
Summer Workshop for Master of Science in Generative AI and the
Humanities (MScGAH): Morning Session on 31 July 2024

Basic Programming with the Python Language

Speaker: Dr. WAN, Mingyu Clara

EMAIL: pku.clara@gmail.com

Department of Chinese and Bilingual Studies
The Hong Kong Polytechnic University

Pre-talk Discussion



Q: Is programming/coding/computing still essential when GenAI technologies like GPT-4 can do coding automatically in seconds?"

Pre-talk Discussion (Cont.)

Suppose we want to generate a word cloud for “Heart Sutra 心經”

write a piece of code in Python to generate a word cloud of the given file.

Certainly! Below is a Python script that reads the content of the file 心經.txt, and generates

To generate a word cloud for a file containing traditional Chinese text, you'll need to use libraries like `wordcloud`, `jieba` for text segmentation, and `matplotlib` for displaying the word cloud. Below is an example of how you can achieve this in Python:

1. Install the necessary libraries:

```
sh pip install wordcloud jieba matplotlib
```

2. Python code to generate the word cloud:

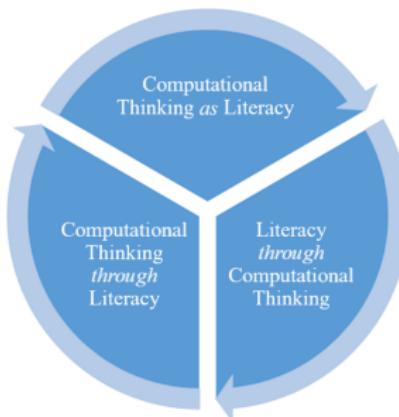
```
python  
import jieba  
from wordcloud import WordCloud  
import matplotlib.pyplot as plt
```



Chat-GPT + Prompt + Colab ⇒ Done: programming unnecessary?

Programming is an important Thinking

- What if the code went wrong? Does GenAI debug properly?
- Can GenAI be adaptable when **data** is different?
- Can GenAI work out an efficient **algorithm** in the program?...

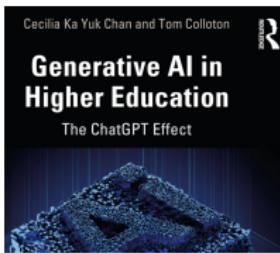


A three-dimensional framework for understanding computational literacy.¹

¹Jacob, S. R., & Warschauer, M. (2018). Computational thinking and literacy. *Journal of Computer Science Integration*, 1(1).

Computational Literacy for AI Literacy

Computational Literacy is one among the seven key components of AI literacy in Chan & Colloton (2024)²



“Computational Literacy, or computational thinking, emphasizes problem-solving processes and involves formulating problems and their solutions in ways that computers can also execute. It emerged as a crucial skill in the context of computing and programming education.”

²Chan, C. K. Y., & Colloton, T. (2024). Generative AI in Higher Education: The ChatGPT Effect (p. 287). Taylor & Francis.

Pre-talk Discussion (Cont.)

Q: Is programming/coding/computing still essential when GenAI technologies like GPT-4 can do coding automatically in seconds?"

A: Yes, it is still very important.

- GenAI can be a powerful assistant(~~x~~ a replacement) in programming.



- Hence, that's why we're here:
 - to develop computational literacy through the fundamentals of programming, also an essential LO in the MScGAH program.

Table of Contents

1. Introduction to Programming Language
2. Foundations in Python Programming
3. Basic Data Analysis and Visualization

Introduction to Programming Language

Programming Language

- **Definition:** Programming languages are formal languages comprising **a set of instructions** that produce output.
- **Purpose:** They are used to implement **algorithms** and enable **communication** between humans and machines.
- **Importance:** They are essential for developing AI systems, software applications, and a wide range of **digital solutions**.



Q: How many program languages? and commonly used ones?

A: 8,945+; 50-100



image source

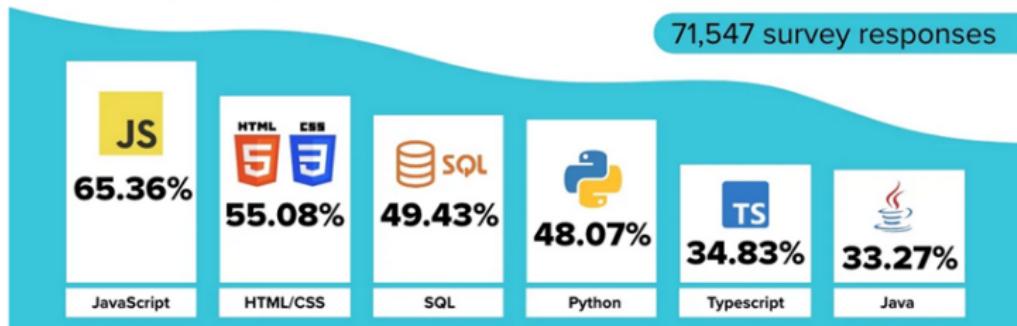
Statistics in using program languages (in general)

May 2021	Programming Language	Ratings	Chart Ratings
1	C	13.38%	
2	Python	11.87%	
3	Java	11.74%	
4	C++	7.81%	
5	C#	4.41%	
6	Visual Basic	4.02%	
7	JavaScript	2.45%	
8	Assembly language	2.43%	
9	PHP	1.86%	
10	SQL	1.71%	

TOIBE index for 2021

[image source](#)

Statistics in using program languages (by developers)

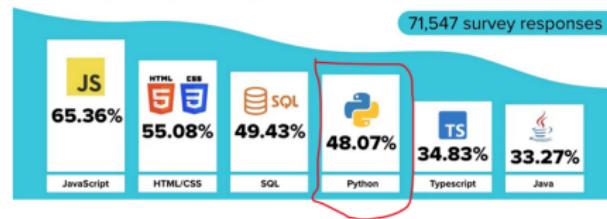


Among developers

[image source](#)

Python is popular:)

May 2021	Programming Language	Ratings	Chart Ratings
1	C	13.38%	
2	Python	11.87%	
3	Java	11.74%	
4	C++	7.81%	
5	C#	4.41%	
6	Visual Basic	4.02%	
7	JavaScript	2.45%	
8	Assembly language	2.43%	
9	PHP	1.86%	
10	SQL	1.71%	



Q: which program language do you use/want to learn most?

(depending on the purpose/role of the programmer: statistics-R; nlp-Python; Software-Java/C; website-HTML,CSS; database-SQL)

Types of Programming Languages

- High-Level Languages:
 - Examples: Java, Perl
 - Characteristics: More abstract, closer to human language.
- Low-Level Languages:
 - Examples: Assembly, Machine Code
 - Characteristics: Closer to machine language, to hardware.

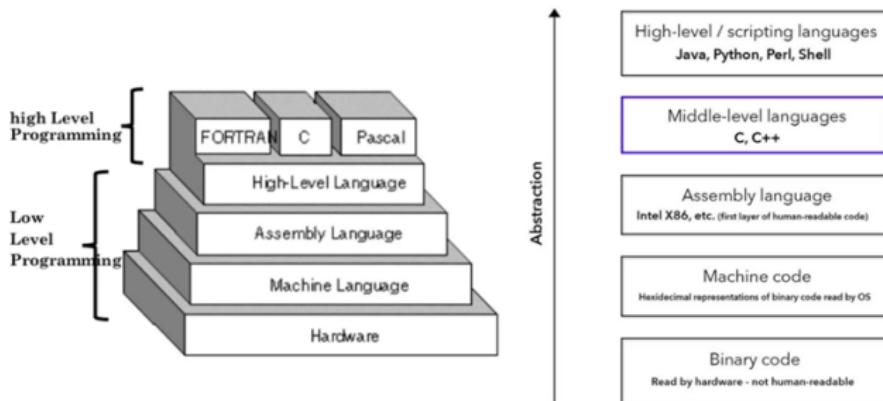


image source

How about our foci: Python?

Python is a high-level program language, syntax close to English.



```
# Python3 program to find maximum
# in arr[] of size n
def largest(arr, n):
    ans = max(arr)
    return ans;

# Driver code
if __name__ == '__main__':
    arr = [10, 324, 45, 90, 9808]
    n = len(arr)
    print ("Largest in given array ", largest(arr, n))
```

image source

More features of Python

Python is a **versatile, object-oriented, high-level programming** language that emphasizes simplicity, readability, and a comprehensive standard library, making it well-suited for a wide range of applications, from scripting and web development to data analysis and machine learning.

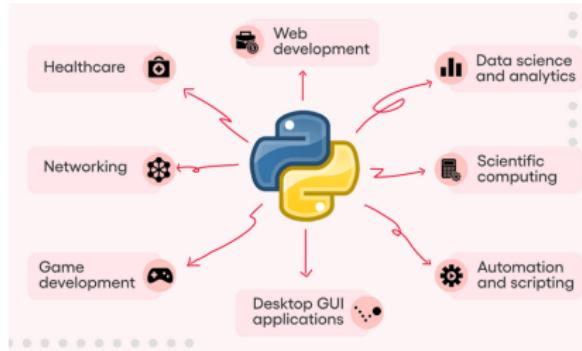


image source

Reflection: As a computational linguist or data scientist, **Python** is a most commonly programming language (easy and many existing libraries). In addition, **R**, **LaTeX**, **C**, **Java**, etc. are also popularly used for different purposes.

Object-Oriented Programming (OOP)

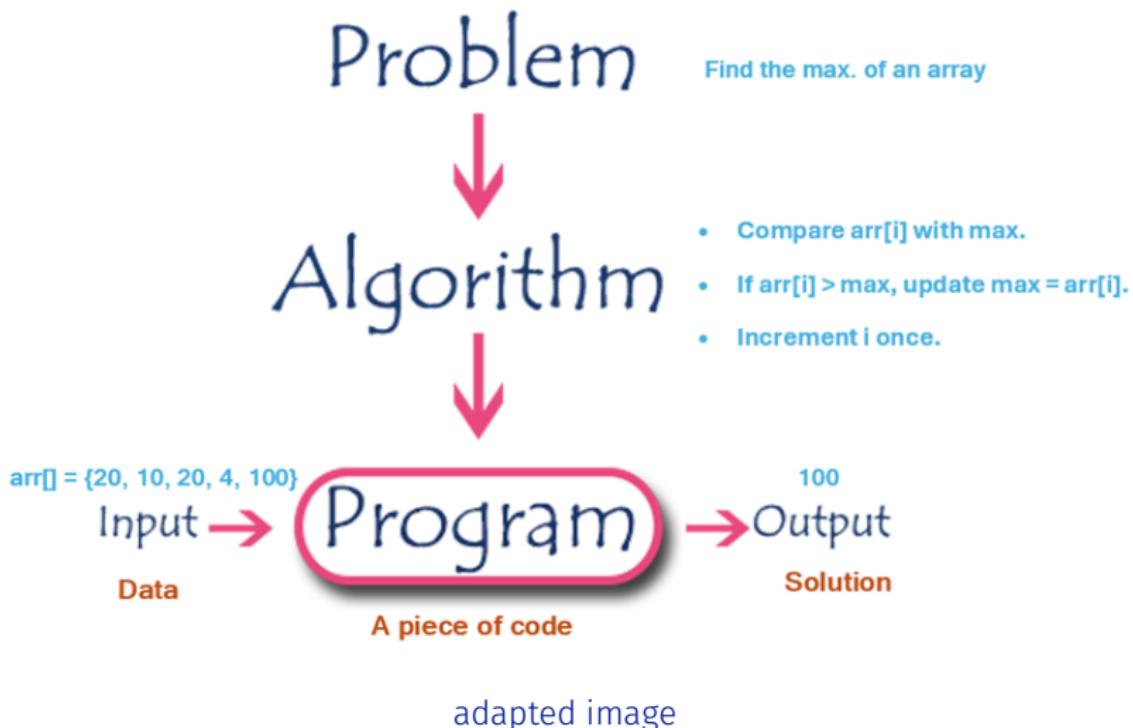
- **Concept:** a programming paradigm that provides a means of structuring programs so that *properties* and *behaviors* are bundled into **objects**.
 - For example, an **object** could represent a dog with *properties* like a *name*, *age* and *behaviors* such as *description*, *speak*, and so on.

```
class Dog:  
    species = "Canis familiaris"  
  
    def __init__(self, name, age):  
        self.name = name  
        self.age = age  
  
        # Instance method  
    def description(self):  
        return f"{self.name} is {self.age} years old"  
  
        # Another instance method  
    def speak(self, sound):  
        return f"{self.name} says {sound}"
```

sample code

Versus: Procedural Programming: e.g.: C, Pascal, Fortran, COBOL.

Program = Data + Algorithm → Problem-solving?



Data Structures

- Primitive Data Structures

- e.g. Integer, Float, Character, Point...

- Non-Primitive Data Structures

- Linear data structures (Sequential)

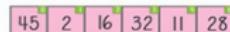
- Static, e.g. array, list
- Dynamic, e.g. queue, stack, linked list

- Non-linear data structures (Hierarchical)

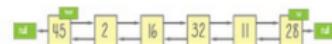
- e.g. tree, graph

Check this video for more

Array



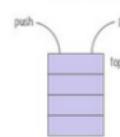
Linked List



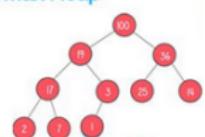
Queue



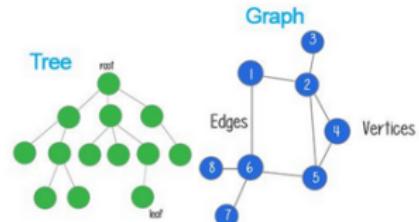
Stack



Max Heap

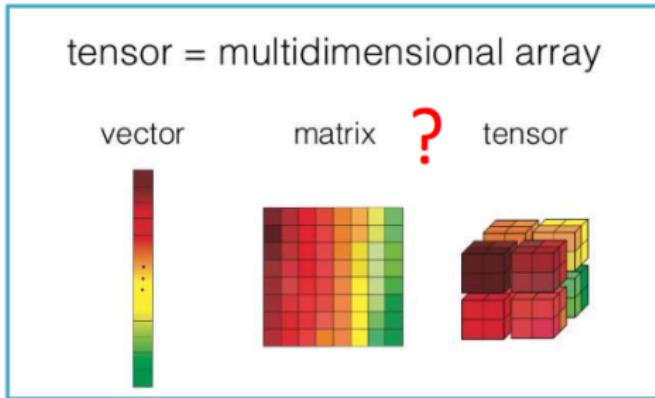


Graph



adapted image

Data can be high-dimensional and complex

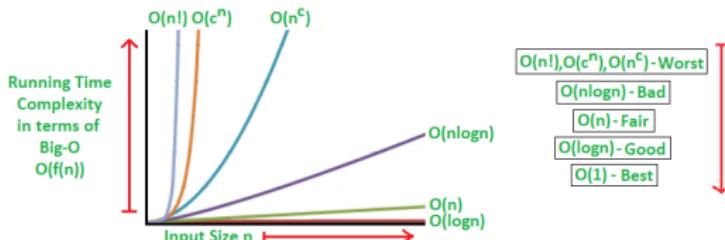


Data in various forms of *vectors*, *matrices*, and *tensors* are fundamental elements for many modern AI and machine learning models, such as Decision Tree (DT), Support Vector Machines (SVMs), Recurrent Neural Networks (RNNs), Convolutional Neural Networks (CNNs), Graph Neural Networks (GNNs), Transformers, and so on...

Algorithms Analysis

Solutions to a problem can be multiple. The **efficiency** of an algorithm matters. Analysis of algorithms is to compare algorithms (or solutions) mainly in terms of running time but also in terms of other factors (e.g., memory, developer effort, etc.) - **Complexity**: asymptotic notation

- **Time Complexity**: Time complexity is used to measure the amount of time required to execute the code.
- **Space Complexity**: Space complexity means the amount of space required to execute successfully the functionalities of the code.



Recommended Readings (books)

- **Programming Thinking**
 - “Code: The Hidden Language of Computer Hardware and Software” (Petzold, C. 2000)
 - “Guide to Programming for the Digital Humanities: Lessons for Introductory Python” (Kokensparger, B. 2018)
- **Python Programming**
 - “Learn Python the Hard Way” (Shaw, Z. A. 2024)
 - “Think Python: How to Think Like a Computer Scientist” (Downey, A. et al. 2016)
- **Data Structures & Algorithms**
 - “Data Structures and Algorithms in Python” (Goodrich, M. T. et al. 2013)
- **Machine/Statistical Learning and Applications**
 - “Mathematics for Machine Learning” (Deisenroth, M. P. et al. 2020)
 - “Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow” (Géron, A. 2022)

- Online tutorials
 - w3Schools
 - Python Geeks
 - Data analysis with Python
 - Data visualization with Python
- GitHub repertoires
 - Python Basics
 - Natural Language Processing with Python (NLTK)
 - Language Processing
- For humanities
 - Python for Humanists
 - Programming for Linguists

Recap via MentiMeter

Recall the previous concepts via a few questions in [MentiMeter](#).

Join at [menti.com](#) | use code 63 63 50 2

Instructions

Go to

www.menti.com

Enter the code

63 63 50 2



Or use QR code



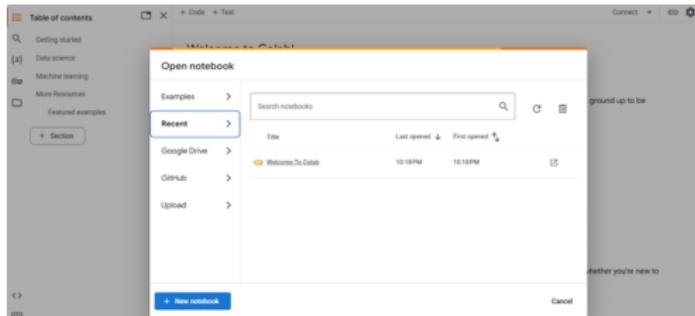
Foundations in Python Programming

Getting Ready for Python Programming

- **Offline Programming (local setups)**
 - Install [Python](#) on your computer;
 - Choose a code editor, and run the code on a terminal
(e.g., [Notepad++](#), [Sublime](#))
 - or an IDE (Integrated Development Environment)
(e.g., [PyCharm](#), [Visual Studio Code](#));
 - Set up a virtual environment for packages and dependencies;
 - !pip install all packages to use, such as:
 - [NumPy](#): for scientific computing and data manipulation
 - [Pandas](#): for data analysis and manipulation
 - [Matplotlib](#): for data visualization
 - [Django](#): for web development
 - [TensorFlow](#): for machine learning and deep learning
- **Online Programming**
 - Simpler, most packages installed, and not using local space, e.g.
[replit](#), [Google Colab \(Colab\)](#) (for demonstration in this talk)

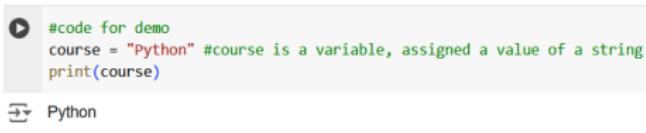
Getting Started with Google Colab³

- Visit the Google Colab website (<https://colab.research.google.com>)
- Sign in with your Google account
- Create a new notebook or open an existing one
- Start coding and exploring the power of Colab!



³Google Colab (short for Collaborative Notebook) is a cloud-based Jupyter Notebook environment, which allows you to write and execute Python code directly in your web browser. Colab is provided and hosted by Google, making it a free and accessible platform for data analysis, machine learning, and more...

Key Notions

- **Variables**
 - a designated place in memory where data is kept;
 - Assign Values to Variables;
 - give a variable a value by using the assignment operator =
- 

```
#code for demo
course = "Python" #course is a variable, assigned a value of a string
print(course)
```
- **Constants**
 - a variable type whose value cannot be altered, e.g., PI = 3.14;
 - constants are typically declared and allocated in a module.
- **Literals**
 - A fixed value of a specific data type. Literals are constants that are self-explanatory, e.g. Boolean literals: True or False.

Key Notions: Keywords

- **Keywords**

- a set of predefined words, making coding meaningful;
- reserved words that can not be used as a variable name, function name, or any other identifier.



```
import keyword #package

# printing all keywords at once using "kwlist()"
print("The list of keywords is : ")
print(keyword.kwlist)
print(len(keyword.kwlist))
```



The list of keywords is :

```
['False', 'None', 'True', 'and', 'as', 'assert', 'async', 'await', 'break', 'class', 'continue', 'def', 'del', 'elif', 'else',  
35
```

Key Notions: Identifiers

Identifiers

- a user-defined name given to a variable, function, class, etc.
- a combination of character, digits, and an underscore.
- case-sensitive

Rules for Naming Python Identifiers

- It cannot be a reserved python keyword.
 - It should not contain white space.
 - It can be a combination of A-Z, a-z, 0-9, or underscore.
 - It should start with an alphabet character or an underscore (_).
 - It should not contain any special character other than an underscore (_).
-
- e.g. `var1`, `_var1` (valid); `1var`, `var#` (invalid)
 - How about `my_name1`, `@my-name`, `_my_name2`?

Key Notions: Statements

- Single Line Statements
 - pieces of code that a Python interpreter can carry out

- Multi-line Statements

- To extend the statement to one or more lines we can use braces , parentheses (), square [], semi-colon “;”, and continuation character slash “\”.

```
▶ #initializing a string using explicit multi-line statement
speaker = "Clara" + \
           " M.Y." + \
           " Wan"
print(speaker)
```

→ Clara M.Y. Wan

Key Notions: Indentation

Indentation

- a way of telling a Python interpreter that the group of statements belongs to a particular block of code.

```
def read_file(file_path):
    with open(file_path, 'r', encoding='utf-8') as file:
        text = file.read() #indentation within open file
    return text
```



```
speaker = ['Clara', 'Vincent', 'John']
if 'Clara' in speaker:
    print("Coffee for breakfast") #indentation within if statement
else: print("no breakfast")
```



Coffee for breakfast

created with 'tab' or four spaces; try changing the name.

Key Notions: Comments

Single Line Comments

- The hash (#) symbol denotes the starting of a comment.

```
# This is the syntax of a comment in Python
print("GFG")

# Comments dont have any effect on the interpreter / output
```

Multi-line Comments

- enclosed by triple double quotes (""""") or triple single quotes ('''') to provide detailed explanations, documentation, or notes about the code, and they can span multiple lines.

""" Multi-line comment used
print("Python Comments") """

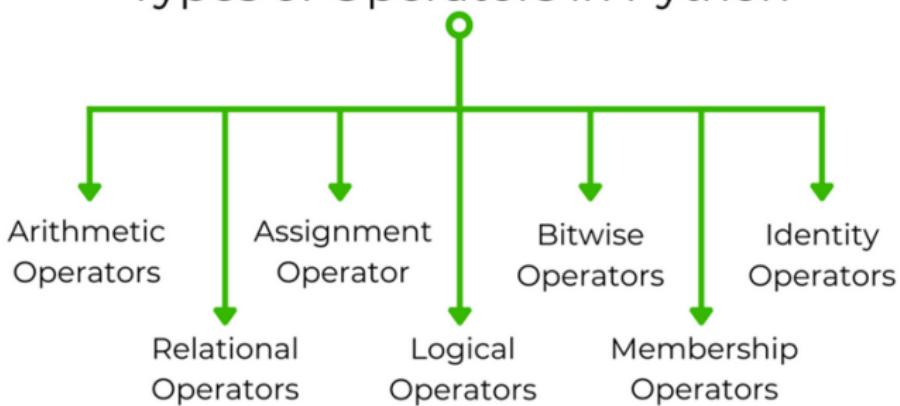
Numbers

- Python supports several numeric data types, including:
 - **Integer** (int): e.g., 20, -3
 - **Float**: e.g., 1.2, 3/4
 - **Complex**: e.g., 1-3i, 2+5j
- **Arithmetic Operations**: Addition (+), Subtraction (-), Multiplication (*), Division (/), Integer Division (//), Modulo (%), Exponentiation (**)
- **Type Conversion**:
 - int(x): Convert x to an integer
 - float(x): Convert x to a floating-point number
 - complex(x): Create a complex number from x

Key Notions: Operators

Operators are symbols that perform operations on operands (values or variables) ([Comprehensive learning resources](#))

Types of Operators in Python



Key Notions: Comparison Operators

Except for **Arithmetic Operators**, there are:

- **Comparison Operators:** <, <, >=, <=, !=, ==



```
#practice comparison operators
a = 50
b = 65
print('a > b is',a>b)
print('a < b is',a<b)
print('a == b is',a==b)
print('a != b is',a!=b)
print('a >= b is',a>=b)
print('a <= b is',a<=b)
```



```
a > b is False
a < b is True
a == b is False
a != b is True
a >= b is False
a <= b is True
```

Key Notions: Logical Operators

- Logical Operators: 'and', 'or', 'not'



```
#logical operators
a = True
b = False
print('a and b is',a and b)
print('a or b is',a or b)
print('not a is',not a)
```



a and b is False
a or b is True
not a is False

Key Notions: Assignment Operators

- **Assignment operators** are used to assign values to variables.
The most basic assignment operator is the = (equal) sign.
 - e.g., $x = 5$, assigns the value 5 to the variable x .
- **Augmented Assignment operators** combine an arithmetic or bitwise operation with the assignment;

Operator	Description	Example
<code>+=</code>	Addition assignment	$x += 3$ is equivalent to $x = x + 3$
<code>-=</code>	Subtraction assignment	$x -= 2$ is equivalent to $x = x - 2$
<code>*=</code>	Multiplication assignment	$x *= 4$ is equivalent to $x = x * 4$
<code>/=</code>	Division assignment	$x /= 2$ is equivalent to $x = x / 2$
<code>%=</code>	Modulus assignment	$x %= 5$ is equivalent to $x = x \% 5$
<code>**=</code>	Exponentiation assignment	$x **= 3$ is equivalent to $x = x ** 3$
<code>//=</code>	Integer division assignment	$x //= 2$ is equivalent to $x = x // 2$
<code>&=</code>	Bitwise AND assignment	$x &= 0b1010$ is equivalent to $x = x \& 0b1010$

Data Types: List

List

- Lists are ordered collections of items, including numbers, strings, and other objects, defined using square brackets [] and elements are separated by commas.



```
# Creating a List with
# mixed type of values
# (Having numbers and strings)
List = [1, 2, 'Clara', 4, 'is', 6, 'Sleepy']
print(List)
```



[1, 2, 'Clara', 4, 'is', 6, 'Sleepy']

Data Types: List (Cont.)

List Operations:

- **Accessing elements:** my_list[0] (first element), my_list[-1] (last element)
- **Slicing:** my_list[1:4] (elements from index 1 to 3)
- **Concatenation:** [1, 2] + [3, 4] (creates [1, 2, 3, 4])
- **Repetition:** [1, 2] * 3 (creates [1, 2, 1, 2, 1, 2])
- **Membership:** "four" in my_list (checks if "four" is in the list)



```
# Creating a List of numbers
List2 = [10, 20, 14]
print(len(List2))
```

 3

Data Types: List (Cont.)

List Methods:

- *list.append (element)*

```
▶ # Adds List Element as value of List.  
List = ['Mathematics', 'AI', 1997, 2000]  
List.append(2024)  
print(List)
```

```
→ ['Mathematics', 'AI', 1997, 2000, 2024]
```

- *list.insert(position, element)*

```
▶ List = ['Mathematics', 'AI', 1997, 2000]  
# Insert at index 2 value 10087  
List.insert(2, 2022)  
print(List)
```

```
→ ['Mathematics', 'AI', 2022, 1997, 2000]
```

Data Types: List (Cont.)

List Methods:

- *min/max(iterable, *iterables[, key])*



```
numbers = [5, 2, 8, 1, 9]
print(min(numbers))
print(max(numbers))
```



1
9

- *list.sort([key, [Reverse_flag]])*



```
List = [2.3, 4.445, 3, 5.33, 1.054, 2.5]

#Reverse flag is set True
List.sort(reverse=True)

#List.sort().reverse(), reverses the sorted list
print(List)
```



[5.33, 4.445, 3, 2.5, 2.3, 1.054]

Data Types: List (Cont.)

List Methods:

- *list.pop([index]):* Removes an item from a specific index in a list.
- *del list.[index]:* Deletes an element from the list by index.
- *list.remove(element):* Removes a specific element by value/name.

```
▶ List = [2.3, 4.445, 3, 5.33, 1.054, 2.5]
  #pop
  print(List.pop())
  print(List.pop(0))
  #del
  del List[0]
  print(List)
  #remove
  List.remove(3)
  print(List)
```

```
→ 2.5
  2.3
  [3, 5.33, 1.054]
  [5.33, 1.054]
```

Data Types: String

String are a sequence of characters that can include letters, numbers, symbols, and even spaces. Strings are defined using single quotes ('), double quotes (""), or triple quotes (""" or ''') for multi-line strings.

```
# Single-quoted strings
string1 = 'Hello, world!'
string2 = 'Python is awesome.'

# Double-quoted strings
string3 = "I love coding."
string4 = "Python's syntax is clean."

# Triple-quoted strings (for multiline)
string5 = """This is a
multiline
string."""
```

String Operations

Similar to List operations.

```
▶ # string operations
# Concatenation
print(string1 + " " + string2) # Output: Hello, world! Python is awesome.

# Repetition
print(string3 * 3) # Output: I love coding.I love coding.I love coding.

# Accessing characters
print(string1[0]) # Output: H
print(string2[-1]) # Output: .

# Slicing
print(string4[7:13]) # Output: syntax
print(string5[:5]) # Output: This

# String methods
print(string1.upper()) # Output: HELLO, WORLD!
print(string2.lower()) # Output: python is awesome.
print(string3.replace('love', 'like')) # Output: I like coding.
```

Formatting Strings

Formatting Strings allows you to insert values into a string in a readable and flexible manner.

```
# formatting strings
name = "Alice"
age = 25
print("My name is {} and I'm {} years old.".format(name, age))
# Output: My name is Alice and I'm 25 years old.

print(f"My name is {name} and I'm {age} years old.")
# Output: My name is Alice and I'm 25 years old.
```

→ My name is Alice and I'm 25 years old.
My name is Alice and I'm 25 years old.

Data Types: Tuple

Tuple is an ordered collection of items, similar to a list. The main difference between tuples and lists is that tuples are **immutable**, meaning that the elements within a tuple cannot be modified after the tuple is created.

```
# Creating a tuple
point = (2, 3)
colors = ('red', 'green', 'blue')

# Tuple with a single element
single_tuple = (42,)

# Accessing elements
print(point[0]) # Output: 2
print(colors[1]) # Output: green

# Slicing
print(colors[0:2]) # Output: ('red', 'green')

# Unpacking
x, y = point
print(x, y) # Output: 2 3

# Concatenation
more_colors = colors + ('yellow', 'orange')
print(more_colors) # Output: ('red', 'green', 'blue', 'yellow', 'orange')

# Repetition
print(point * 3) # Output: (2, 3, 2, 3, 2, 3)
```

2
green
('red', 'green')
2 3
('red', 'green', 'blue', 'yellow', 'orange')
(2, 3, 2, 3, 2, 3)

Data Types: Dictionary

Dictionary is an unordered collection of **key-value pairs**. Dictionaries are one of the most versatile and commonly used data structures in Python, providing an efficient way to store, organize, and retrieve data.



```
# Creating a dictionary
person = {
    "name": "Alice",
    "age": 30,
    "city": "Hong Kong"
}

# Empty dictionary
empty_dict = {}
```

Accessing Dictionary Elements

```
▶ # Accessing values
print(person["name"]) # Output: "Alice"
print(person.get("age")) # Output: 30

# Adding new key-value pairs
person["country"] = "USA"
print(person) # Output: {'name': 'Alice', 'age': 30, 'city': 'New York', 'country': 'USA'}

# Updating existing values
person["age"] = 31
print(person) # Output: {'name': 'Alice', 'age': 31, 'city': 'New York', 'country': 'USA'}

# Removing key-value pairs
del person["city"]
print(person) # Output: {'name': 'Alice', 'age': 31, 'country': 'USA'}
```

```
→ Alice
30
{'name': 'Alice', 'age': 30, 'city': 'Hong Kong', 'country': 'USA'}
{'name': 'Alice', 'age': 31, 'city': 'Hong Kong', 'country': 'USA'}
{'name': 'Alice', 'age': 31, 'country': 'USA'}
```

Dictionary Methods

In Python, dictionaries provide a variety of built-in methods that allow you to perform various operations on the key-value pairs.

```
▶ # Get all keys
    print(list(person.keys())) # Output: ['name', 'age', 'country']

    # Get all values
    print(list(person.values())) # Output: ['Alice', 31, 'USA']

    # Get all key-value pairs
    print(list(person.items()))
    # Output: [('name', 'Alice'), ('age', 31), ('country', 'USA')]
```

→ ['name', 'age', 'country']
['Alice', 31, 'USA']
[('name', 'Alice'), ('age', 31), ('country', 'USA')]

Other Data Types

Complex Data Types: e.g. tree, graph, heap, stack, queue, and more...

- **Trees** are hierarchical data structures that consist of nodes connected by edges. They are commonly used to represent and navigate complex relationships, such as file systems, organizational structures, and decision-making processes.
- **Graphs** are data structures that consist of nodes (vertices) connected by edges. They are used to represent complex relationships and networks, and are often used in applications such as social media, transportation, and recommendation systems.
- **Heaps** are tree-based data structures that satisfy the heap property, where the value of each node is greater than (or less than) or equal to the values of its children. They are commonly used in priority queues, sorting algorithms, and other applications that require efficient access to the maximum or minimum element.
- and others...

Built-in Functions

Python comes with a wide range of built-in functions that you can use to perform common tasks. Here are some of the most useful ones:

Function	Description	Example
<code>print()</code>	Outputs a message to the console.	<code>print("Hello, World!")</code>
<code>len()</code>	Returns the length of a sequence (e.g., string, list, tuple).	<code>len("Python")</code> returns 6
<code>range()</code>	Generates a sequence of numbers.	<code>list(range(1, 6))</code> returns [1, 2, 3, 4, 5]
<code>input()</code>	Prompts the user for input and returns it as a string.	<code>name = input("What's your name? ")</code>
<code>type()</code>	Returns the type of an object.	<code>type(42)</code> returns

These are just a few examples of the many built-in functions available in Python. Explore the [Python documentation](#) to learn more!

User-defined Functions

In addition to the built-in functions, you can also create your own custom functions in Python. These are called user-defined functions.

```
▶ #user-defined functions
def function_name(parameter1, parameter2, ..., parameterN):
    """
        Docstring explaining the function's purpose.
    """
    # Function body
    return value

#calling a function
result = function_name(arg1, arg2, ..., argN)
```

User-defined Functions: Example

```
#Example: compare two numbers
def compare_numbers(num1, num2):
    """
    Compares two numbers and returns the larger of the two.

    Args:
        num1 (float): The first number to compare.
        num2 (float): The second number to compare.

    Returns:
        float: The larger of the two numbers.
    """
    if num1 > num2:
        return num1
    else:
        return num2

larger_number = compare_numbers(10, 5)
print(larger_number) # Output: 10
```

→ 10

Getting User Input

In Python, you can use the `input()` function to get user input. The `input()` function takes an optional prompt message as an argument and returns the user's input as a string.

```
[31] #getting user input
    user_input = input("Enter your name: ")
    print(f"Hello, {user_input}!")

    #converting input types
    age = int(input("Enter your age: "))
    height = float(input("Enter your height (in meters): "))

    #handling errors
    try:
        age = int(input("Enter your age: "))
    except ValueError:
        print("Invalid input. Please enter a number.")
```

→ Enter your name: clara
Hello, clara!
Enter your age: 18
Enter your height (in meters): 1.74
Enter your age: mama
Invalid input. Please enter a number.

Read A File

Opening a File → Reading File Contents → Closing the File (Using the with Statement) → Error Handling

```
▶ #reading file locally
#first, open a file
file = open("心經.txt", "r")

#Reading File Contents
contents = file.read() #Reads the entire contents of the file as a string.
#file.readline() #Reads a single line from the file.
#file.readlines() #Reads all lines from the file and returns them as a list of strings.
print(contents)

#file close
file.close() #to free up system resources after reading the file

...
#use with
with open("example.txt", "r") as file:
    contents = file.read()
    print(contents)
...
#handling error
try:
    file = open("example.txt", "r")
    contents = file.read()
    print(contents)
except FileNotFoundError:
    print("File not found.")
finally:
    file.close()
```

般若波羅蜜多心經
觀自在菩薩。行深般若波羅蜜多時。
照見五蘊皆空。度一切苦厄。

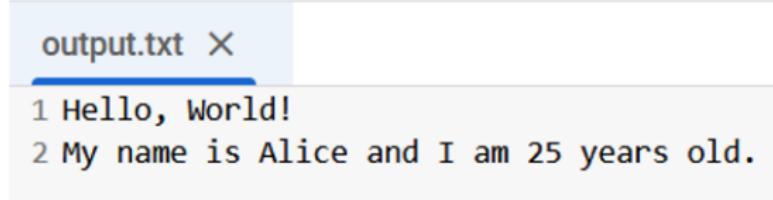
Output

-Printing to the Console: We've tried the `print()` function a lot.

-Writing to Files

- You can also write output to files using the `open()` function and the `write()` method.

```
#write to a file locally
with open("output.txt", "w") as file:
    file.write("Hello, World!")
    file.write("\nMy name is Alice and I am 25 years old.")
```



A screenshot of a code editor showing a file named "output.txt". The file contains two lines of text: "Hello, World!" and "My name is Alice and I am 25 years old.". The file is currently selected, indicated by a blue border around its tab.

```
output.txt ×
1 Hello, World!
2 My name is Alice and I am 25 years old.
```

Redirecting Output

You can redirect Python's output to other destinations, such as a file or the system console, using the `sys` module.

```
#redirecting output
import sys

# Redirect output to a file
sys.stdout = open("output2.txt", "w")
print("This output2 will be written to the file.")
sys.stdout.close()

# Restore output to the console
sys.stdout = sys.__stdout__
print("This output2 will be displayed in the console.")
```

output2.txt X

```
1 This output2 will be written to the file.
2
```

Exercise 1

Task:

Step	Description
1	Ask the user to input the width and height of the rectangle
2	Calculate the perimeter of the rectangle using the following formula: <i>$\text{Perimeter of the Rectangle} = 2 \times (\text{Width} + \text{Height})$</i>
3	Display the perimeter of the rectangle to the user

Exercise 1: suggested solution

```
▶ #suggested solution  
#display required information  
  
print("===== Perimeter of the Rectangle =====")  
  
#ask user to input the width and length  
  
width = float(input("Enter the Width: "))  
  
height = float(input("Enter the Height: "))  
  
#calculate the perimeter  
  
perimeter = 2 * (width + height)  
  
#display the perimeter as the result  
  
print("Perimeter of the Rectangle: " + str(perimeter))
```

```
→ ===== Perimeter of the Rectangle =====  
Enter the Width: 5  
Enter the Height: 2  
Perimeter of the Rectangle: 14.0
```

Homework 1

Task:

Step	Description
1	Import the math package
2	Ask the user to input the radius of the circle
3	Calculate the area of the circle using the following formula: Area of the Circle = pi x Radius x Radius
4	Display the area of the circle to the user

Basic Syntax: if-else condition

The **if-else** condition in Python allows you to execute different blocks of code based on a certain condition.



```
# if-else condition
x = 10
if x > 0:
    print("x is positive")
else:
    print("x is negative or zero")
```



x is positive

Basic Syntax: nested if statements

The **nested if statements** are when you have an **if** statement inside another **if** statement. This allows you to check multiple conditions in a more complex way.



```
#nested if-else
data = 10
if data >= 20:
    print("Equal to or greater than 20")
elif data >= 10:
    if (data == 10):
        print("Equal to 10")
else:
    print("Smaller than 0")
```



Equal to 10

Exercise 2

Task:

Step	Description
1	Ask a user to input the scores of different subjects
2	Sum the scores of three different subjects
3	Check whether the sum of scores is greater than 150
4	Display corresponding message according to the results

Exercise 2: suggested solution

```
▶ #suggested solution
#ask user to input the scores of different subjects
chinScore = float(input("Chinese Score: "))
engScore = float(input("English Score: "))
mathScore = float(input("Maths Score: "))

#sum the scores of different subjects
totalScore = chinScore + engScore + mathScore
print("Total Score: " + str(totalScore))

#check whether the total score is greater than 150
if totalScore > 150:
    print("You passed the exam.")
else:
    print("You failed the exam.")
```

```
→ Chinese Score: 25
English Score: 65
Maths Score: 8
Total Score: 98.0
You failed the exam.
```

Homework 2

Task:

Step	Description
1	Generate a random number
2	Ask the user to input the guessing number until the user guessed the random number correctly or it reached the guessing limit
3	Check whether the user guessed the random number correctly
4	Display the corresponding messages to show the guessing results

Hint: generate random number in Python

Basic Syntax: loops

Loops⁴ in Python allow you to execute a block of code repeatedly until a certain condition is met.

- **for**: loop iterates over a sequence (such as a list, tuple, or range) and executes the block of code for each item in the sequence.
- **While**: loop executes the block of code as long as a certain condition is true. In this example, the loop continues until the value of x is no longer less than 5.

for loop
for i in range(5):
 print(i)

0
1
2
3
4

while loop
x = 0
while x < 5:
 print(x)
 x += 1

0
1
2
3
4

⁴Extended learning for **nested loop, break, continue, pass**

Exercise 3

Task:

Step	Description
1	Generate a list of numbers
2	Prints each number in the list
3	Calculates the sum of all the numbers in the list
4	Finds the largest and smallest numbers in the list

Exercise 3: suggested solution

```
▶ numbers = [15, 32, 8, 45, 22, 19, 11, 3, 87, 6]

# 1. Print each number in the list
print("Printing each number in the list:")
for num in numbers:
    print(num)

# 2. Calculate the sum of all the numbers in the list
print("\nCalculating the sum of all the numbers in the list:")
total_sum = 0
for num in numbers:
    total_sum += num
print(f"The sum of all the numbers is: {total_sum}")

# 3. Find the largest and smallest numbers in the list
print("\nFinding the largest and smallest numbers in the list:")
largest = numbers[0]
smallest = numbers[0]
for num in numbers:
    if num > largest:
        largest = num
    if num < smallest:
        smallest = num
print(f"The largest number is: {largest}")
print(f"The smallest number is: {smallest}")
```



Also try user-defined functions.

Homework 3

Task:

Write a Python script that prints a pyramid pattern using the hash symbol (#) and spaces. The pattern should be 5 rows tall, like this:



Python OOPs Concepts

- Class
- Objects
- Polymorphism
- Encapsulation
- Inheritance
- Data Abstraction

By providing such features, OOP languages offer significant advantages over non-OOP approaches, especially when it comes to developing **complex, scalable, and maintainable software systems.**

Other important techniques: e.g. [Debugging in Google Colab](#)

Basic Data Analysis and Visualization

Data Frame

A DataFrame is a fundamental data structure in the [Pandas](#) library, which is a widely-used Python library for data manipulation and analysis. A DataFrame is a two-dimensional, tabular data structure, similar to a spreadsheet, where data is organized into rows and columns.

```
#data frame
import pandas as pd

# Create a DataFrame
data = {
    'Name': ['John', 'Sarah', 'Michael', 'Emily'],
    'Age': [35, 28, 42, 31],
    'City': ['New York', 'San Francisco', 'Chicago', 'Seattle']
}

df = pd.DataFrame(data)
print(df)
```

	Name	Age	City
0	John	35	New York
1	Sarah	28	San Francisco
2	Michael	42	Chicago
3	Emily	31	Seattle

Loading data frame in csv

The `pandas.read_csv()` function is a [Pandas](#) function used to read a CSV (Comma-Separated Values) file into a DataFrame.

```
[42] #load a data frame in csv
     import pandas as pd

     # Read a CSV file
     df = pd.read_csv('production.csv')

     # Display the first 5 rows
     print(df.head())
```

	Year	Vehicles
0	2017	103.10
1	2018	244.90
2	2019	367.20
3	2020	499.00
4	2021	935.95

Check the codes in [Python_demon.ipynb](#) file:

- Viewing Data
- Selecting Data
- Filtering Data
- Adding and Modifying Data
- Handling Missing Data
- Sorting Data
- Saving and Loading Data

Data Visualization

[Matplotlib](#) is a comprehensive library for creating static, animated, and interactive visualizations in Python. It is widely used for data visualization and is an essential tool in the data analysis workflow. Check the codes in [Python_demon.ipynb](#) file:

- Simple line
- Customized plot
- barplot
- scatterplot
- histogram
- subplots
- multiple plots

Exercise 4

By utilizing both [Pandas](#) and [Matplotlib](#) library, write a program to read a CSV file and plot it as a **scatter plot** with star marker (*) in [green](#) color and size **60**. You are given the recent seven years (2017-2023) of Tesla production figures in [production.csv](#). It plots the ‘**Vehicles**’ data as a scatter chart. The plot title should be “**Scatter chart of Tesla vehicles production**”. The vertical axis label should be “**Production (in '000s)**”. The horizontal label is “**Year**”.

	Year	Vehicles
0	2017	103.10
1	2018	244.90
2	2019	367.20
3	2020	499.00
4	2021	935.95

The production.csv data

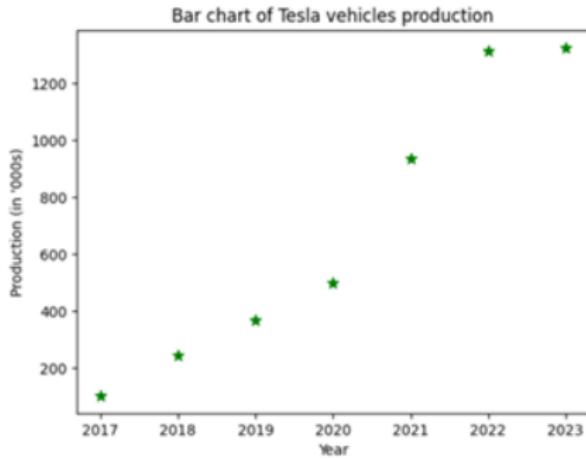
Exercise 4: suggested solution

```
#suggested solution

# load a data frame in csv
import matplotlib.pyplot as plt
import pandas as pd

# Read the CSV (sales.csv) file
df = pd.read_csv('production.csv')
# Plot the scatter diagram

plt.scatter(x=df.Year, y=df.Vehicles, color= "green",
            marker= "**", s=60)
# Plot setting
plt.title('Bar chart of Tesla vehicles production')
plt.xlabel('Year')
plt.ylabel("Production (in '000s)")
# Show the plot
plt.show()
```



Homework 4: Data Analysis

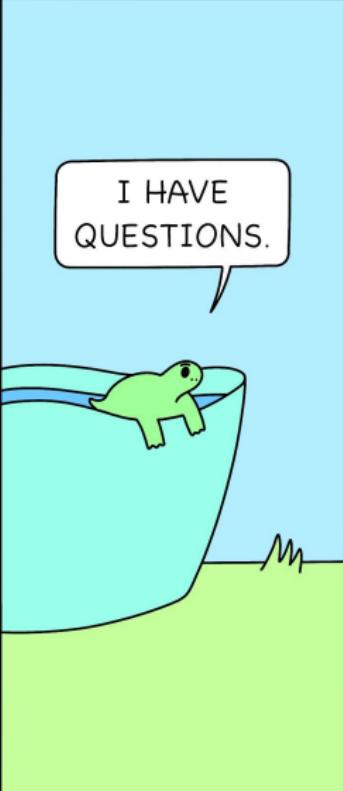
- **DataFrame Creation**
 - Create a DataFrame from a dictionary.
 - Add and modify columns in the DataFrame.
- **Data Selection**
 - Select specific rows and columns from a DataFrame.
 - Filter the DataFrame based on conditions.
- **Descriptive Statistics**
 - Calculate summary statistics (mean, median, standard deviation) for a DataFrame.
 - Group the data by a specific column and calculate aggregate statistics.

Homework 5: Data Visualization

- **Bar Plot**
 - Create a bar plot to compare the sales of different products.
 - Add error bars to represent the uncertainty in sales data.
- **Histogram**
 - Generate a histogram of a dataset following a normal distribution.
 - Customize the number of bins and the appearance of the histogram.
- **Subplots**
 - Create a figure with multiple subplots (e.g., 2x2 grid).
 - Plot different types of charts (line, scatter, bar, histogram) in each subplot.

- Time Series Analysis
- Exploratory Data Analysis
- Correlation Analysis
- Social Network Analysis
- Principal Component Analysis
- Machine Learning
- Natural Language Processing

Any Questions?



I HAVE
QUESTIONS.



I HAVE
ANSWERS.



NOT
NECESSARILY
TO THE SAME
QUESTIONS.