Python Course

Week 2 – Functions, Classes & Essential Tools

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

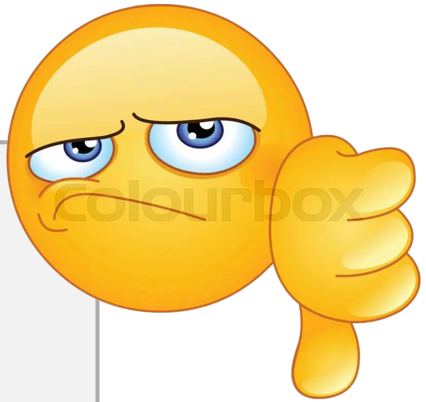
Week 2 – Functions, Classes & Essential Tools

Functions - Why do you need them?

What if you wanted to print ‘Hello, World!’ 3 times?

```python
print("Hello, World!")
print("Hello, World!")
print("Hello, World!")
```

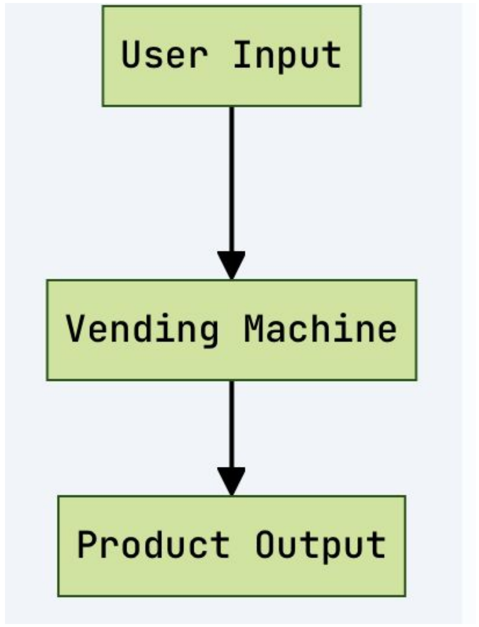
```
Hello, World!
Hello, World!
Hello, World!
```



```python
for i in range(3):
    print('Hello, World!')

```

```
Hello, World!
Hello, World!
Hello, World!
```

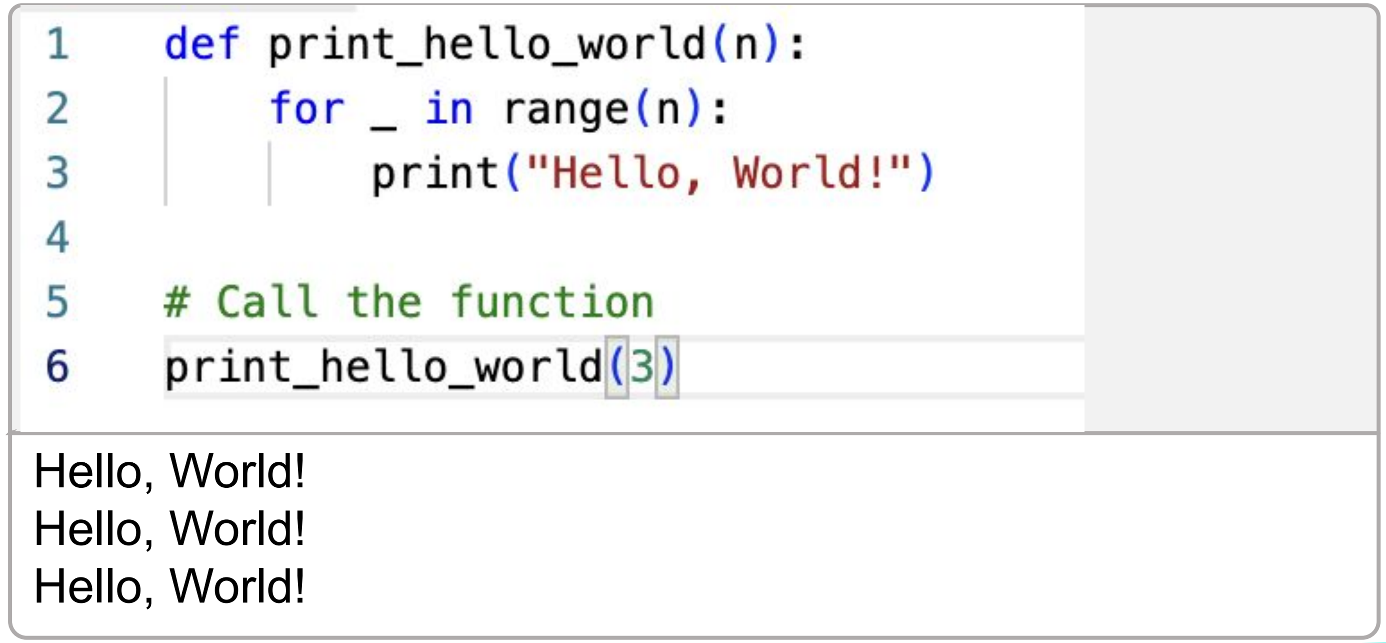
Functions - how do they work?



Functions

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



```python
def print_hello_world(n):
    for _ in range(n):
        print("Hello, World!")

# Call the function
print_hello_world(3)
```

```
Hello, World!
Hello, World!
Hello, World!
```

Functions

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

```python
def greet():
    print("Hello, World!")
# We can call the function like this:
greet()
```

```
Hello, World!
```

Functions

Functions can take an argument (n) to feed into the code and return values

```
def greet_person(name):
        print(f"Hello, {name}!")
# We can call the function like this:
greet_person(”Alice")
```

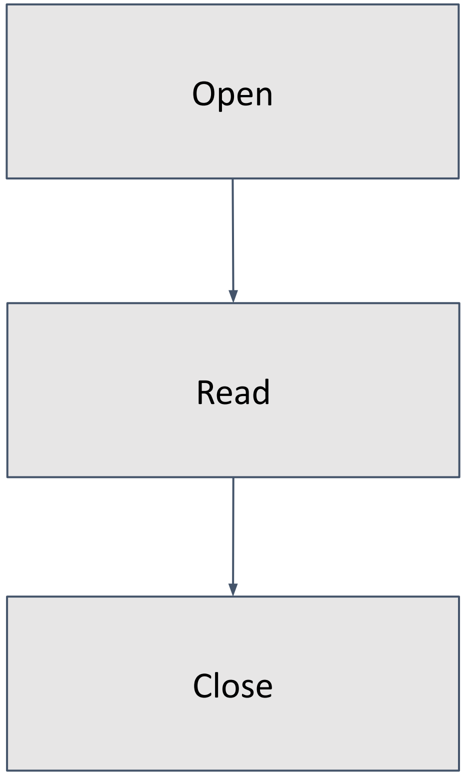
```
Hello, Alice!
```

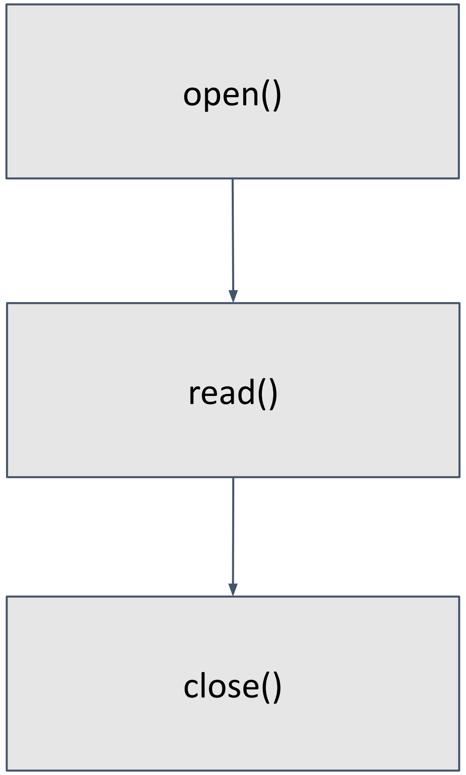
```
def add_numbers(num1, num2):
        sum = num1 + num2
        return sum

# Call this function and store its return value in a variable:
result = add_numbers(2, 3)
print(result) # prints: 5
```

```
5
```

File handling

File reading





File handling

File reading

To read a file in Python, you need to open it using the open function and then read its contents.

read()

Make sure to open and close your file to read it!

open(), close()

```python
# Reading an entire file at once
file = open("example.txt", "r")
content = file.read()
print(content)
file.close()
```

```
Hello, World!
```

File handling

File writing

Writing to a file is similar to reading it. Instead of using the read() method, we use:

write()

```python
# Reading an entire file at once
file = open("example.txt", ”w")
file.write(“Hello, World!”)
file.close()
```

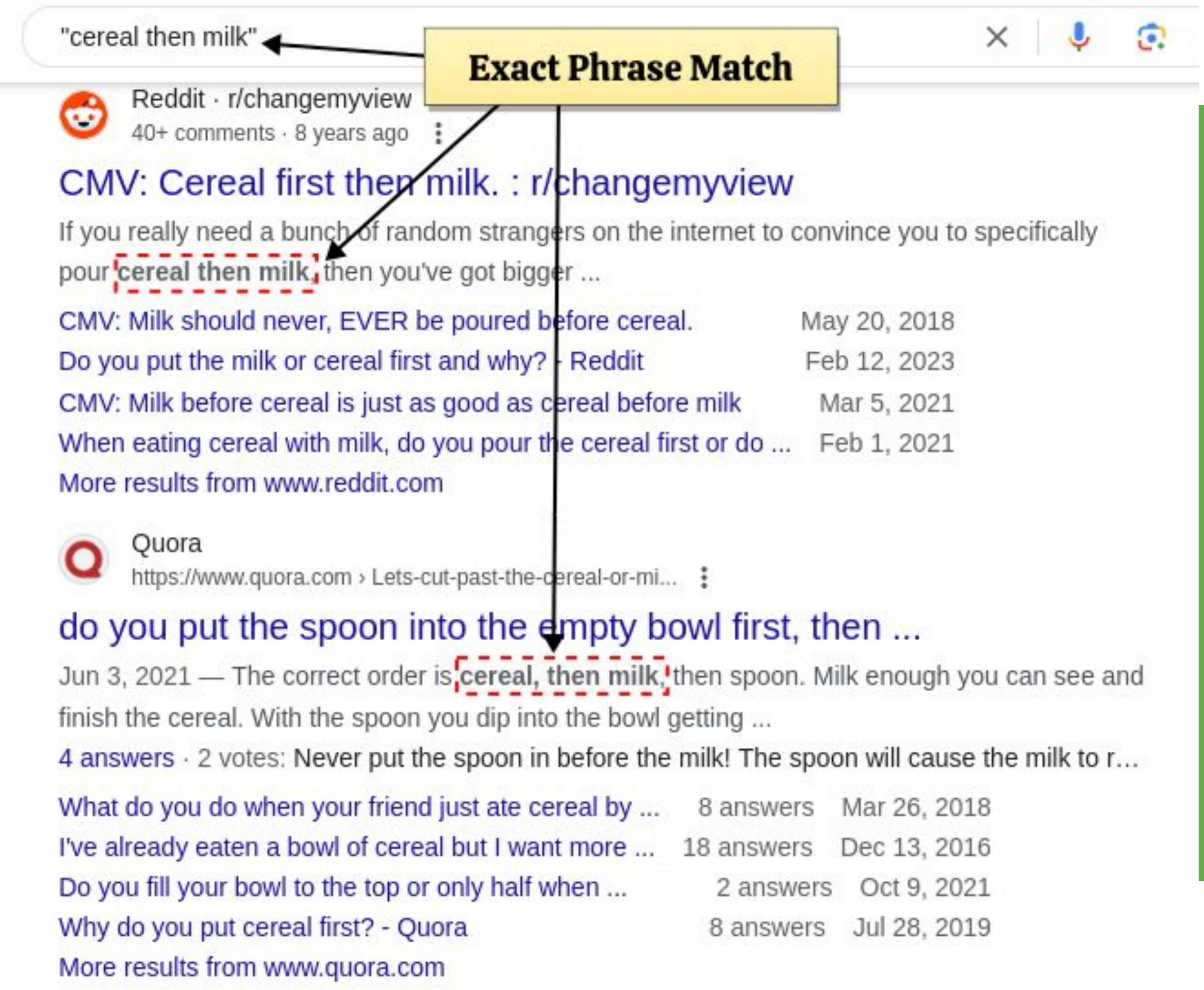
```
Hello, World!
```

File handling

Exercises

(1) Read the contents of a file named exercise1.txt and print it to the console.
(2) Write the string "Python is awesome!" to a file named exercise2.txt.

Searching Text



"cereal then milk"

Exact Phrase Match

Reddit · r/changemyview
40+ comments · 8 years ago

CMV: Cereal first then milk. : r/changemyview

If you really need a bunch of random strangers on the internet to convince you to specifically pour cereal then milk, then you've got bigger ...

CMV: Milk should never, EVER be poured before cereal.	May 20, 2018
Do you put the milk or cereal first and why? - Reddit	Feb 12, 2023
CMV: Milk before cereal is just as good as cereal before milk	Mar 5, 2021
When eating cereal with milk, do you pour the cereal first or do ...	Feb 1, 2021

More results from www.reddit.com

Quora
https://www.quora.com › Lets-cut-past-the-cereal-or-mi...

do you put the spoon into the empty bowl first, then ...

Jun 3, 2021 — The correct order is cereal, then milk, then spoon. Milk enough you can see and finish the cereal. With the spoon you dip into the bowl getting ...

4 answers · 2 votes: Never put the spoon in before the milk! The spoon will cause the milk to r...

What do you do when your friend just ate cereal by ...	8 answers	Mar 26, 2018
I've already eaten a bowl of cereal but I want more ...	18 answers	Dec 13, 2016
Do you fill your bowl to the top or only half when ...	2 answers	Oct 9, 2021
Why do you put cereal first? - Quora	8 answers	Jul 28, 2019

More results from www.quora.com

What is RegEx?

THIS: `str.match(/\d+\.\d+|\d+|[-+*/\(\)]/g);`

whaaaaat?????

Regular expressions

Regex

A sequence of characters that forms a search pattern. For example, you might use the regex **\d{3}-\d{2}-\d{4}** to search for occurrences of a pattern in the format "123-45-6789" within a text file.

Python has a built-in package called **re** that can be used to work with regular expressions.

```python
import re
```

(1) **re.findall()**: Returns all non-overlapping matches in a string as a list.
(2) **re.search()**: Searches for a match and returns a match object if found.
(3) **re.split()**: Splits the source string by the occurrences of the pattern
(4) **re.sub()**: Replaces occurrences of the pattern with another string

Regular expressions

Example

Here we have an abstract of a medical paper, let's try and extract some useful information.

```python
import re

with open('medical_paper.txt', 'r') as file:
	data = file.read()

print(data)
```

Regular expressions

Example

Here we have an abstract of a medical paper, let's try and extract some useful information.

```python
# Find All UK Phone Numbers in a Text File
# The phone numbers are in the format +44 00 0000 0000.

pattern = r"\+44 \d{2} \d{4} \d{4}"
result = re.findall(pattern, data)
print("Found Phone Numbers:", result)
```

Regular expressions

Cleaning data

Real-world data often contains imperfections, including spelling errors, which can significantly impact data analysis and outcomes. An example of such an error is the misspelling of the word "Aspirin" as "Aspiriin" in the text file 'medical_paper.txt'.

```python
# Replace All Occurrences of a Specific Word
with open('medical_paper.txt', 'r') as file:
        data = file.read()

new_data = re.sub(r'\bAspiriin\b', 'Aspirin', data)
print(new_data)

# Writing the modified content back to the file
with open('medical_paper_corrected.txt', 'w') as file:
        file.write(new_data)
```

Artist
Cart
Smart
Art

Regular expressions

Finding data

Regex can also be used to find certain words, such as capitalizations.

```python
# Find All Capitalised Words
pattern = r"\b[A-Z][a-z]*\b"
result = re.findall(pattern, new_data)
print("Found Capitalised Words:", result)
```

```
Found Capitalised Words: ['Title', 'The', 'Effects', 'Aspirin', 'Ibuprofen',
'Cardiovascular', 'Health', 'Authors', 'Dr', 'John', 'Doe', 'Phone', 'Email',
'Dr', 'Jane', 'Smith', 'Phone', 'Email', 'Dr', 'Emily', 'Brown', 'Phone', 'Email',
'Abstract', 'This', 'Aspirin', 'Ibuprofen', 'Keywords', 'Aspirin', 'Ibuprofen',
'Cardiovascular', 'Health', 'Introduction', 'Recent', 'Aspirin', 'Ibuprofen',
'This', 'Methods', 'We', 'Participants', 'Aspirin', 'Ibuprofen', 'Results', 'Both',
'Aspirin', 'Ibuprofen', 'However', 'Aspirin', 'Conclusions', 'While', 'Aspirin',
'Ibuprofen', 'Acknowledgements', 'We', 'Hospital', 'Contact', 'For', 'Dr',
'John', 'Doe']
```

Regular expressions

Removing duplicates

Remember that sets removed duplicates!

```python
# Find All Capitalised Words
pattern = r"\b[A-Z][a-z]*\b"
result = re.findall(pattern, new_data)
set(result)
```

```
{'Abstract', 'Acknowledgements', 'Aspirin', 'Authors', 'Both', 'Brown',
'Cardiovascular', 'Conclusions', 'Contact', 'Doe', 'Dr', 'Effects', 'Email',
'Emily', 'For', 'Health', 'Hospital', 'However', 'Ibuprofen', 'Introduction',
'Jane', 'John', 'Keywords', 'Methods', 'Participants', 'Phone', 'Recent',
'Results', 'Smith', 'The', 'This', 'Title’, 'We', 'While'}
```

Regular expressions

Regex Syntax

Syntax	Description	Example	Matches
.	Any character (except newline)	..	ab, 9!, etc
^	Start of string	^abc	abcde
$	End of string	xyz$	uvwxyz
*	0 or more repetitions of preceding element	ab*c	ac, abc, abbc
+	1 or more repetitions of preceding element	ab+c	abc, abbc
?	0 or 1 repetition of preceding element	ab?c	ac, abc
{m}	Exactly m repetitions of preceding element	a{3}	aaa
{m,n}	From m to n repetitions of preceding element	a{1,3}	a, aa, aaa
[]	Character set	[abc]	a, b, c
‘	Either/or	`a b’	a, b

Regular expressions

Regex Syntax

Syntax	Description	Example	Matches
()	Grouping	(abc){2}	abcabc
\	Escape special characters	\\.	.
\d	Any digit	\d{2}	09, 11
\b	Empty string at beginning/end of word	\babc\b	abc
\w	Any alphanumeric character	\w+	abc123

Regular expressions

Exercises

(1) Find all email addresses in the file 'medical_paper_corrected.txt'. Assume that the email addresses have the pattern "name@domain.com".

(2) Write a Python script to replace all occurrences of a date in the format dd/mm/yyyy to mm-dd-yyyy in a text file.

Regular expressions

Exercises

(3) Read the medical paper in the file 'medical_paper_corrected.txt' and find all mentions of specific medications (e.g., "Aspirin", "Ibuprofen", etc.). List out the medications found, along with their counts (number of times that medication appeared in the text).

(4) Create a function that takes in the text file and a regex pattern as input parameters. The function should return a list of all matches. Demonstrate the function using 'medical_paper_corrected.txt' as the file and a regex expression of your choice for this exercise.

Error handling

When a Python script encounters an error, it stops and generates an error message. The try-except block allows us to catch exceptions and execute specific code when an error occurs.

```
try:

except:
```

Error handling

When a Python script encounters an error, it stops and generates an error message. The try-except block allows us to catch exceptions and execute specific code when an error occurs.

```python
y = 'hello’
try:
	print(y)
except NameError:
	print("Variable y is not defined")
```

```
hello
```

Error handling

```
try:
        print(x_123)
except NameError:
        print("Variable x_123 is not defined")
```

```
Variable x_123 is not defined
```

```
 # Handling multiple exceptions
try:
        # some code that may cause an error
        pass
except (ZeroDivisionError, FileNotFoundError):
        print("Either tried to divide by zero or file was not found.")
```

```
hello
```

Error handling

Exercises

(1) Create a function called safe_divide that takes two arguments, a and b, and returns the result of a / b. If the division is not possible, the function should return None.

(2) Write a function that takes a string as input and returns the string reversed.

(3) Write a function that takes two numbers as input and returns the larger number.

(4) Write a function that takes a list of numbers as input and returns the average of the numbers.

(5) Write a function that calculates BMI with inputs of weight in kg and height in metres.

Classes

A class serves as a blueprint for creating objects. Objects themselves are individual instances of a class and contain both data (with its various attributes) and functionalities (through its methods).

Data (Attributes)

Attributes are variables that belong to the class. They hold information that is relevant to the object. For example, in a **Patient** class, attributes could include **name**, **age**, and **medical_history**.

Actions (Methods)

Methods are functions that belong to the class and can be called on any object of that class. They usually manipulate the attributes or perform specific tasks. In a **Patient** class, methods could include **add_diagnosis()** or **schedule_appointment()**.

Classes

Object Oriented Programming (OOP)

A programming paradigm based on the concept of "objects," which represent both data and the functionalities to manipulate that data. In OOP, code is organised into reusable "classes" and "objects" to make it more modular, maintainable, and intuitive to understand.

(1) Encapsulation: The bundling of data (attributes) and methods (functionalities) into a single unit (object).

(2) Inheritance: The mechanism by which one class can inherit the attributes and methods from another class.

(3) Polymorphism: The ability to present the same interface for different data types.

(4) Abstraction: Hiding complex reality while exposing only the necessary parts.

Classes

Procedural programming

An alternative to Procedural Programming. Procedural Programming is focused on just writing functions to operate on data.

	Procedural	OOP
Advantages	Easier to understand and write, especially for small tasks. May execute faster due to lower overhead.	Better modularity makes it easier to manage complex systems.
Disadvantages	Harder to scale and maintain as projects grow. Limited code reuse capabilities.	Inheritance allows for significant code reuse. Can be complex for simple projects. Potential overhead of method calls and attribute look-ups.

Classes

When to use which

(1) Small, Simple Projects: Procedural programming is often sufficient.
(2) Large, Complex Projects: OOP offers greater maintainability and is more scalable.
(3) Team-based Development: OOP is generally more structured and easier for multiple developers to collaborate on.

Classes

Example

```python
# Basic Structure of a class
class MyClass:
    # Class attribute
    my_attribute = "This is a class attribute"

    # Instance method
        def my_method(self):
        print("This is a method of the class")
```

Classes

Example

```
# Creating an object of MyClass
my_object = MyClass()

# Accessing attribute of class
print(my_object.my_attribute)
```

```
This is a class attribute
```

```
# Calling method of class
my_object.my_method()
```

```
This is a method of the class
```

Classes

New instances

Initialises (assign values to) any object's properties or perform operations that are necessary to do when the object is being created (for example, generating a unique id).

__init__

We use the **self** parameter to denote the first parameter in instance methods (functions) of a class. It's convention to call it **self**, but a

Classes

New instances

```python
class Person:
        def __init__(self, name, age):
                self.name = name
                self.age = age

        def greet(self):
                print(f"Hello, my name is {self.name} and I am {self.age} years old.")

# Creating an object of Person
alice = Person("Alice", 25)
alice.greet()
```

```
Hello, my name is Alice and I am 25 years old.
```

Classes

Exercises

(1) Create a class "Dog" with an attribute "name" and a method "bark" which prints "Woof!”

(2) Create a class "Rectangle" with attributes "width" and "height" and a method "area" which returns the area of the rectangle.

(3) Create a class "Circle" with an attribute "radius" and two methods "area" and "circumference" which return the area and the circumference of the circle, respectively.

Classes

Exercise 4 – Store patient medical history

(1) Build a simple Python class to store a patient's medical history, including their name, diagnoses, treatments received, and dates of treatments or diagnoses.
(2) You should also be able to add new diagnoses, new treatments, dates for these treatments or diagnoses, and display the entire medical history for a patient.

Your final class should allow a patient named John with past history of Hypertension and Type 2 Diabetes to be created.

He has received Antihypertensive medication and Metformin for diabetes on dates '10/10/2021' and '11/11/2022', respectively. Initalise the class with this information.
Finally, with today's date, add a diagnosis of Flu and add an appropriate treatment, then display the full medical history with this new information.

Random module

The random module in Python is used to generate pseudo-random numbers for various probabilistic distributions. It is an extremely useful module for tasks such as simulating events, generating random data, and for games and gambling mechanics.

The numbers generated with random are not truly random; they are deterministic, but are usually sufficient for most cases where unpredictable results are desired.

random.

Random module

Random functions

(1) Random float: **random.random()** returns a number between 0.0 and 1.0
(2) Random integer: **random.randint(x,y)** returns a number between x and y
(3) Random elements: **random.choice(list)** returns a random list element
(4) Random shuffle: **random.shuffle(list)** shuffles the list elements

Random module

Exercise 1 – Dice simulator

Write a program that simulates the rolling of a fair 6 sided dice. Roll the dice 10 times.

Hint: Use random.randint() to generate a number between 1 and 6. Alternatively, use random.choices()

Random module

Exercise 2 – Dice simulator

Write a program that simulates the rolling of a weighted 6 sided dice that is more likely to land on the number 2.

Hint: use random.choices() and input a weights parameter.

The weights parameter itself is a list and contains weight values that correspond to each element of the list that contains items to be chosen from.

Random module

Exercise 3 – Random lottery picker

Simulate a lottery draw by picking 6 unique random numbers between 1 and 49. Use random.sample() to ensure the uniqueness of the numbers.

Random module

Exercise 4 – Random password generator

Create a program that generates a random password with 1 uppercase, 1 lowercase, 1 digit, and 1 special character.

Random module

Monty Hall Problem

(From the US tv show ‘Let’s make a deal’)

- **Three doors**: One has a prize, two are empty.
- The player **chooses a door**.
- The host **opens one of the other doors** that does NOT have the prize.
- The player is given a choice:
  - **Stick** with their original door.
  - **Switch** to the remaining unopened door.

Key Question:

- **Does switching or sticking give a better chance of winning?**

- **What do you think?**

Expected Result:

- **Sticking wins ~33% of the time**
- **Switching wins ~66% of the time**

👉 **Switching doubles the chances of winning!**

Random module

Exercise 5 – Monty Hall Problem simulator

(1) Create a program that randomly places a prize behind one of three doors, asks the user to choose a door, and then reveals one of the doors that does not contain the prize.

(2) The user then gets the option to stick with their original choice or switch to the other unopened door.

(3) Simulate this scenario 1000 times both for sticking and switching and calculate the win rate for each strategy.

Homework

Text-based game

In this text-based game, players battle with unique characters, each possessing distinct health, moves, and catch rates. The goals? Defeat opponents and capture characters by making smart moves and timing your captures perfectly.

Each character is defined with a specific set of attributes and abilities, as outlined in the Character class.

Homework

Starting code

```python
import random
class Character: # Define the character class
    def __init__(self, name, health, moves, catch_rate):
        self.name = name
        self.health = health
        self.moves = moves #dictionary of moves with their hit percentages
        self.catch_rate = catch_rate # Catch rate depends on rarity and health

    def attack(self, move, opponent): #
        if random.random() <= self.moves[move]:
            damage = random.randint(10, 20) # Random 10-20 damage
            opponent.health -= damage
            print(f"{self.name} used {move}. It's a hit! {opponent.name} lost
            {damage} health!")
        else:
            print(f"{self.name} used {move}. But it missed!")
```

Homework

Starting code

```python
def throw_ball(character):
    if random.random() <= character.catch_rate:
        print(f"Success! You caught {character.name}!")
    else:
        print(f"Oops! {character.name} escaped!")
```

Homework

Starting code

```
# Create characters
char1 = Character("Char1", 100, {"WeaponA": 0.85, "WeaponB":
0.95}, 0.3)
char2 = Character("Char2", 100, {"WeaponC": 0.85, "WeaponD":
0.95}, 0.3)

# Simulate Char1 attacking Char2
char1.attack("WeaponA", char2)

# Try to catch Char1
throw_ball(char2)
```

Homework

Exercises

(1) Add a turn-based system to the game, where the characters alternate turns for attacking.

(2) Expand the Character class with more properties, such as "type" (water, fire, grass, etc.) and "level". Adjust the attack method to take these new properties into account (e.g., water type moves do more damage to fire type characters).

(3) Add a "defense" property to the Character class. When a character is attacked, the damage is reduced by the character's defense.

Homework

Character Game

(4) Create more characters and add them to a list. The player can choose which character they want to use for a battle.

(5) Create a system where characters can gain experience and level up after winning a battle. When a character levels up, their health, attack, and defense increase.