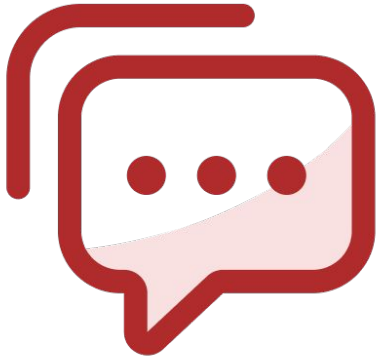
AI in Digital Health Python Programming Course

UoN MedTech



Audience Q&A

Overview of the Course

Weeks 1 & 2

- Focused on learning the basics of Python

Week 3

- Data analysis
- Real world health dataset

Week 4

- Introduction to Machine Learning and Deep learning

Week 5

- Deep Learning

- Computer Vision
- Natural Language Processing

Week 6

- Deep Learning (contd.)

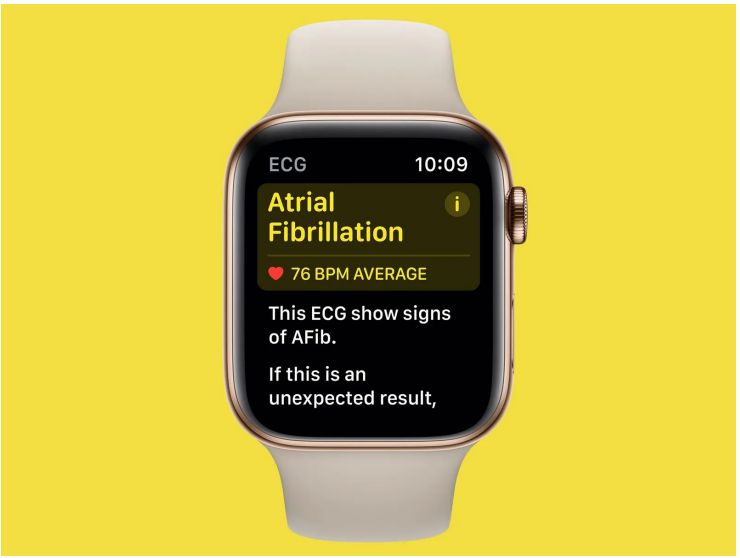
- NLP with Transformers
- GPT
- Reinforcement Learning

Python Course

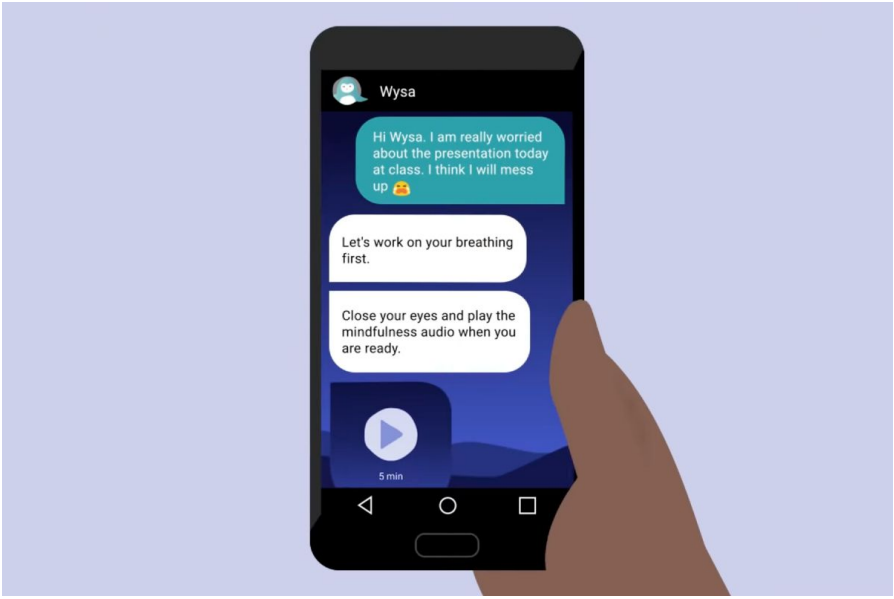
Week 1 – Introduction to Python

Intro. to Digital Health

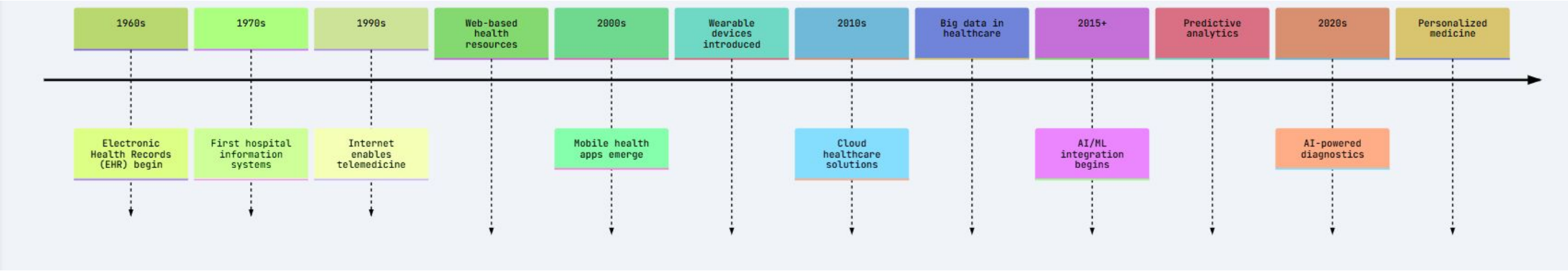
Digital Health

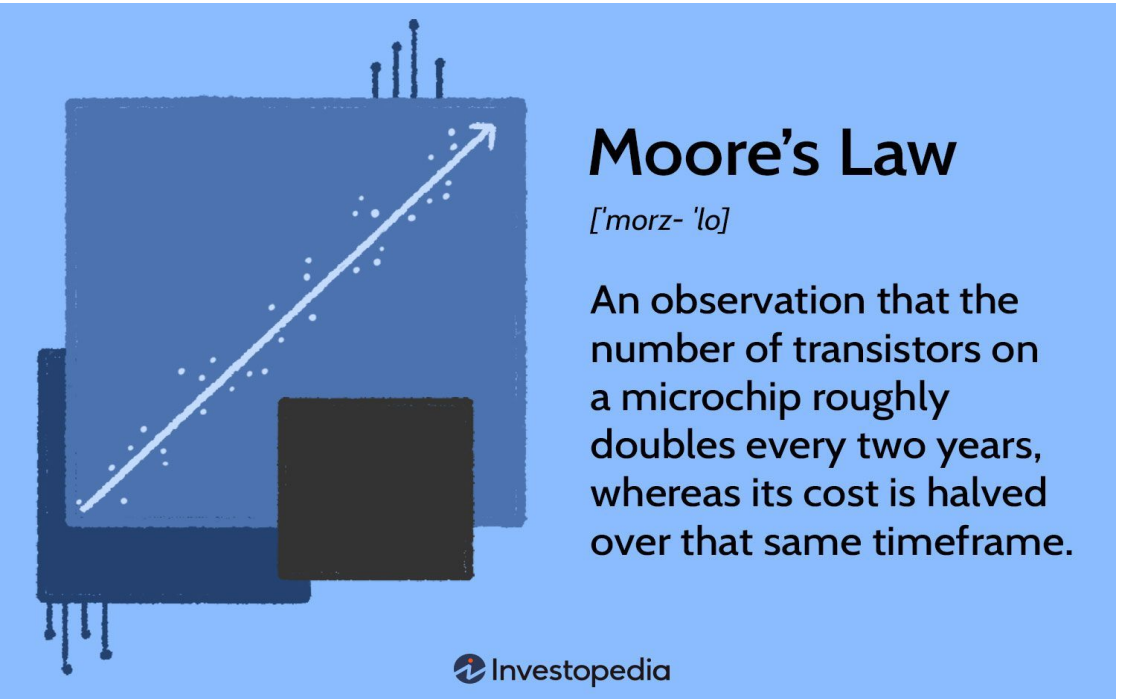


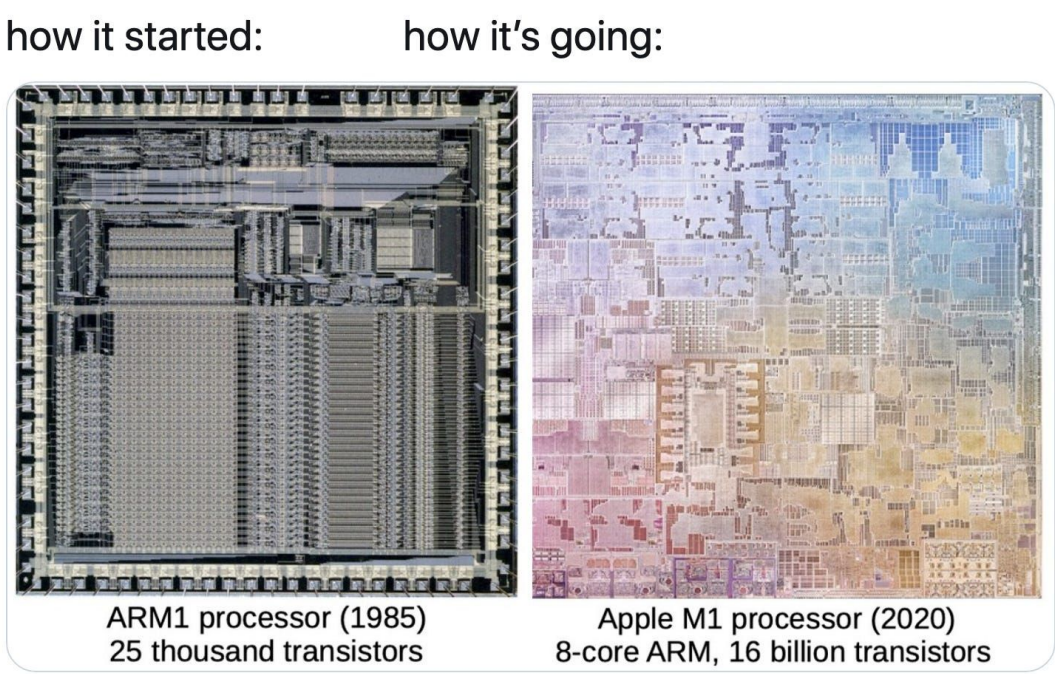
- Patient Monitoring
- Prevention
- Treatment
- Administration
- Research



AI in Digital Health

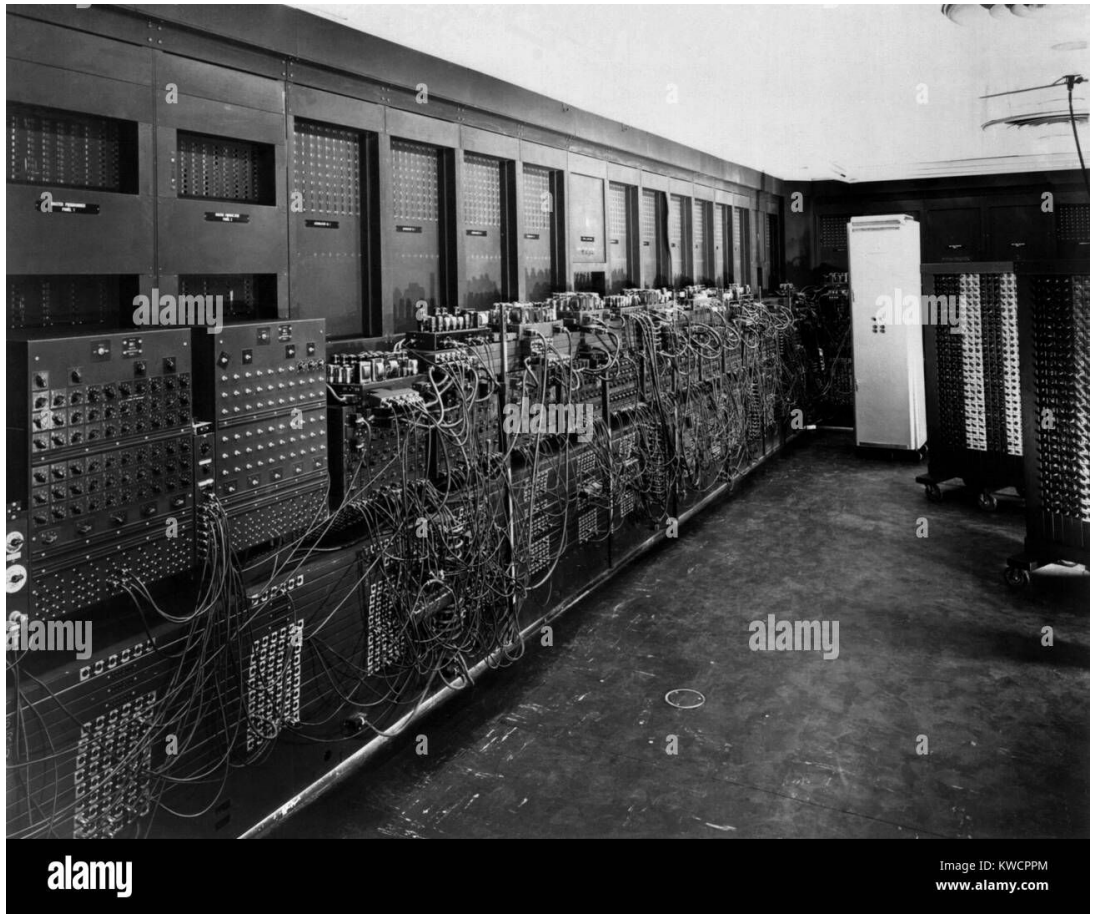






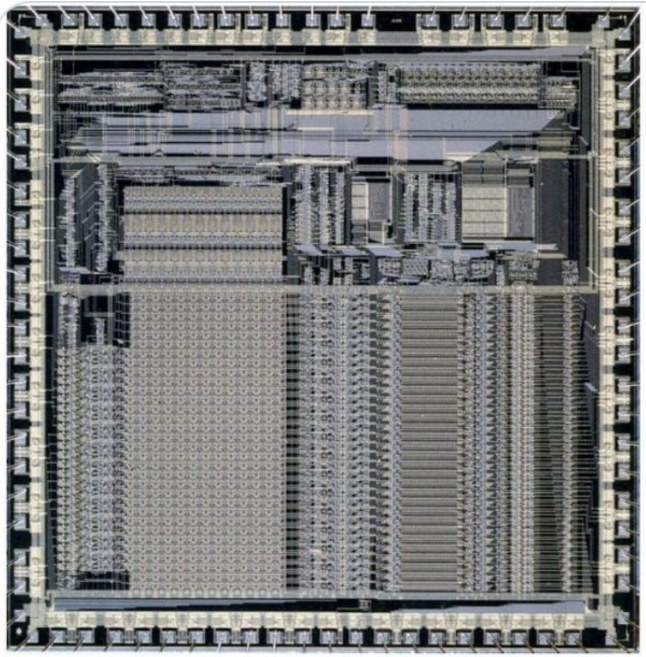
What do we mean when we say code?

What is a Programming Language and why do we need one?

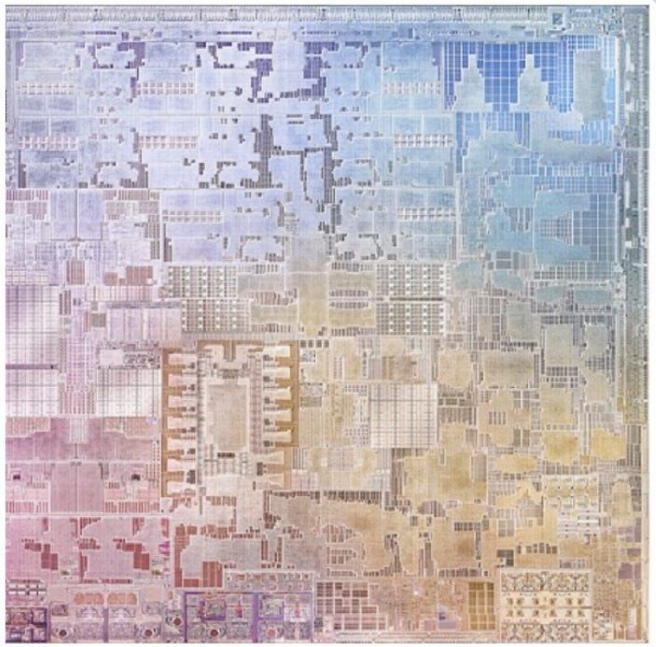


What is a Programming Language and why do we need one?

how it started: how it's going:



ARM1 processor (1985)
25 thousand transistors



Apple M1 processor (2020)
8-core ARM, 16 billion transistors

What is a Programming Language and why do we need one?

x = 5 + 3

5 is stored as `00000101`
3 is stored as `00000011`

First steps

Let’s run the first line of code

```
1    print ( ‘Hello world!’ )
```

```
Hello world!
```

To call a function in Python, you have to:

(1) Type the name of the function
(2) The parameter in round brackets

function (parameter)

Comments

Notes in the code that explain
(1) what the code is doing
(2) why certain decisions were made
(3) prevent code from running without removing it

They are crucial for readability and for revision by yourself and others

They are called with # or ##

```
1    # Print a greeting to the user
2    print ("Hello, it's nice to meet you!")
```

```
Hello, it's nice to meet you!
```

Comments

Note that double quotes (“) were used instead of single quotes this time (’)

If single quotes are used:

```
1    # Print a greeting to the user
2    print (‘Hello, it’s nice to meet you!’)
```

Execution error

❗ SyntaxError: invalid syntax (line 1)

Variables

Inputs

Allows users to submit data that is stored as variables

input ()

```
1    your_firstname = input()
```

```
abcd
```

```
1    your_surname = input()
```

```
xyz
```

Variables

Inputs

The submitted data can now be used in subsequent code

```
1    your_name = your_firstname + your_surname
2    print (your_name)
```

```
Abcd xyz
```

Variables

Mini-exercise

Modify the code to add a space between abcd and xyz

Variables

Mini-exercise

Modify the code to add a space between abcd and xyz

```
1    your_name = your_firstname + “ “ + your_surname
2    print (your_name)
```

```
abcd xyz
```

Numerical Operations

Basic numerical operations

(1) Addition **+**
(2) Subtraction **–**
(3) Multiplication ***
(4) Division */*
(5) Power **^**
(6) Round **round(**number**,** decimal places**)**
(7) Remainder **%**
(8) Range **in range(**min**,** max**)**

```
1    1 + 2
```

```
3
```

Numerical Operations

Data type

Determines what operations the data can undergo

type()

Data type	Abbreviation	Description
String	str()	Text
Integer	int()	Whole numbers
Float	float()	Decimal numbers
List	list[]	Nested list
Tuple	tuple()	Immutable ordered sequence
Boolean	bool()	True or False value
None	NoneType	Absence of value

Numerical Operations

Mini-exercise

Find the data type of **your_name**

Numerical Operations

Mini-exercise

Find the data type of **your_name**

```
1    type (your_name)
```

```
str
```

Note that some data types can be cast (converted)

```
1    type (str(3))
```

```
str
```

Numerical Operations

Exercise – Create a number program

The answer should be 10 every time

Numerical Operations

Exercise – Create a number program

(1) Accept an input for a number
(2) Add 5
(3) Multiply by 3
(4) Subtract 15
(5) Divide by the input
(6) Add 7

The answer should be 10 every time

Numerical Operations

Exercise – Create a number program

```
# receive input
number = int(input())

# perform operations
output = ((((number + 5)* 3)- 15)/ number)+ 7

# print output
print(output)
```

Numerical Operations

Exercise – Performing unit conversions

(1) Accept an input for height in centimeters
(2) Convert this to feet and inches
(3) Print the result as Height: x ft y in.

1 inch = 2.54 cm and 1 foot = 12 inches

Numerical Operations

Exercise – Performing unit conversions

```
1    # height_converter – cm to feet and inches
2
3    # height in centimeters
4    height_cm = float(input())
5
6    # convert to feet and inches
7    height_inches = height_cm/2.54.
8    height_feet = str(int(height_inches/12))
9    remaining_inches = str(round(height_inches % 12, 2))
10
11   # print final result
12   print('Height:' + height_feet + ' ft ' + remaining_inches + ' in’)
```

List Operations

Lists

Hold an ordered sequence of elements
Initiated with [], and elements are separated by commas

```
1    my_list = [1, ‘apple’, 3.14, [2, 3, 4]]
```

(1) The order of the elements is always maintained
(2) The content can be changed after creation (mutable)
(3) Elements can be of mixed types

List Operations

Indexing

Access an element by its position in the list

A list starts with position 0

```
1    first_element = my_list[0]
2    print (first_element)
```

```
1
```

Index (Position)	Value
0	1
1	'apple'
2	3.14
3	[2, 3, 4]

List Operations

Mini-exercise

Print the final element

```
1    final_element = my_list[3]
2    print (final_element)
```

```
[2, 3, 4]
```

Index (Position)	Value
0	1
1	'apple'
2	3.14
3	[2, 3, 4]

List Operations

Slicing

sublist = my_list[start:end]

Start: first element in the slice to be included.
End: last element the list goes up to, but does not include.

If the start is omitted, it defaults to 0.
If the end is omitted, it defaults to the last element of the list.

Negative indices start from the end of the list and work to the start.

- -1 is the last element
- -2 is the second to last element
- … etc

List Operations

Slicing

sublist = my_list[start:end]

```
1    sublist = my_list[1:3]
2    print (sublist) # Returns the first, second and third
elements
```

```
['apple', 3.14]
```

```
1    sublist = my_list[:2]
2    print (sublist) # Returns the first two elements
```

```
[1, 'apple']
```

List Operations

Slicing

sublist = my_list[start:end]

```
1    sublist = my_list[2:]
2    print (sublist) # Returns the last two elements
```

```
[3.14, [2, 3, 4]]
```

List Operations

Mini-exercise

Now, print the last three items of the list

```
1    sublist = my_list[1:]
2    print (sublist) # Returns the last three elements
```

```
['apple', 3.14, [2, 3, 4]]
```

List Operations

Length

Total number of elements it contains

len ()

Takes the list as its argument and returns an integer

```
1  length = len(my_list)
2  print (length)
```

```
4
```

List Operations

Append

Add an element to the end of the list

my_list.append()

```
1    a_city = ‘London’
2    cities = [‘New York’, ‘Tokyo’]
3    cities.append(a_city)
4    print(cities)
```

```
[‘New York’, ‘Tokyo’, ‘London’]
```

If statements

Creates a logical condition that needs to be fulfilled before a segment of code runs

```
1    # Define a variable with a number
2    number = 15
3
4    # Check if the number is greater than 10
5    if number > 10:
6        print (cities)
```

```
The number is greater than 10
```

If statements

Logical operators

(1) Greater than **>**
(2) Less than **<**
(3) Equals **=**
(4) Equals that returns True as a value **==**
(5) Not equals **!=**
(6) Less than or equal to **<=**
(7) Greater than or equal to **>=**
(8) Add one **+= 1**
(9) Subtract one **-= 1**

If statements

Mini-exercise

Let's make sure London is only added once

```
1        a_city = 'London'
2        cities = ['London', 'New York', 'Tokyo']
3
4        # Start code
5
6        # End Code
7        print (cities)
```

```
['London', 'New York', 'Tokyo']
```

If statements

Mini-exercise

Let's make sure London is only added once

```
1    a_city = 'London'
2    cities = ['London', 'New York', 'Tokyo']
3
4    if cities[0] != 'London':
5          cities.append(a_city)
6
7    print (cities)
```

```
['London', 'New York', 'Tokyo']
```

If statements

Mini-exercise

Cross-check by changing position 0 to Amsterdam

```
1    a_city = ‘London’
2    cities = [‘Amsterdam’, ‘New York’, ‘Tokyo’]
3
4    if cities[0] != ‘London’:
5          cities.append(a_city)
6
7    print (cities)
```

```
[‘Amsterdam’, ‘New York’, ‘Tokyo’, ‘London’]
```

If statements

Mini-exercise

Let’s make sure London is only added once

```
1   a_city = 'London'
2   cities = ['London', 'New York', 'Tokyo']
3
4   if a_city not in cities:
5       cities.append(a_city)
6
7   print(cities)
```

[‘London’, ‘New York’, ‘Tokyo’]

Sets

Set

$$[x_1, x_2, \ldots]$$

Built-in data type that holds an unordered collection of unique elements
Duplicates are automatically removed/not added.

Sets

Set

```
1    # Create a set with some cities
2    cities = set(['New York', 'Tokyo'])
3    # Define a new city
4    a_city = 'London'
5
6     # Add the city to the set
7    cities.add(a_city)
8    # Print the set
9    print (cities)
10
11    # Try adding the same city again
12    cities.add(a_city)
13    # Print the set again
14   print (cities)
```

```
{'London', 'New York', 'Tokyo'}
{'London', 'New York', 'Tokyo'}
```

Sets

Trying this with another city, you will that sets do not preserve the order

```
1    # Create a set with some cities
2    cities = set(['London', 'New York', 'Tokyo'])
3    # Define a new city
4    another_city = 'Paris'
5
6    cities.add(another_city)
7    # Print the set
8    print (cities)
9
10    # Try adding the same city again
11    cities.add(another_city)
12    # Print the set again
13   print (cities)
```

```
{'Tokyo', 'New York', 'London', 'Paris'}
{'Tokyo', 'New York', 'London', 'Paris'}
```

Flow loops

For loop

Used to iterate over a sequence

for element in sequence:
 Do something with the element

```
1    for i in range (1,6):
2        print (i)
```

```
1
2
3
4
5
```

Flow loops

Breaks

Interrupts the for loop

break

```
1    for i in range (1,6):
2        if i == 3:
3            break
4        print (i)
```

```
1
2
```

Flow loops

While loop

Keeps executing code as long as a specified condition is true

while condition:

```
1    i = 1
2    while i < 6:
3        print (i)
4        i += 1
```

```
1
2
3
4
5
```

Flow loops

Function

A reusable piece of code that performs a specific task
Useful when the same code is applied multiple times
Often uses loops, but it is not limited to this purpose

```
def function_name(n):
return value
```

Calling a function -> function_name(n)

Flow loops

Function

Let’s generate a Fibonacci sequence

```
1   # Initiate the function with an input n
2   def generate_fibonacci(n):
3   # First two terms are initiated, and an empty set generated
4       a, b = 0, 1
5       fib_sequence = []
6
7   # Use a for loop to generate the rest of the sequence
8       for _ in range (n):
9           fib_sequence.append(a)
10          a, b = b, a + b
11      return fib_sequence
12
13  # Use the function for the first 20 terms of the Fibonacci sequence
14  generate_fibonacci(20)
```

Flow loops

Mini-exercise

Break out of the function if the sequence is greater than 50

Flow loops

Mini-exercise

Break out of the function if the sequence is greater than 50

```
1    # Initiate the function with an input n
2    def generate_fibonacci(n):
3    # First two terms are initiated, and an empty set generated
4        a, b = 0, 1
5        fib_sequence = []
6
7    # Use a for loop to generate the rest of the sequence
8        for _ in range (n):
9            if a > 50:
10               break
11           fib_sequence.append(a)
12           a, b = b, a + b
13       return fib_sequence
```

```
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
```

Flow loops

Mini-exercise

Use a while loop to generate a sequence up to the input number

Flow loops

Mini-exercise

Use a while loop to generate a sequence up to the input number

```
1   # Initiate the function with an input x
2   def generate_fibonacci(x):
3   # First two terms are initiated, and an empty set generated
4         a, b = 0, 1
5         fib_sequence = []
6
7   # Use a while loop to restrict the sequence
8         while a <= x:
9               fib_sequence.append(a)
10              a, b = b, a + b
11        return fib_sequence
12  x = 100
13  generate_fibonacci(100)
```

```
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]
```

“Homework”

Exercises

(1) Calculating the Area of a Rectangle

(2) Managing a To-Do List

(3) Grade Classifier

(4) Multiplication Table Printer

(5) Countdown Timer Without and With Delay

Exercise 1

Calculating the Area of a Rectangle

Write a Python script that calculates the area of a rectangle. You should:

(1) Define a variable length and assign it a value of 10.

(2) Define another variable width and assign it a value of 5.

(3) Calculate the area of the rectangle (length * width) and store it in a variable named area.

(4) Print the area of the rectangle.

Exercise 2

Managing a To-Do List

Create a simple to-do list application. You should:

(1) Create a list named todo_list containing at least three tasks as strings.

(2) Add a new task to the end of the list using a list operation.

(3) Remove the first task from the list.

(4) Print the updated to-do list.

Exercise 3

Grade Classifier

Write a program that classifies student grades. You should:

(1) Define a variable grade and assign it a value.

(2) Use an if-else statement to print:

a) "Outstanding" if the grade is greater than or equal to 90.

b) "Good" if the grade is between 80 and 89.

c) "Needs Improvement" if the grade is below 80.

Exercise 4

Multiplication Table Printer

Create a script that prints the multiplication table for a given number. You should:

(1) Define a variable number with a value of 5 (or any number of your choice).

(2) Use a for loop to iterate from 1 to 10.

(3) For each iteration, print the product of the number and the iterator.

Exercise 5a

Countdown Timer

Write a program that acts as a simple countdown timer. You should:

(1) Define a variable countdown_start and assign it a value of 10.

(2) Use a while loop to count down to zero from the starting value.

(3) Print each number and "Liftoff!" when the countdown reaches zero.

Feel free to modify the values and experiment with the code to see different results. Good luck!

Exercise 5b

Countdown Timer with Delay

Modify the previous countdown timer program to include a 1-second delay between each number, using the time.sleep() function. You should:

(1) Import the time module at the beginning of your script.

(2) Define a variable countdown_start and assign it a value of 10.

(3) Use a while loop to count down to zero from the starting value.

(4) After printing each number, use time.sleep(1) to pause the execution for 1 second.

(5) Print "Liftoff!" when the countdown reaches zero.