py* file by going to File and selecting New File.

Write code and save

From Run choose
Run Module

Results



```
Python 3.7.4 Shell
Python 3.7.4 (v3.7.4:09359112e, Jul 8 2019, 14:54:52)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license()" for more
information.
>>>
RESTART: /Users/Kayvan/OneDrive - University of Greenwich/U
niversity of Greenwich Teaching/Term 1/MATH1166 Problem-Solv
ing and Mathematical Thinking/1. Resources/4. Codes and Scri
pts/Fib.py
0
1
1
2
3
5
8
13
21
34
55
89
144
233
377
610
987
>>>
Ln: 22 Col: 0
```

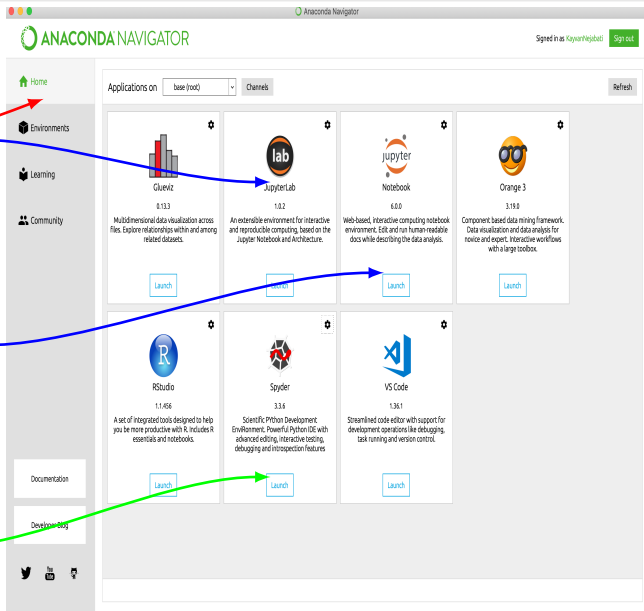
```
fib.py - /Users/Kayvan/OneDrive - University of Gree...
# Python 3: Fibonacci Series up to n
def fib(n):
    a, b = 0, 1
    while a < n:
        print(a, end=" ")
        a, b = b, a+b
    print()
fib(1000)
Ln: 7 Col: 9
```

Anaconda Interface

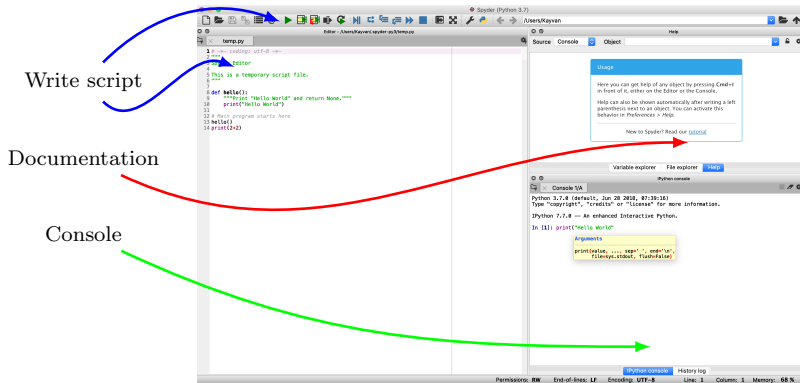
Anaconda Navigator

Jupyter

Spyder

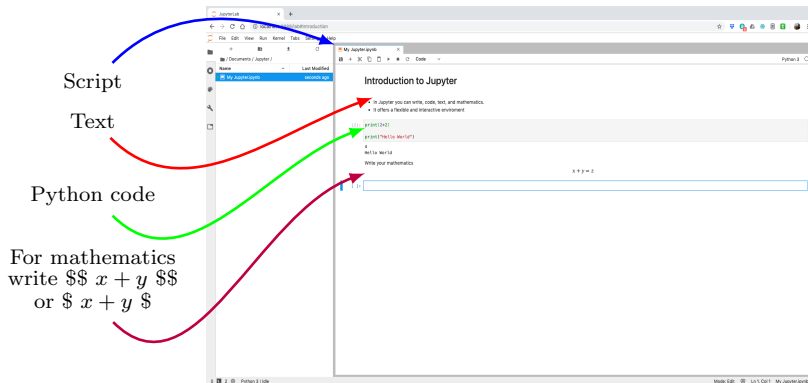


- Spyder www.spyder-ide.org is a **scientific environment** in Python. You can access it through Anaconda.
- It has advanced features for **editing**, **analysis**, debugging, **interactive execution**, and beautiful **visualisation** capabilities. You can add **Notebook** and **Reports**.



Jupyter(Lab)

- Jupyter(Lab) www.jupyter.org is an **interactive** development environment.
- Jupyter(Lab) opens in a **browser** and allows to write **code**, **text**, and **mathematics** in the same script to produce interactive documents.



- While at the **University of Greenwich** you should be able to access Python and Anaconda on most computers.
- Alternatively you can also get access to them through **Virtual Lab Desktop**.
- Find **instructions** on how to use Virtual Lab Desktop on <http://ach-support.gre.ac.uk/labdesktop/>.
- You can install this at **home** or on a **tablet, phone** or other device, following these instructions.

Packages for Python

- No all the **tools** that programmers need are already built into Python.
- **Packages**, Modules, and Libraries **increase** the **functionality** of Python by offering tools useful for particular tasks.
- Some packages already exist in Python (for example `pip`, `math`, `statistics`, etc...) and some need to be **installed**.
- To load an existing package use `import` in Python

```
>>> import <package-name>
```

- To install³ use `pip` in the terminal

```
$> pip install <package-name>
```

- To update a package to the latest version use

```
$> pip install -U <package-name>
```

³ Anything starting with a \$ is to be executed in **Command (Anaconda) Prompt/Terminal**. All other codes, and those with `>>>`, to be executed in Python.

Anaconda

- Has many packages already installed.
- You can **update** Anaconda by running

```
$> conda update conda
```
- You can **install** a package by

```
$> conda install <package-name>
```
- You can **update a package** by

```
$> conda update <package-name>
```
- You can **update all** packages by

```
$> conda update --all
```
- Finally to remove a package use

```
$> conda remove <package-name>
```

Exercise 3: Playing with numpy

NumPy provides multidimensional arrays functions to operate on arrays. To load type

```
import numpy
```

- Figure out what the following lines of code do.

```
1  a = numpy.array([[6, 7, 8],[1, 2, 3]])
2  a
3  a.shape
4  a.ndim
5  a.size
6  type(a)
7  numpy.arange(15).reshape(3, 5)
8  numpy.zeros((3,4))
```

- You can load a package and give it a different name.

```
import numpy as np
a = np.array([[6, 7, 8],[1, 2, 3]])
```

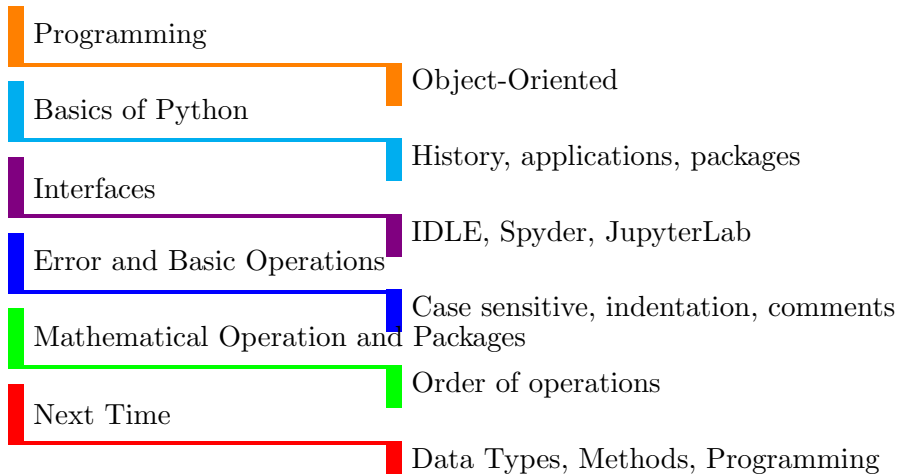
Some useful and well-known packages of Python which we shall learn about.

- ❶ **NumPy** www.numpy.org operation on data stored in arrays, linear algebra, Fourier transform, and random number.
- ❷ **SymPy** www.sympy.org is used for symbolic mathematics, calculus, and number theory.
- ❸ **Pandas** www.pandas.pydata.org provides high-performance, easy-to-use data structures and data analysis tools.
- ❹ **SciPy** www.scipy.org mathematics, statistics, science, and engineering, integration, differentiation, gradient optimization.
- ❺ **Matplotlib** www.matplotlib.org is a Python 2D plotting library which produces publication quality figures.
- ❻ **Ployly** www.plot.ly/python Plotly's Python graphing library makes interactive, publication-quality graphs.

More sophisticated well-known packages of Python.

- ❶ **Scikit-learn** www.scikit-learn.org/stable simple and efficient tools for data mining and data analysis.
- ❷ **Theano** www.deeplearning.net/software/theano, define, optimize, and evaluate mathematical expressions involving multi-dimensional arrays efficiently.
- ❸ **Beautiful Soup** www.crummy.com/software/BeautifulSoup Pythonic idioms for navigating, searching, and modifying a parse tree in HTML for extracting data from the web.
- ❹ **Seaborn** www.seaborn.pydata.org/ data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics.

What we did today...



Programming	
Basics of Python	Object-Oriented
Interfaces	History, applications, packages
Error and Basic Operations	IDLE, Spyder, JupyterLab
Mathematical Operation and Packages	Case sensitive, indentation, comments
Next Time	Order of operations
	Data Types, Methods, Programming

Week 7

Data Types, Methods, and Programming

Keywords in Python

False		class		finally		is		return
None		continue		for		lambda		try
True		in		nonlocal		while		raise
and		del		global		not		with
as		elif		if		or		yield
assert		else		import		pass		
break		except		def		from		

1

Introduction

- Module Aims and Assessment
- Topics to be Covered
- Reading List and References

2

Week 5: Data and Visualisation with Excel

- What is Excel?
- Data Entry and Functions
- Visualisation Methods
- Tables and Pivots

3

Week 6: Introduction to Python 3

- What is Python?
- Installing and Running
- Basic Programming and Mathematics
- Interfaces for Python: IDLE, Jupyter, Spyder
- Packages: math, cmath, numpy

4

Week 7: Data Types, Methods, and Programming

- Code and Data
- Data Types: int, str, bool, float, complex
- Variables and Assignments
- Collection of Data: tuple, list, set, dict
- Logical and Comparison Operations
- First Programme

5

Week 8: Conditional Statements and Loops

- if, else, and elif Statements
- for and while Loops
- break and continue Statements

6

Week 9: Built-in and User-Defined Functions

- Functions in Mathematics
- Built-in Functions
- User-Defined Functions
- Python Anonymous Functions lambda

7

Week 10: Matrices, Dataframes, and Data Manipulation

- Matrices with numpy and sympy
- Linear Algebra, Symbolic Mathematics, Calculus, and Number Theory with sympy
- Data Manipulation and Visualisation with pandas
- Import Data pandas.read_excel()

8

Week 11: Statistics and Visualisation Methods

- Statistics with scipy
- Plotting with matplotlib
- Interactive Plots with plotly

9

Week 12: Numerical Algorithms

- Introduction to Numerical Analysis
- Roots of Nonlinear Equations
- The Bisection Method
- Error for Bisection Method

By the end of this session you will be able to...

- 1 Understand different data types - integer, floats, strings, Boolean, and methods.
- 2 Learn about variables, assignment statements, conventions, and keywords.
- 3 Manipulate different data collection types - tuples, lists, sets, dictionaries, and ranges.
- 4 Learn about logical and comparison operations.
- 5 Apply the concepts learnt to write and test Python programmes.

Last Week

We learnt how to use Python with different interfaces and played around with codes and packages.

- **Think about** what we learnt last week: choices for programming in Python; mathematical operators and their order in Python; **print** function; and packages and answer questions in <https://www.menti.com/im5r1bitpb>.

This Week

- We look at more **technical aspects** of programming and how to use Python.
- This is very useful as you will develop a concrete knowledge of programming which will remain with you as when learning other languages.
- You will research and programme during the session through exercises in order to gain a fully proficiency of the concepts.

- **Code and Data**

The **main building** blocks of programming are code and data.

- **Code** (or method) is a set of **instructions** that tell the computer what to perform and how it should perform.
- **Data** refers to the **quantities, characters, and/or symbols** on which operations are performed. Anything the computer needs to remember is a piece of data.

- **RAM and CPU**

Inside a computer...

- Ready **programme** is loaded into **RAM**, random access memory.
- The **CPU**, central processing unit, **executes** the instructions of the programme.
- Instructions usually boil down to **load data, preform arithmetic, make comparisons, and store data.**

Example (Data)

- Number of students in class
- Examination marks average
- Name of students
- Whether a switch is in an on or off position
- Solutions of the equation $x^2 + 1 = 0$

Question

What is the difference between the data in the examples above?

Types of Data

In we shall look at four main types.

- **Integers** numbers (integers) are counting numbers, like 1, 2, 3, but also include 0 and negative numbers, e.g., number of people in a room.
- **Floating-point** numbers (floats) are numbers that have a decimal point in them, e.g., price of beard; irrational numbers.
- **Strings** (text) are any sequences of characters e.g., name, address.
- **Booleans** are a type of data that can only have one of two values: True or False, e.g., the state of a light switch: True for on, False for off.

Remark 1:

- 1 Integers seem to be included in floats, but computer has easier time dealing with integers than floats!
- 2 There are many types of data for example MP3, JPG, PDF.

Type Function

You can use the function `type()` to determine the type of data.

Exercise 1: Research and Find

- ❶ Five examples for each of the four data type integers, floats, strings, and Booleans.
- ❷ Suppose you are required to create the following table in Python

Receipt of Purchase

Name:
Date of Birth:
Address:
Phone Number:
Number of Items:
Total Price:
Deposit Paid?

what data type does each row belong to?

Variables and Assignment Statements

- **Variables** are used in order to store and manipulate data. A variable is a named memory location that holds a value.
- Memory locations in **RAM** can be used as a **variable**. Different types of data take up different amount of memory.
- For example, when you play a **computer game** a variable is created which holds your **score** at each point in time.
- An **assignment statement** is a line of code in which a variable is given a value. It has the general form

`<variable> = <expression>`

- Below variables on the left of = are assigned the values on the right of "=" sign

`age = 29`

`name = 'Fred'`

`alive = True`

`gpa = 3.9`

- Note that "=" sign is called the **assignment operator**.

Names and Rules

By definition all variables must have a name. There are some rules governing this.

- Must start with a letter (or an underscore).
- Cannot start with a digit or be a keyword.
- Can have up to 256 total characters.
- Can include letters, digits, underscores, dollar signs, and other characters.
- Cannot contain spaces or maths symbols (+, -, /, *, %, parentheses).

For example you can have `numberOfFishInAquarium`, but not `49ers`, `table+chairs`, `my age`, `import`, or `(coins)`.

Python converts your code into a language of ones and zeros, when the compiler reads your code, it looks for special words called **keywords** to understand what your code is trying to say.

- Sometimes we would like to hold a **collections** of data items, sets/vectors of objects, rather than variables.
- This is done through various methods, **Sequence**, **Sets**, and **Mapping** types.
- These **Sequence Operations (Methods)**, for example **membership in** and **length len()**.

Tuples

A tuple is an **ordered sequence** of zero or more object references. You can create a tuple by separating items with **commas (and round brackets)**, for example,

```
t=("venus", -28, "gre", "21", 19.74, "-28")
```

- It is easy to **extract** items from a tuple, but we **cannot** replace or delete any of their items (**immutable**).
- The code `x in t` is a sequence operation which results in **True** if an item of `t` is equal to `x` and **False** otherwise.

Exercise 2: Research and Find

Common sequence operations valid for tuples and write them with definition on <https://www.menti.com/im5r1bitpb>. **Hint:** You may use the Python Library Reference <https://docs.python.org/3>.

Lists

A list is an **ordered sequence** of zero or more object references. You can create a list by separating items with **commas and square brackets**, for example,

```
L=[-17.5, "kilo", ("ram", 5, "echo"), 7]
```

- It is easy to extract items from a list, and we **can** replace and delete any of their items (**mutable**).
- It is also possible to insert, replace, and delete slices of lists.
- There are many methods for lists, e.g., `L.append(x)` appends item `x` to the end of list `L`.

Exercise 3: Research and Find

Methods (sequence operations) for mutable sequence types and write them with definition on <https://www.menti.com/im5r1bitpb>.

Sets

A set is an **unordered collection** object references that refer to **hashable^a** objects. You can create a set by separating items with **commas and curly brackets**, for example,

```
S={7, "veil", 0, ("x", 11), frozenset([8, -4, 7]),  
  "sun"}
```

- Sets are mutable, so we can easily add or remove items.
- Since they are unordered, no notion of index position, and so cannot be sliced strided.
- Usual set operations: | union, & intersection, - set difference, ^ symmetric difference.

^a All of Python's immutable built-in objects are hashable, while no mutable containers (such as lists or dictionaries) are.

Dictionaries

A dict is an **unordered collection** of zero or more key–value pairs whose **keys** are object references that refer to **hashable** objects, and whose **values** are object references referring to objects of **any type**. For example, if you want to keep data for a customer's Name: George, Age: 29, and Sex: Male, you can use

```
C=dict({"Name":"George", "Age":29, "Sex":"Male"})
```

- Dictionaries are mutable, so we can easily add or remove items
- They are unordered, like sets, so no notion of index position and cannot be sliced or strided.
- There are several ways of creating dictionaries.
- You can use the usual set operations.

- We can test objects for **truth**. For example, testing if `x` belongs to a set `S` or two objects are the same.
- By default, an object is considered **True** unless its class defines either a boolean method that returns **False** or a has length 0. Truth can be **tested** using `bool()`.
- Some built-in objects considered false:
 - Constants defined to be false: `None` and `False`.
 - Zero of any numeric type: `0`, `0.0`, `0j`, `Decimal(0)`, `Fraction(0, 1)`.
 - Empty sequences and collections: `''`, `()`, `[]`, `set()`, `range(0)`.
- Testing for truth is important when using `if` or `while` statements (i.e., doing something if/while something is true).

Python has four types logical operators.

- The **Boolean Operations** are **and**, **or**, **not**; for example,

```
0 or 1
"Ali" and 2
not 1
```

They work as follows

<code>x or y</code>	if <code>x</code> is false, then <code>y</code> , else <code>x</code>
<code>x and y</code>	if <code>x</code> is false, then <code>x</code> , else <code>y</code>
<code>not x</code>	if <code>x</code> is false, then <code>True</code> , else <code>False</code>

- The **Comparison** operators are `<`, `<=`, `>`, `>=`, `==`, `!=`, `is`, `is not`.

Exercise 4: Research

Find and discuss what each of the comparison operators mean.

Having understood the basics of Python, we can now start creating computer programmes.

To Write a Programme We Need To

- 1 Have a **problem** and we need to state it as **clearly** as possible.
- 2 Determine what the **input** is, i.e., what information is given.
- 3 Know how to **solve** the problem **theoretically**, by hand, and for simple examples.
- 4 Decide what the **output** should be.
- 5 Write the **script** that allows for the input, performs calculation, and returns output.
- 6 **Test** the script on several examples and check for efficiency.

Example

Problem

Write a programme which **solves** any **quadratic equation** with real coefficient and returns two solutions.

Programme Steps 1, 2, 3, 4

- 1 Find the solutions for any equation

$$ax^2 + bx + c = 0 \text{ where } a \neq 0.$$

- 2 Input is real numbers a, b, c , with $a \neq 0$.

- 3 Solve by hand

$$ax^2 + bx + c = 0 \implies x^2 + \frac{b}{a}x + \frac{c}{a} = 0 \implies$$

$$x^2 + 2\frac{b}{2a}x + \frac{b^2}{4a^2} - \frac{b^2}{4a^2} + \frac{c}{a} = 0 \implies \left(x + \frac{b}{2a}\right)^2 = \frac{b^2}{4a^2} - \frac{c}{a}$$

$$\implies x - \frac{b}{2a} = \pm \frac{\sqrt{b^2 - 4ac}}{2a} \implies x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a}, \quad x_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}.$$

- 4 Output should be x_1 and x_2 .

Example Cont. I Programme Step 5

Following is the first attempt at creating the script.

```
1  # Input, you can change these and rerun.
2  a=1;b=1;c=0
3
4  # Here we do the calculations.
5  import math # For taking square root
6  # we need to import the package math
7  delta=b**2-4*a*c
8  firstSolution=(-b+math.sqrt(delta))/2*a
9  secondSolution=(-b-math.sqrt(delta))/2*a
10
11 # Use print function to display output
12 print("The first solution is", firstSolution, "and
    the second solution", secondSolution)
```

Programme Step 6

Test the script for following values and make sure you get the correct solution.

Values No	a	b	c
1	1	1	0
2	2	3	0
3	0.5	3	3
4	2	1	1
5	" x "	1	1

Record any issues and find a way to fix them. Test more if necessary!

Programme Steps 6 Cont.

- 1 Fine.
- 2 Returns The first solution is 0.0 and The second solution -6.0, the correct solution is $x_1 = 0$ and $x_2 = 1.5$.
Fix: in lines 8 and 9 of the code we need to have `2*a` in bracket, i.e., `(2*a)`.
- 3 Fine.
- 4 Returns error, here the solutions are imaginary, but `math.sqrt` cannot take the square root of an imaginary number. **Fix:** use the `cmath` in line 7 package together with `cmath.sqrt` in line 8 and 9.
- 5 Returns error - correctly as the input is not a float number.

Example Cont. IV Programme Final Version

Following is the second attempt at the script after testing.

```
1  # Input, you can change these and rerun.
2  a=1;b=1;c=0
3
4  # Here we do the calculations.
5  # For taking square root we need cmath
6  import cmath
7  delta=b**2-4*a*c
8  firstSolution=(-b+cmath.sqrt(delta))/(2*a)
9  secondSolution=(-b-cmath.sqrt(delta))/(2*a)
10
11 # Use print function to display output
12 print("The first solution is", firstSolution, "and the second
    solution is", secondSolution)
```

Question

What if we wanted the programme to request for the input?

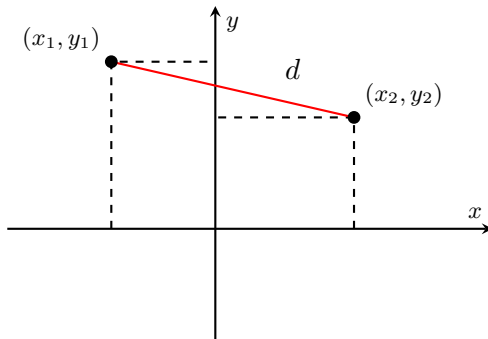
Example Cont. IV: Alternative Script

Programme Using input() Function.

```
1  # Start by input
2  a=float(input("Enter a Non-Zero Real Number for a:"))
3  # Use the input function to ask for a, b, c.
4  b=float(input("Enter a Real Number for b:"))
5  # It is useful to have type float,
6  c=float(input("Enter a Real Number for c:"))
7  # so upon entrance turn the input into float.
8
9  # Here we do the calculations.
10 import cmath # For taking square root
11 delta=b**2-4*a*c
12 firstSolution=(-b+cmath.sqrt(delta))/(2*a)
13 secondSolution=(-b-cmath.sqrt(delta))/(2*a)
14
15 # Finally use print function to display output
16 print("The first solution is", firstSolution, "and the second
    solution is", secondSolution)
```

Exercise 5: Straight-Line Distance

Write a programme to compute the straight-line distance between two points in a plane up to 3 decimal places.



What we did today...

Data Types

`int, float, str, bool, complex, ...`

Variables

Assignment statement and rules

Collection of Data

`tuple(), list(), set(), dict()`

Logical and Comparison

`and, or, not, in, is, <, ==, >=, !=`

First Programme

State, input, solve, output, script, test

Next Time

Conditional Statements and Loops

Week 8

Conditional Statements and Loops

```
a=int(input("Enter an integer"))
while a<10:
    for i in range(1,10,a):
        if i%3==0:
            print(i,"is divisible by 3")
        elif i%5!=0 and i%2!=0:
            print(i, "is not divisible by 5 and 2")
        else:
            print(i, "is not divisible by 3, but can be
divisible by other primes")
    a=a+1
else:
    print("End of a complicated and very strange while, for,
if statement!")
```

Lecture Contents

- 1 **Introduction**
 - Module Aims and Assessment
 - Topics to be Covered
 - Reading List and References
- 2 **Week 5: Data and Visualisation with Excel**
 - What is Excel?
 - Data Entry and Functions
 - Visualisation Methods
 - Tables and Pivots
- 3 **Week 6: Introduction to Python 3**
 - What is Python?
 - Installing and Running
 - Basic Programming and Mathematics
 - Interfaces for Python: IDLE, Jupyter, Spyder
 - Packages: math, cmath, numpy
- 4 **Week 7: Data Types, Methods, and Programming**
 - Code and Data
 - Data Types: int, str, bool, float, complex
 - Variables and Assignments
 - Collection of Data: tuple, list, set, dict
 - Logical and Comparison Operations
 - First Programme
- 5 **Week 8: Conditional Statements and Loops**

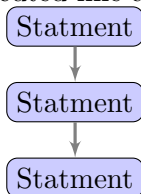
- if, else, and elif Statements
- for and while Loops
- break and continue Statements

- 6 **Week 9: Built-in and User-Defined Functions**
 - Functions in Mathematics
 - Built-in Functions
 - User-Defined Functions
 - Python Anonymous Functions lambda
- 7 **Week 10: Matrices, Dataframes, and Data Manipulation**
 - Matrices with numpy and sympy
 - Linear Algebra, Symbolic Mathematics, Calculus, and Number Theory with sympy
 - Data Manipulation and Visualisation with pandas
 - Import Data pandas.read_excel()
- 8 **Week 11: Statistics and Visualisation Methods**
 - Statistics with scipy
 - Plotting with matplotlib
 - Interactive Plots with plotly
- 9 **Week 12: Numerical Algorithms**
 - Introduction to Numerical Analysis
 - Roots of Nonlinear Equations
 - The Bisection Method
 - Error for Bisection Method

By the end of this session you will be able to...

- 1 Learn to use conditional statements and loops.
- 2 Construct control flows using `if`, `else`, and `elif` statements.
- 3 Create loops using `for` and `while`.
- 4 Understand how to use `break` and `continue` statements.

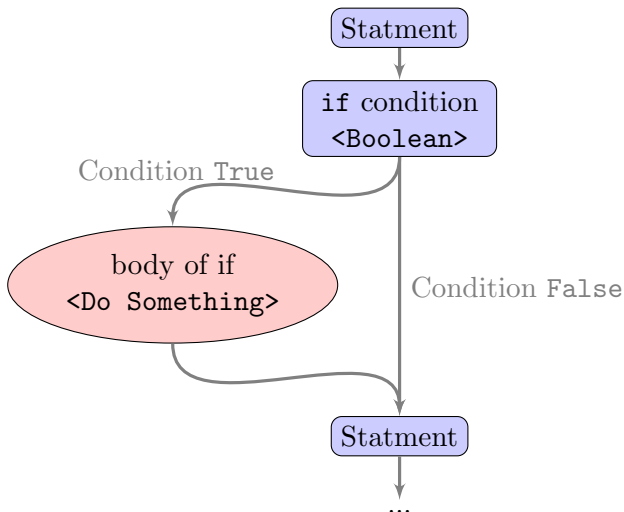
- Last week we create programmes where individual commands would be executed line after line.



- This week we will be looking at diverting control, e.g., *"If today is Monday, then I will go to work"*, or *"while I am hungry, I will eat"*.
- There are two main ways of doing this:
 - **Conditional** statements with `if`, `else`, and `elif` using `bool` data types, e.g., **Comparison** and **Logical Operators**.
 - **Loops** with `for` and `while` using `range`, `list`, and `bool`.
- **Flow charts**, representation of all the possible paths through a process, are used to visualise conditional statements and loops.

if statement

The `if` statement is used to divert a flow depending on a `bool` type data, i.e., depending on `True` or `False` perform different actions.



Python Simple if Statement

The general expression for if statement is as follows.

```
if <Boolean expression>:    # True/False
    # condition followed by colon
    <indented block of code>
    # Any number of indented lines
```

```
# Example 1
Cond=True # Statment
if Cond: # True condition
    print("Condition was", Cond)
# Example 2
x=float(input("Enter a number")) # Statment
if x>=0: # True/False condition
    print(x, "is a non-negative number")
    print(x, "plus 10 is equal to", x+10)
```

Exercise 1:

- 1 Create variables which keeps the score for each of you and your friend in a game. Write a code which prints

I Win

if your score is higher than your friend's.

- 2 Write a code takes a number x and if it is non-zero it prints

A non-zero number

also returns $1/x$.

- 3 Write a code takes a number x check if it is odd, prints

Odd number

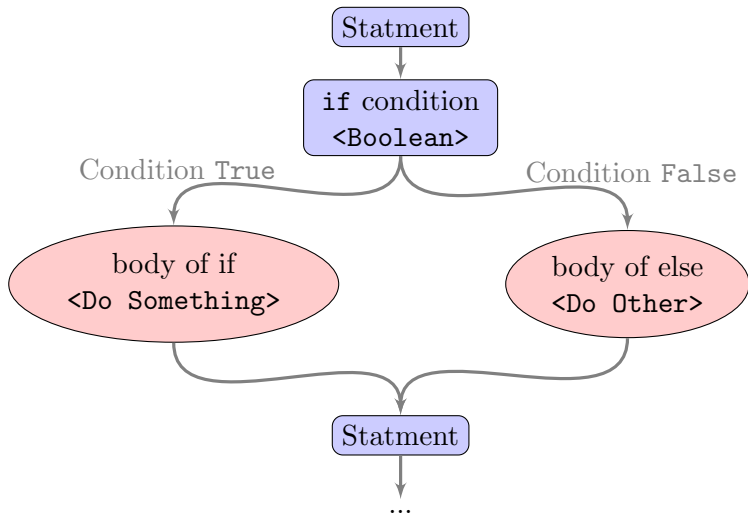
for odd numbers checks if x^3 is greater than 100 prints

A large number

and returns $\ln(x)$ for these. **Hint:** you may need to use if statement within another if statement, nested statements.

if and else Statements

In many occasions you may need to perform two operations depending on a condition being true or false.



Python if and else Statements

The general expression for if and else statement is as follows.

```
if <Boolean expression>:    # True/False
    # condition followed by colon
    <indented block of code>
    # some number of indented lines
else: # else followed by colon
    <indented block of code>
    # other number of indented lines
```

```
# Example
Cond=input("Choose True or False") # Statment
if Cond: # True condition
    print("Condition you chose was", Cond)
else:
    print("Condition you chose was", Cond)
```

if and elif Statements

Sometimes we have more than two operations to perform. In this case we use `elif` statements.

```
if <Boolean expression>:    # True/False
    # condition followed by colon
    <indented block of code>
    # some number of indented lines
elif <Boolean expression>:
    # else if followed by colon
    <indented block of code>
    # other number of indented lines
elif <Boolean expression>:
    # more else if followed by colon
    <indented block of code>
    # more number of indented lines
# as many elif as you may need....
else:
    <indented block of code>
    # default number of indented lines
```

Example if and elif Statements

A code which classifies your degree type according to the final mark.

```
mark=float(input("Enter final mark"))
if mark>=70:
    print("First Class")
elif mark>=60:
    print("2.1")
    print("Mark is", 70-mark, "away from a first")
elif mark>=50:
    print("2.2")
elif mark>=40:
    print("Pass")
else:
    print("May need to resit some modules.")
```

Exercise 2:

- 1 Write a code which asks for the age of the universe and if the correct answer is provided prints

Correct, Well done!

and if the wrong answer is provided prints

Unfortunately that is incorrect!

- 2 Write a code which asks for temperature and rainy/sunny and prints the following.

If temperature is greater than 20 and sunny

Nice and warm day.

If temperature is greater than 10 and sunny

Sunny but chilly.

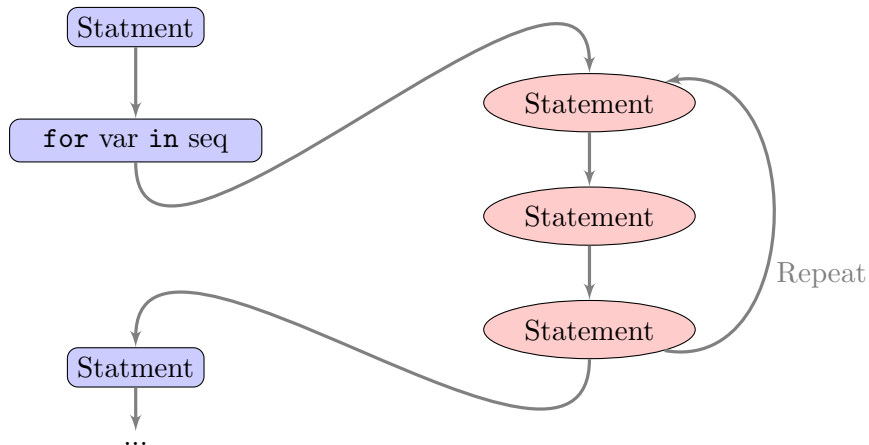
If temperature is greater than 20 and rainy

warm but rainy.

Otherwise Good luck with the weather today!

for Loop

A **for** loop changes the flow to repeat a statement for each item in a given sequence.



Python for Statements

The general expression for for statement is as follows.

```
for var in collectiontype:    # for with
    # range/tuple/list/set followed by colon
    <indented block of code>    # some number of
    # indented lines to be
    # preformed for each item
```

```
# Example 1
```

```
for i in range(2,20,3):
    print(i)
    print(i, "squared is", i*i)
```

```
# Example 2
```

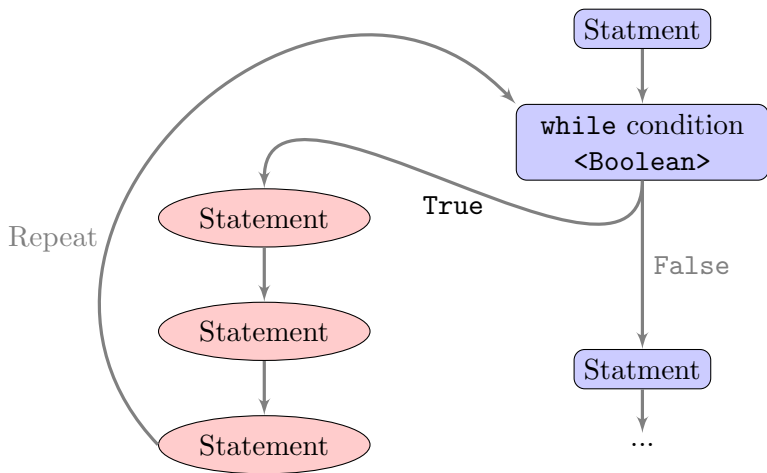
```
for i in "Kayvan":
    print(i)
```

```
# Example 3
```

```
for i in ("1", "sunny", "day", "2.16"):
    print(len(i))
```

while Loop

A **while** loop changes the flow to repeat a statement while a condition is **True**. You can also use **else** with a **while** loop.



Python while Statements

The general expression for while statement is as follows.

```
while <Boolean expression>:    # True/False
    # condition followed by colon
    <indented block of code> # some number of
    # indented lines to be preformed until
    # Boolean condition is false
```

```
# Example 1
```

```
a=float(input("Enter a number"))
```

```
while a<=10:
```

```
    print(a); a+=1
```

```
# Example 2
```

```
a=input("Enter the lecturer's name for MATH1166")
```

```
while a!="Kayvan":
```

```
    a=input("Enter another guess")
```

```
else:
```

```
    print("You have guessed correctly!")
```

Exercise 3:

- 1 Write a **for** loop to print the type of each element in
`T=(-17.5, "kilo", "ram", 5, ["echo", 7])`
- 2 Write a **for** loop to double the elements which are divisible by 5 and positive in

`L=[0, 5, 1, 3, 4, 8, 7, 13, -15, 12, 10]`

- 3 Write a **while** loop which computes the product of first 50 integers, i.e.,

$$1 \times 2 \times \cdots \times 50.$$

- 4 Write a **for** loop instead of **while** loop for above.
- 5 Write a code which prints all strings and their length in

`L=[-17.5, "kilo", "ram", 5, ("echo", 7)]`

break Statements

The statement **break** can be used to immediately transfer to the first statement past the last line of the loop. The code

```
while True:
    <statement(s)>
```

runs for ever! It can be stopped using **break**. For example,

```
while True: # loop forever
    line = input("Type anything, type 'done' to exit: ")
    )
    if line == 'done':
        break # transfers control out of the loop
    print("You entered:", line)
print("Finished")
```

The `continue` statement continues with the next iteration of the loop.

```
for num in range(2, 10):  
    if num % 2 == 0:  
        print("Found an even number", num)  
        continue # Go back to the loop  
    print("Found a number", num)
```

which can be used to reduce unnecessary printing.

What we did today...

Flows Conditional Statements

if, else, elif

Loops

for, while, else

Other statments

break, continue

Next Time

Built-in and user-defined functions

Week 9

Built-in and User-Defined Functions

Built-in Functions

abs()	delattr()	hash()	memoryview()	set()
all()	dict()	help()	min()	setattr()
any()	dir()	hex()	next()	slice()
id()	divmod()	ascii()	object()	sorted()
bin()	enumerate()	input()	oct()	staticmethod()
int()	eval()	bool()	open()	tuple()
ord()	exec()	isinstance()	breakpoint()	range()
pow()	filter()	issubclass()	bytearray()	super()
str()	float()	bytes()	print()	iter()
len()	format()	callable()	property()	compile()
chr()	frozenset()	list()	classmethod()	vars()
sum()	getattr()	locals()	repr()	zip()
type()	globals()	map()	reversed()	round()
max()	hasattr()	complex()	__import__()	

1

Introduction

- Module Aims and Assessment
- Topics to be Covered
- Reading List and References

2

Week 5: Data and Visualisation with Excel

- What is Excel?
- Data Entry and Functions
- Visualisation Methods
- Tables and Pivots

3

Week 6: Introduction to Python 3

- What is Python?
- Installing and Running
- Basic Programming and Mathematics
- Interfaces for Python: IDLE, Jupyter, Spyder
- Packages: math, cmath, numpy

4

Week 7: Data Types, Methods, and Programming

- Code and Data
- Data Types: int, str, bool, float, complex
- Variables and Assignments
- Collection of Data: tuple, list, set, dict
- Logical and Comparison Operations
- First Programme

5

Week 8: Conditional Statements and Loops

- if, else, and elif Statements
- for and while Loops
- break and continue Statements

6

Week 9: Built-in and User-Defined Functions

- Functions in Mathematics
- Built-in Functions
- User-Defined Functions
- Python Anonymous Functions `lambda`

7

Week 10: Matrices, Dataframes, and Data Manipulation

- Matrices with numpy and sympy
- Linear Algebra, Symbolic Mathematics, Calculus, and Number Theory with sympy
- Data Manipulation and Visualisation with pandas
- Import Data `pandas.read_excel()`

8

Week 11: Statistics and Visualisation Methods

- Statistics with scipy
- Plotting with matplotlib
- Interactive Plots with plotly

9

Week 12: Numerical Algorithms

- Introduction to Numerical Analysis
- Roots of Nonlinear Equations
- The Bisection Method
- Error for Bisection Method

By the end of this session you will be able to...

- 1 Learn about functions and how they operate in Python.
- 2 Use the built-in functions.
- 3 Create user-defined functions.

Much of mathematics is concerned with the study of functions, this is due to the fact that they have some much applications is real life!

Question

What is a function?

Definition (Function)

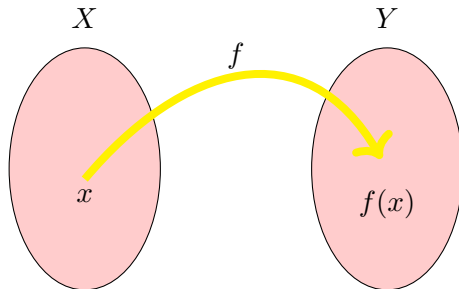
Let X and Y be sets. A **function** from X to Y is a rule that **assigns** to each $x \in X$ a single element of T , denoted by $f(x)$. We write

$$f : X \longrightarrow Y$$

to mean that f is a function from X to Y . If $f(x) = y$, we often say f sends $x \mapsto y$.

Functions: Domain, Range, Image

The set X is known as the **domain** and Y the **range**.



The elements of Y which can be reached by applying f to elements of X for a set called the **image** of f . That is

$$\text{Im } f = f(X) = \{f(x) \mid x \in X\}.$$

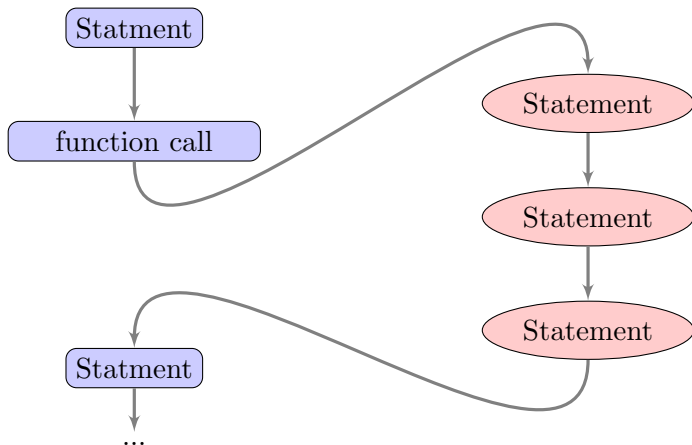
For example,

$$f : \mathbb{R} \longrightarrow \mathbb{R}$$

$$x \mapsto x^2.$$

Functions in Programming

A function **diverts** the follow once called.



Built-in Functions

- We have seen the built-in **operators** in Python

$+$, $-$, $*$, $/$, $**$, $//$, $\%$, $<$, $>$, \dots

- Similarly Python comes with a number of built-in **functions**.
- Using a function is known as **calling a function**. Here's what a generic call to a function with arguments looks like:

```
<functionName>(<argument1>, <argument2>, ...)
```

- A function can have zero or any **number** of **input** variables as well as **default** values.
- We have worked with many other functions already `print()`, `input()`, `type()`, `bool()`, `int()`, `string()`, `float()`, `tuple()`, `dict()`, `pow()`, `complex()`, `frozenset()`, ...
- If you need a function which does not exist in Python already, you can define your own function.

Arguments in Built-in Functions

- Some functions accept a number of propositional arguments.
- We have already used `print()`, whose input can be a number of variables.

```
print("Hello, World", "Today I am", 22, sep=" - ", end="
    ")
```

- The `print()` function has three key word arguments: `sep`, `end`, and `file`.
 - The `sep` parameter's **default** is a space; if two or more propositional arguments are given, each is printed with the `sep` in between.
 - The `end` parameter's **default** is `\n`, which is why a newline is printed at the end of calls to `print()`.
 - The `file` parameter's default is `sys.stdout`, the standard output stream, which is usually the console.

User-Defined Functions

You can define a function using the general expression.

```
def functionName(parameter1, parameter2,...):  
    # def with function name, a number of  
    # inputs followed by colon  
    <indented block of code>    # some number of  
    # indented lines to be  
    # preformed on the inputs
```

Example 1

```
def hello(): # name of the function and no input  
    print("Hello, World!") # Prints "Hello, World"
```

Example 2

```
def powerAddFun(x,y=2): # has two input x and y  
    # Default value for y is 2  
    return pow(x,y), x+y # Return a tuple output
```

Call by `hello()`, `powerAddFun(2,3)`, or `powerAddFun(2)`.

Functional Compositions

- You can **compose** functions, i.e., apply one after the other.

```
age=20 # Composition of  
print(type(age)) # Compose print() and type()
```

- You can write functions that apply to **collection** types.

```
def enumerate(sequence, start=0): # Number  
    n = start # elements in a collection type  
    for elem in sequence:  
        yield n, elem  
        n += 1
```

- You can use **conditional** statements and loops in functions.

```
def all(iterable): # Decide if elements  
    for element in iterable: # in a collection  
        if not element: # type are True  
            return False  
    return True
```

Exercise 1: Functions

- 1 Write a function that returns

$$x^2 + x - 2$$

for a given x . Apply your function to `range(1,10)`.

- 2 Write a function which has inputs `height`, in meters, `weight` in kilo, and returns BMI. Recall

$$\text{BMI} = \frac{\text{weight}}{\text{height}^2}.$$

- 3 Write a function which has no input, but once called asks for a name and prints

```
....(name) will be a great mathematician!
```

- 4 Write a function that calculates the mean of all numbers in a collection type.

Python Anonymous Functions `lambda`

- A `lambda` function is a small anonymous function.
- It can have any number of inputs, but only one expression

```
functionName = lambda arguments : expression
```

- For example,

```
powerFunLambda = lambda x,y : pow(x,y)
```

- The power of `lambda` is better shown when you use them as an anonymous function inside another function.

```
def myFunc(n): # for each n
    return lambda a : a * n # create a lambda
myDoubler = myFunc(2) # returns a --> 2*a
myDoubler(11) # returns 2*11
```

What we did today...

Functions

Assignments between two sets

Built-in Functions

`print()`, `input()`, `type()`, ...

User-Defined Functions

`def` followed by name, ...

Python Lambda

`lambda arguments : expression`

Next Time

Matrices and Data Manipulation

Week 10

Matrices and Data Manipulation

```
from sympy import * # import sympy
```

```
M = Matrix([[1, 0, 1, 3],  
            [2, 3, 4, 7],  
            [-1, -3, -3, -4]])
```

```
M_rref = M.rref() # Use sympy.rref() method
```

```
print("The Row echelon form of matrix M and the pivot  
      columns : {}".format(M_rref))
```

The Row Reduced Echelon Form of matrix M and the pivot
columns: (Matrix([
 [1, 0, 1, 3],
 [0, 1, 2/3, 1/3],
 [0, 0, 0, 0]]), (0, 1))

1

Introduction

- Module Aims and Assessment
- Topics to be Covered
- Reading List and References

2

Week 5: Data and Visualisation with Excel

- What is Excel?
- Data Entry and Functions
- Visualisation Methods
- Tables and Pivots

3

Week 6: Introduction to Python 3

- What is Python?
- Installing and Running
- Basic Programming and Mathematics
- Interfaces for Python: IDLE, Jupyter, Spyder
- Packages: math, cmath, numpy

4

Week 7: Data Types, Methods, and Programming

- Code and Data
- Data Types: int, str, bool, float, complex
- Variables and Assignments
- Collection of Data: tuple, list, set, dict
- Logical and Comparison Operations
- First Programme

5

Week 8: Conditional Statements and Loops

- if, else, and elif Statements
- for and while Loops
- break and continue Statements

6

Week 9: Built-in and User-Defined Functions

- Functions in Mathematics
- Built-in Functions
- User-Defined Functions
- Python Anonymous Functions lambda

7

Week 10: Matrices, Dataframes, and Data Manipulation

- Matrices with numpy and sympy
- Linear Algebra, Symbolic Mathematics, Calculus, and Number Theory with sympy
- Data Manipulation and Visualisation with pandas
- Import Data `pandas.read_excel()`

8

Week 11: Statistics and Visualisation Methods

- Statistics with scipy
- Plotting with matplotlib
- Interactive Plots with plotly

9

Week 12: Numerical Algorithms

- Introduction to Numerical Analysis
- Roots of Nonlinear Equations
- The Bisection Method
- Error for Bisection Method

By the end of this session you will be able to...

- 1 Know how to create vectors and matrices with **numpy** and **sympy**.
- 2 Perform linear algebra operations, symbolic manipulations, and number theoretic calculations.
- 3 Create/manipulate dataframes and load data with **pandas**.
- 4 Produce summary tables and visualisation from data.

- **So far** we have learnt all basics of programming:
 - Basic calculations and operations.
 - Data types, manipulation of collection types.
 - Conditional statements and Loops.
 - Use defined and built-in functions.
 - Creating programmes to solve problems.
- We can now look at **advanced usage** of Python for
 - linear algebra and number theory,
 - statistics and data analysis,
 - plotting/visualisation, and
 - sophisticated mathematical problems.
- Libraries **numpy** and **sympy** are used for creating and manipulating **matrices** as well as **symbolic** mathematics.
- The library **pandas** is used for loading and **analysis** of data into Python.

Package with numpy and sympy

- For calculation that are repeated for a set of input values, it is useful to store data as **arrays** and computation in terms of array: **vectorisation**.
- The package **NumPy** provides efficient functions for manipulating and processing arrays.
- NumPy **arrays** bear some resemblance to Python's **list** data structure.

```
import numpy # You can use: import numpy as np
V=[1,2,3,4] # list
Vn=numpy.array([1,2,3,4]) # A numpy array
```

- However, NumPy arrays are homogeneous with fixed size.
- **homogeneous**: all elements in the array have the same data type. **Fixed size** means that an array cannot be resized.
- You can preform **linear algebra** operations with NumPy arrays.

Arrays with Library numpy

Easiest way to create arrays, i.e., vectors and matrices is through lists within a list.

- The code

```
V=numpy.array([1,2,3,4]) # A row numpy array
```

creates a row vector say

$$V = (1 \quad 2 \quad 3 \quad 4).$$

- However, the code

```
V=numpy.array([[1],  
               [2],  
               [3],  
               [4]]) # A column numpy array
```

create the column vector $V = \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \end{pmatrix}.$

Matrices

To create a matrix use lists with a list. The code

```
A=numpy.array([[1, 2, 3],  
               [0, 1, 0],  
               [3, 7, 13],  
               [0.1, 0, 0.1]])
```

creates the matrix

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 0 & 1 & 0 \\ 3 & 7 & 13 \\ 0.1 & 0 & 0.1 \end{pmatrix}.$$

Tensors and Basic Attributes of NumPy Arrays

Create **multidimensional** arrays, tensors, with types

```
At=numpy.array([[[1,2],[2,3]],  
                [[0,1],[1,0]],  
                [[3,7],[13,17]]], dtype=complex)
```

The functions used to find the basic attributes of arrays

```
type(At), At.shape, At.size, At.ndim, At.dtype  
(numpy.ndarray, (3, 2, 2), 12, 3, dtype('complex128'))
```

Change data types use `astype()`; for example,

```
A.astype(numpy.int)  
array([[ 1,  2,  3],  
       [ 0,  1,  0],  
       [ 3,  7, 13],  
       [ 0,  0,  0]])
```

Creating Arrays

There are several functions, alongside `numpy.array()`, which allow you to create arrays according to patterns.

Function	Definition
<code>numpy.zeros((m,n))</code>	m rows and n columns of zeros
<code>numpy.ones((m,n))</code>	m rows and n columns of ones
<code>numpy.arange(n)</code>	array containing $0, \dots, n - 1$
<code>numpy.random.rand(n)</code>	array containing n random numbers
<code>numpy.linspace(a,b,s)</code>	from a to b in s steps
<code>numpy.eye(n,k=r)</code>	$n \times n$ identity matrix shifted by r
<code>numpy.diag(L)</code>	zero matrix with L on the diagonal
<code>numpy.meshgrid(L,S)</code>	mesh of arrays L and S

You can apply numpy functions to arrays, e.g.,

```
numpy.log(numpy.linspace(1,10,10)).
```

Some functions: `numpy.cos`, `numpy.sin`, `numpy.tan`,
`numpy.arccos`, `numpy.arcsin`, `numpy.cosh`, `numpy.sinh`,
`numpy.tanh`, `numpy.sqrt`, `numpy.exp`, `numpy.log`.

Indexing and slicing are done using similar rules for lists. For example, for vectors

```
a[n:m:p] # Take elements from n to m in steps of p
```

For matrices we can separate row and column by commas

```
A[m,n] # Take element in row m column n
```

```
A[1,:] # Take second row
```

```
A[:,1] # Take second column
```

```
A[:2, :2] # upper half diagonal block matrix
```

```
A[::2, ::2] # every second element starting from  
0,0
```

Make comparison and subset accordingly

```
a > 5 # boolean comparison for elements > 5
```

```
a[a > 5] # Take elements > 5
```

```
A[A != 0] # Take nonzero elements
```

Reshape vectors: `reshape(n,m)`, `flatten()`, `transpose()`,
`numpy.hstack()`, `numpy.vstack()`, `numpy.append(a,x)`,

- The standard **arithmetic operations** with arrays perform elementwise.

```
x = numpy.array([[1, 2], [3, 4]])
y = numpy.array([[5, 6], [7, 8]])
x*y
array([[ 5, 12],
       [21, 32]])
```

- Comparisons are performed componentwise

```
x<y
array([[ True,  True],
       [ True,  True]])
```

- There are also several aggregate functions: `numpy.mean`, `numpy.std`, `numpy.var`, `numpy.sum`, `numpy.prod`, `numpy.cumsum`, `numpy.cumprod`, `numpy.min`, `numpy.max`, `numpy.argmin`, `numpy.argmax`, `numpy.all`, `numpy.any`.

There are several functions for **linear algebra** operations.

Function	Definition
$a+b$, $a-b$, $k*a$	addition/subtraction, scalar
<code>A.transpose()</code>	transpose of A
<code>numpy.dot(a,b)</code>	dot product and multiplication
<code>numpy.cross(a,b)</code>	cross product for 2/3-D
<code>numpy.matmul(A,B)</code>	multiplication
<code>numpy.linalg.matrix_power(B,n)</code>	raise matrix B a power n
<code>numpy.linalg.det(A)</code>	determinant of A
<code>numpy.linalg.inv(A)</code>	inverse of A
<code>numpy.linalg.solve(A,b)</code>	solve $Ax = b$
<code>numpy.linalg.eigvals(A)</code>	eigenvalues of A
<code>numpy.linalg.eig(A)</code>	eigenvalues/vectors of A
<code>numpy.linalg.norm(A)</code>	matrix or vector norm

Functions offered by NumPy and SymPy (see next slide) can be used to make many computations for concepts you learn in **MATH1167 Techniques of Calculus and Linear Algebra.**

- You can use the **symbolic** mathematics library **sympy** for some matrix manipulations.
- The code

```
M=sympy.Matrix([[1, 0, 1],  
                [2, 3, 4],  
                [-1, -3, -3]])
```

creates the matrix

$$M = \begin{pmatrix} 1 & 0 & 1 \\ 2 & 3 & 4 \\ -1 & -3 & -3 \end{pmatrix}.$$

- Some functions on matrices are `M.shape`, `M.row(m)`, `M.col(n)`, `M+N`, `M*N`, `M**-1`, `M.det()`, `M.rref()`, `M.eigenvals()`, `M.eigenvects()`, `M.diagonalise()`, `M.nullspace()`.

Exercise 1: Vectors and Matrices

- ❶ Create the following vectors and matrices, using NumPy,

$$A = \begin{pmatrix} 1 & 0 & 1 \\ 2 & 3 & 4 \end{pmatrix}, \quad u = 5i + 12j, \quad v = i + 2j - 2k,$$

where i, j, k are the standard unit vectors.

- ❷ Extract the matrix

$$\begin{pmatrix} 1 & 1 \\ 2 & 4 \end{pmatrix}$$

from A .

- ❸ Extract the third column of A .

- ❹ Find the following.

$$u + v, \quad u \cdot v, \quad u \times v, \quad |u|, \quad \widehat{v}, \\ A^T, \quad Au, \quad AA^T, \quad \det A^T A.$$

- ❺ Find the eigenvalues and eigenvectors of AA^T .
❻ Find the reduced echelon form of A .

- You can define symbols using function of `sympy` and perform many mathematical operations.

```
# declare symbols
a, b, c, d = sympy.symbols("a, b, c, d")
x, y, z, t = sympy.symbols("x, y, z, t")
M = sympy.Matrix([[a, b], # define matrices
                  [c, d]]) # in terms of
symbols
N = sympy.Matrix([[x, y],
                  [z, t]])
M*N # * is matrix multiplication in sympy
```

- Have symbolic versions of quantities and expressions

```
sympy.pi
sympy.sqrt(27)
expr=sympy.exp((x+y)**2)
```

- Evaluate using `evalf()`, e.g, `sympy.sqrt(27).evalf(5)`.

Symbols and Calculus sympy Functions

Function	Definition
<code>sympy.symbol("x,y,z")</code>	define symbols
<code>sympy.exp()</code> , <code>sympy.sqrt()</code>	usual functions
<code>sympy.Lambda(x,x**2)</code>	lambda functions
<code>sympy.expand(expr)</code> , <code>sympy.factor(expr)</code>	expand/factor expr
<code>expr.subs({x:a,y:b})</code>	substitute $x = a$ and $y = b$
<code>sympy.Eq(x+y,4)</code>	define equation $x + y = 4$
<code>sympy.simplify(expr)</code>	simplify expr
<code>sympy.diff(sympy.exp(x**2),x,2)</code>	differentiate e^{x^2} twice
<code>sympy.Derivative(sympy.exp((x+y)**2),x,y)</code>	derivative $\frac{\partial^2}{\partial x \partial y} e^{(x+y)^2}$
<code>sympy.integrate(x**3,(x,a,b))</code>	integral $\int_a^b x^3 dx$
<code>sympy.Integral(x**3,x)</code>	integral $\int x^3 dx$
<code>sympy.Integral(x**3,x).doit()</code>	compute $\int x^3 dx$
<code>sympy.limit(sympy.sin(x)/x,x,0)</code>	$\lim_{x \rightarrow 0} \frac{\sin x}{x}$
<code>sympy.solve(sympy.Eq(x**2,-1),x)</code>	solve $x^2 + 1 = 0$
<code>sympy.Sum(1/x**2,(x,1,sympy.oo)).doit()</code>	evaluate $\sum_{x=1}^{\infty} \frac{1}{x^2}$

Number Theory sympy Functions

The library SymPy also offers many **number theoretic** functions.

Function	Definition
<code>sympy.isprime(n)</code>	True if n is prime
<code>sympy.primerange(m, n)</code>	primes p between $m \leq p < n - 1$
<code>sympy.randprime(m, n)</code>	random prime p between $m \leq p < n - 1$
<code>sympy.primepi(n)</code>	Number of primes $p \leq n$
<code>sympy.prime(n)</code>	the n^{th} prime
<code>sympy.primefactors(n)</code>	prime factors of n
<code>sympy.divisor_count(n)</code>	number of divisors of n
<code>sympy.factorint(n)</code>	factorisation of n

These can be used in computations for concepts you learn in **MATH1172 Vector Calculus and Number Theory** during your second year.

Data Manipulation with pandas

- In many case we have **dataframes** or **.csv** files which needs to be analysed as arrays in Python.
- Easy-to-use data structures and data analysis tools **pandas** library is used for dealing with such cases.
- A **dataframe** is a two-dimensional array with labelled axes. You can create dataframe using **pandas**.

```
df = pandas.DataFrame({  
    'A': numpy.linspace(0,2,8)**2,  
    'B': pandas.date_range('20130101', periods=8),  
    'C': pandas.Series(range(8), dtype='float32'),  
    'D': numpy.array([1,3] * 4, dtype='int32'),  
    'E': pandas.Categorical(["train", "train", "test",  
        "train"]*2),  
    'F': ['foo', 'goo']*4,  
    'G': numpy.random.randn(8)}, index=list(range(8)))  
df.name= "Mydata"
```

is a dataframe with 8 rows and 7 columns.

- The code produces the following dataframe.

A	B	C	D	E	F	G
0.000000	2013-01-01	0.0	1	train	foo	-0.947895
0.081633	2013-01-02	1.0	3	train	goo	0.841866
0.326531	2013-01-03	2.0	1	test	foo	0.310235
0.734694	2013-01-04	3.0	3	train	goo	0.171024
1.306122	2013-01-05	4.0	1	train	foo	-0.759685
2.040816	2013-01-06	5.0	3	train	goo	0.141803
2.938776	2013-01-07	6.0	1	test	foo	0.604985
4.000000	2013-01-08	7.0	3	train	goo	-0.300757

- **Attributes of dataframe** can be found using `df.index`, `df.columns`, `df.dtypes`, `df.describe()`, `df.median()`, `df.mean()`, `df.std()`.
- **Indexing and slicing** can be preformed through, for example, `df["A"]`, `df.A[2:]`, `df.loc[1:5, ["A", "B"]]`, `df[(df.E == "train") & (df.G > 0)]`, `df.sort_index(axis=1, ascending=False)`.

- **Grouping** and **pivoting** can be done through

```
df.groupby(['E', 'F']).sum()  
pandas.pivot_table(df, values='A',  
                    index=['E', 'F'], columns=['D'])
```

- **Summary** plots

```
df.plot(y=["G", "A"], x="B", kind='line', title='  
    Line Plot')  
df.plot(y=["G", "A"], x="B", kind='bar', title='Bar  
    Plot')  
df.plot(y=["G", "A"], x="B", kind='box', title='Box  
    Plot')  
df.plot.scatter(y="A", x="C", c='G', title='  
    Scatter Plot')
```

- **Transpose** data `df.T` or turn into NumPy object `df.to_numpy()`.
- To **import** a .csv file use `pandas.read_excel()`.

What we did today...

NumPy arrays and Matrices

Linear Algebra

SymPy and Symbolic

Data with Pandas

Next Time

`numpy.array()`, functions, and attributes

`numpy.dot()`, `numpy.linalg.det()`

`sympy.symbols("x, y, z")`, calculus, number theory

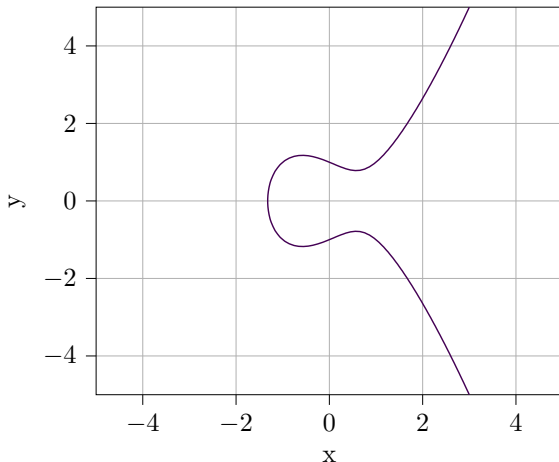
`pandas.DataFrame()`, visualisation, `pandas.read_excel()`

Statistics and Visualisation Methods

Week 11

Statistics and Visualisation Methods

Elliptic Curve $y^2 = x^3 - x + 1$



Lecture Contents

1

Introduction

- Module Aims and Assessment
- Topics to be Covered
- Reading List and References

2

Week 5: Data and Visualisation with Excel

- What is Excel?
- Data Entry and Functions
- Visualisation Methods
- Tables and Pivots

3

Week 6: Introduction to Python 3

- What is Python?
- Installing and Running
- Basic Programming and Mathematics
- Interfaces for Python: IDLE, Jupyter, Spyder
- Packages: math, cmath, numpy

4

Week 7: Data Types, Methods, and Programming

- Code and Data
- Data Types: int, str, bool, float, complex
- Variables and Assignments
- Collection of Data: tuple, list, set, dict
- Logical and Comparison Operations
- First Programme

5

Week 8: Conditional Statements and Loops

- if, else, and elif Statements
- for and while Loops
- break and continue Statements

6

Week 9: Built-in and User-Defined Functions

- Functions in Mathematics
- Built-in Functions
- User-Defined Functions
- Python Anonymous Functions lambda

7

Week 10: Matrices, Dataframes, and Data Manipulation

- Matrices with numpy and sympy
- Linear Algebra, Symbolic Mathematics, Calculus, and Number Theory with sympy
- Data Manipulation and Visualisation with pandas
- Import Data pandas.read_excel()

8

Week 11: Statistics and Visualisation Methods

- Statistics with scipy
- Plotting with matplotlib
- Interactive Plots with plotly

9

Week 12: Numerical Algorithms

- Introduction to Numerical Analysis
- Roots of Nonlinear Equations
- The Bisection Method
- Error for Bisection Method

By the end of this session you will be able to...

- 1 Use the statistical capabilities of Python with `scipy`.
- 2 Make plots and different visualisations with `matplotlib`.
- 3 Create interactive plots with `plotly`.

SciPy Scientific

The library `scipy` is a collection of mathematical algorithms built on NumPy. It offers many features including

- statistical functions,
- integration and differential equation solvers,
- interpolation,
- signal processing, and many others.

Matplotlib Graphics

The library `matplotlib` is 2-D plotting library which produces high quality figures.

Plotly's Interactive

The library `plotly` makes interactive, publication-quality graph and animations.

- The **stats** component of **scipy** allows for implementation of **combinatorial** functions and **random variables**.
- It includes **special functions**:
 - `scipy.special.factorial(n)` for $n!$
 - `scipy.special.comb(n, k)` for

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

- `scipy.special.comb(n, k, repetition=True)` for

$$\frac{(k+n-1)!}{k!(n-1)!}$$

- `scipy.special.perm(n, k)` for

$$\frac{n!}{(n-k)!}$$

- These can be used in computations for concepts you learn in **STAT1040 Probability and Statistical Inference**.

Continuous

- `stats.norm.rvs(loc=mu, scale=sigma, size=k)` generates `k` samples from normal distribution $\mathcal{N}(\mu, \sigma)$.
- Related functions include `stats.norm.pdf`, `stats.norm.cdf`, `stats.norm.ppf`, `stats.norm.moment`, `stats.norm.stats`, `stats.describe`.
- Includes `stats.expon.rvs`, `stats.gamma.rvs`, `stats.chi.rvs`

Discrete

- `stats.binom.rvs(n, p, size=k)` generates `k` samples from normal distribution $\text{Bin}(n, p)$.
- Related functions include `stats.norm.pmf`, `stats.norm.cdf`, `stats.describe`.
- `stats.bernoulli.rvs`, `stats.poisson.rvs`, `stats.geom.rvs`.

Plotting with matplotlib

- Matplotlib is widely used for plotting in Python.
- At the top of the hierarchy is the `matplotlib` "state-machine environment" which is provided by the `matplotlib.pyplot` module (it is often used with NumPy).

```
import matplotlib.pyplot as plt
import numpy as np
```

- At this level, simple functions are used to add plot elements.

```
x = np.linspace(0, 2, 100)
plt.plot(x, x, label='linear')
plt.plot(x, x**2, label='quadratic')
plt.plot(x, x**3, label='cubic')
plt.xlabel('x label')
plt.ylabel('y label')
plt.title("Simple Plot")
plt.legend()
plt.show()
```

Plot Options with matplotlib

You can plot a list, Numpy arrays, or other collections types and customise the appearance.

```
x=[1,3,5,9,9,2,3,4,9,10]; y=np.linspace(0, 3, 10)**2
plt.plot(x, color='blue', marker='*',
         linestyle='solid', linewidth=2, markersize=10,
         label="x vs index", alpha=0.2) # First plot
plt.plot(x, y, color='red', marker='o',
         linestyle='dashed', linewidth=2, markersize=8,
         label="x vs y", alpha=0.8) # Second plot
plt.ylabel("Y"); plt.xlabel("X") # Axes labels
plt.title("Two Plots") # Title for Plot
plt.legend(loc=9) # Show legend locations 1,...,10
plt.annotate('Local Max', xy=(3, 9), xytext=(0, 10),
            arrowprops=dict(facecolor='black', shrink=0.05))
plt.axis([-1, 11, -1, 12]) # Axes limits
plt.grid(True); plt.show() # Show grid and plot
```

More on matplotlib

You can produce scatter, line, box, bar, 3-D, and many other plots. Produce several plots in one figure.

```
years = [1950, 1960, 1970, 1980, 1990, 2000, 2010]
gdp = [300, 543, 1075, 2862, 5979, 10289, 14958]
debt = [100, 1200, 800, 1100, 6000, 15000, 11000]
fig, ((ax1, ax2, ax3, ax4), (ax5, ax6, ax7, ax8)) = plt.
    subplots(2,4, figsize=(15,10))
ax1.plot(years, gdp, color='g', marker='o')
ax2.step(years, gdp, color='r')
ax3.bar(years, gdp, color='b')
ax4.hist(gdp, color='m')
ax5.errorbar(years, gdp, debt, color='k')
ax6.scatter(years, gdp, color='c', marker='*')
ax7.fill_between(years, gdp, debt, color='y')
ax8.boxplot(debt)
plt.tight_layout(pad=0.4, w_pad=0.5, h_pad=1.0)
plt.show()
```


Matplotlib is now **old** and **boring**!

Plotly

- Modern library which allows for interactivity and creation of animations.
- You can use either Plotly Express, for quick graphs, or Plotly Graphics Objects, for advanced usage.

You can create a line, bar, box, and many others using `plotly.express` library.

```
import plotly.express as px
fig = px.line(x=years, y=gdp,
              labels={'x':'Years', 'y':'Billions of Dollars'},
              title='Nominal GDP')
fig.show()
```

Graphics Objects

For more customisation use `plotly.graph_objects` library.

```
import plotly.graph_objects as go
fig = go.Figure(data=go.Scatter(x=years, y=gdp,
    mode='lines+markers'))
fig.update_layout(title='Nominal GDP')
fig.update_xaxes(title_text='Years')
fig.update_yaxes(title_text='Billions of Dollars')
fig.show()
```

```
x = np.linspace(0, 2, 20); fig = go.Figure()
fig.add_trace(go.Scatter(x=x, y=x, mode='markers',
    name='markers')) # Add traces
fig.add_trace(go.Scatter(x=x, y=x**2,
    mode='lines+markers', name='lines+markers'))
fig.add_trace(go.Scatter(x=x, y=x**3, mode='lines',
    name='lines'))
fig.update_layout(title='Simple Plot'); fig.show()
```

What we did today...

SciPy

Statistics and special functions

Basic Plotting

`matplotlib.pyplot`

Advanced Interactivity

`plotly.express`, `plotly.`
`graph_objects`

Next Time

Numerical Methods

Week 12

Numerical Algorithms

```
# Fixed point method to find zeros of  $f(x)=x-\cos(x)$ 
import math
f = lambda x: x-math.cos(x)
g = lambda x: math.cos(x)
x=[0.1]
err=[f(x[0])]
NofIt=50
tolorance=0.0005
for i in range(NofIt):
    x.append(g(x[i]))
    err.append(x[i+1]-x[i])
    if abs(err[i+1]) < tololerance:
        print("Number of Iterations", i+1, "Final x is",
              x[-1], "final evaluation", f(x[-1]), sep="\n ")
        break
```

- 1 **Introduction**
 - Module Aims and Assessment
 - Topics to be Covered
 - Reading List and References
- 2 **Week 5: Data and Visualisation with Excel**
 - What is Excel?
 - Data Entry and Functions
 - Visualisation Methods
 - Tables and Pivots
- 3 **Week 6: Introduction to Python 3**
 - What is Python?
 - Installing and Running
 - Basic Programming and Mathematics
 - Interfaces for Python: IDLE, Jupyter, Spyder
 - Packages: math, cmath, numpy
- 4 **Week 7: Data Types, Methods, and Programming**
 - Code and Data
 - Data Types: int, str, bool, float, complex
 - Variables and Assignments
 - Collection of Data: tuple, list, set, dict
 - Logical and Comparison Operations
 - First Programme
- 5 **Week 8: Conditional Statements and Loops**

- if, else, and elif Statements
 - for and while Loops
 - break and continue Statements
- 6 **Week 9: Built-in and User-Defined Functions**
 - Functions in Mathematics
 - Built-in Functions
 - User-Defined Functions
 - Python Anonymous Functions lambda
 - 7 **Week 10: Matrices, Dataframes, and Data Manipulation**
 - Matrices with numpy and sympy
 - Linear Algebra, Symbolic Mathematics, Calculus, and Number Theory with sympy
 - Data Manipulation and Visualisation with pandas
 - Import Data pandas.read_excel()
 - 8 **Week 11: Statistics and Visualisation Methods**
 - Statistics with scipy
 - Plotting with matplotlib
 - Interactive Plots with plotly
 - 9 **Week 12: Numerical Algorithms**
 - Introduction to Numerical Analysis
 - Roots of Nonlinear Equations
 - The Bisection Method
 - Error for Bisection Method

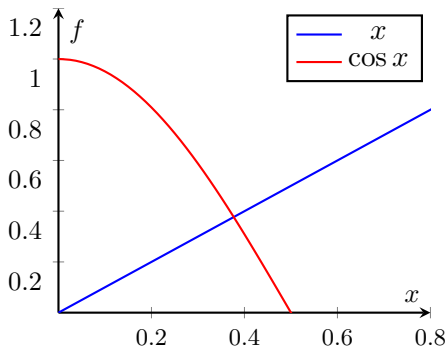
By the end of this session you will be able to...

- 1 Understand the basic concepts of numerical analysis.
- 2 Use Python to implement algorithms solving numerical problems.

Introduction to Numerical Analysis

- **Numerical analysis** is the of algorithms for obtaining **numerical** (or approximations) **solutions** of mathematical problems.
- Consider the zero, in the interval $(0, 1)$, of the function

$$f(x) = x - \cos x = 0.$$



- Cannot **solve** exactly, we need to find an **approximation**.

Solving Mathematical Problems Numerically

- There are many situations similar to the previous slide:
 - Solutions of Equations in One Variable
 - Interpolation and Polynomial Approximation
 - Numerical Differentiation and Integration
 - Initial-Value Problems for Ordinary Differential Equations
 - Iterative Techniques in Matrix Algebra
 - Numerical Solutions of Nonlinear Systems of Equations
 - Boundary-Value Problems for Ordinary Differential Equations
 - Numerical Solutions to Partial Differential Equations
- In all the above cases advanced numerical techniques are applied in order to approximate solutions.
- You will study some of the topics above in **MATH1169 Numerical Mathematics** in your second year.

Roots of Nonlinear Equation

- We are concerned with **estimating** roots of functions, i.e., x such that $f(x) = 0$.
- For example approximating solutions of

$$f(x) = \frac{1}{x} - \tan(x) \text{ or } f(x) = 1 - xe^x.$$

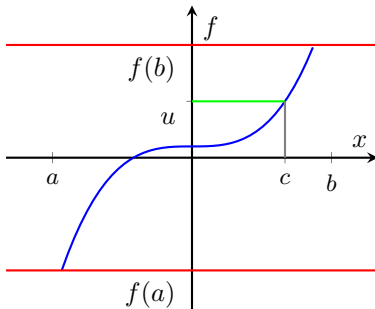
- Problem **dates back** to 1700 B.C.E Babylonian: they approximated the solution to $f(x) = x^2 - 2$ as $x = 1.424222$.
- Using **iterative methods**, given a function $f(x)$, we generate a **sequence** x_n for $n = 1, \dots, \infty$ which converges to a root x^* of $f(x)$ that is

$$\lim_{n \rightarrow \infty} f(x_n) = 0.$$

The Bisection Method

Theorem (Intermediate Value Theorem (Bolzano 1817))

Suppose f is a **continuous** function on an interval $[a, b]$. If u is a number between $f(a)$ and $f(b)$, then there exists $c \in [a, b]$ with $f(c) = u$.



Thus if $f(x)$ is continuous on $[a, b]$ and $f(a)f(b) < 0$, then there exists $x^* \in [a, b]$ with $f(x^*) = 0$.

Bisection Method Algorithm

- Suppose $f(x)$ is a continuous function on $[a, b]$ and $f(a)f(b) < 0$.
- We find a sequence x_n converging to a solution x^* of f using the following procedure.
 - ① Let $a_1 = a$ and $b_1 = b$.
 - ② For $n \geq 1$, calculate $x_n = \frac{a_n + b_n}{2}$. If $f(x_n) = 0$, then stop.
 - ③ If $f(x_n)f(a_n) < 0$, set $a_{n+1} = a_n$ and $b_{n+1} = x_n$.
 - ④ If $f(x_n)f(b_n) < 0$, set $a_{n+1} = x_n$ and $b_{n+1} = b_n$.
 - ⑤ Repeat until $|b_n - a_n|$ is sufficiently small.

Exercise 1 Bisection Method by Hand:

- ① Show that the function

$$f(x) = x - \cos x$$

has a root in the interval $(0, 1)$.

- ② Perform 3 iterations of the bisection method to find an approximation for the root.

The Bisection Method In Python

The following is a way to implement the bisection method.

```
f = lambda x: x-math.cos(x) # Function
a,b=0,1 # Initial values
x=[] # Empty list to keep x
err=[f(a)] # First error
NofIt=50 # Number of iterations
tolorance=0.00005 # Error tolerance
for i in range(NofIt): # Go through iterations
    x.append((a+b)/2)
    if f(x[i])*f(a)<0:
        b=x[i]
    else:
        a=x[i]
    e.append(b-a)
    if abs(e[i+1]) < tolerance:
        print("Number of Iterations", i+1, "Final x is",
              x[-1], "final evaluation", f(x[-1]), sep="\n ")
        break
```

Theorem (Bisection Method)

Suppose f is a continuous function on $[a, b]$ and $f(a)f(b) < 0$. The Bisection method generates a sequence x_n approximating a zero x^ of f with*

$$|x_n - x^*| \leq \frac{b - a}{2^n}, \text{ for } n \geq 1.$$

Other Root Finding Methods

Other root finding algorithm include:

- The Secant Method,
- Fixed Point Iteration,
- Newton-Raphson Method.

The have similar implementations in Python which you will study in your future years!

What we did today...

Numerical Analysis

Introduction, applications

Root Finding Algorithms

Bisection, Fixed Point

Next Time

There won't be any!

Thanks for Your Attention!

Have a good holiday season!

Please Do Not Forget To

- Ask any **questions** now or through my contact details.
- Drop me **comments** and **feedback** relating to any aspects of the course.
- Come and see me during Student Drop-in Hours:
MONDAYS 12:00-13:00 (MATHS ARCADE) AND
FRIDAYS 14:00-15:00 (QM315).
Alternatively, email to make an appointment.

Thank You!