IMPERIAL

BIOE50010 – Programming 2

Revision and Q&A

Binghuan Li | binghuan.li19@imperial.ac.uk

5 January, 2026

Exam

- Date: 9 January 2026
- Time: 10.00-12.00 (120 minutes)

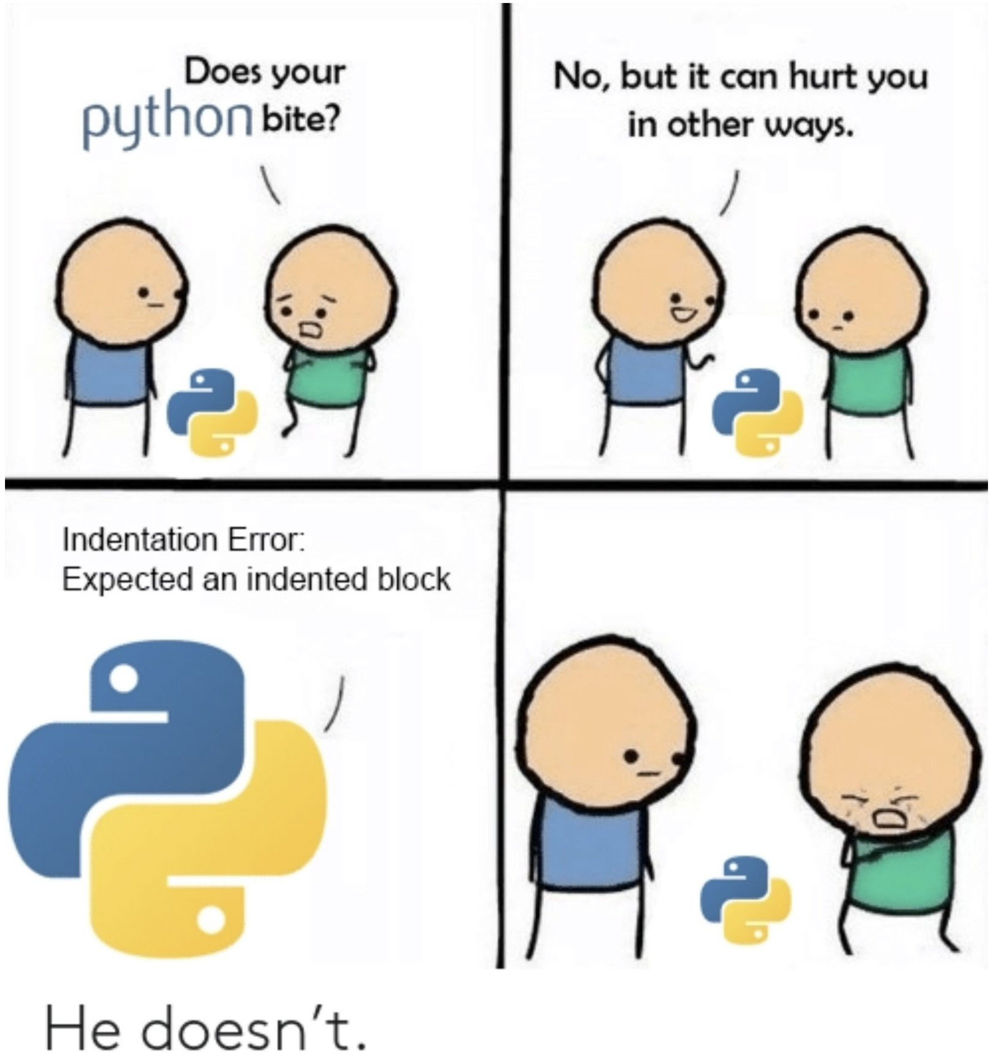
- 50% of final mark

- Exam paper format:
  - Total marks: 100
  - 4 compulsory questions
  - Each question is worth 25 marks

- Examination format:
  - In-person, in computer labs
  - w/o access to the internet
  - w/o AI tools
  - w/o access to lecture slides
  - w/ access to "`thinkpython2.pdf`"
  - w/ access to Python help() function

Tips for Revision

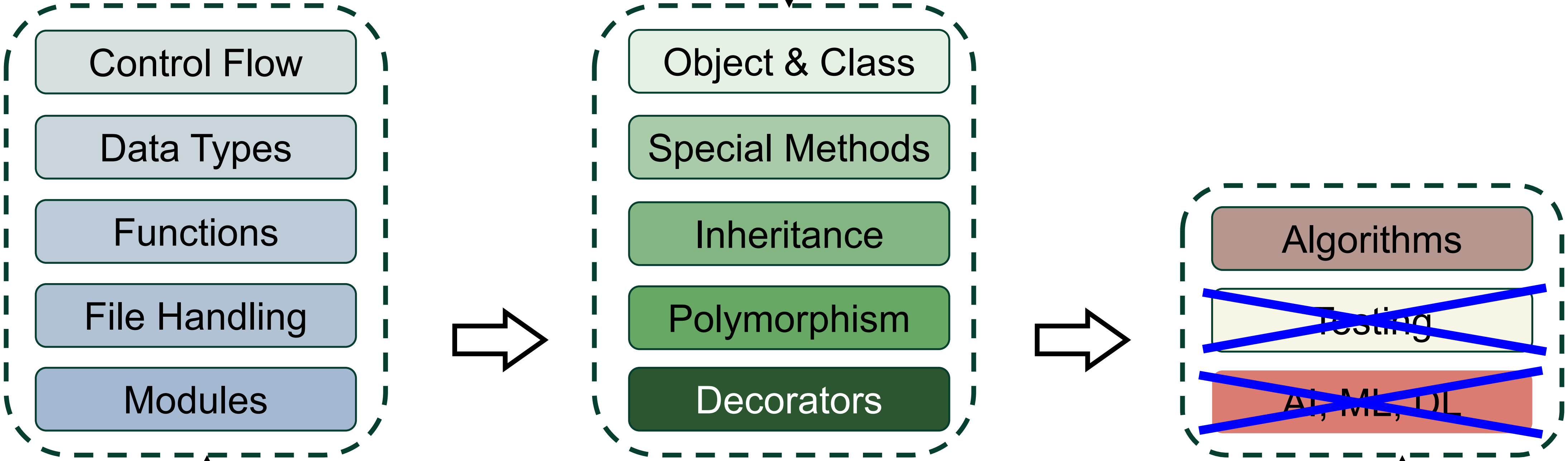
- Do not just look at code and say “oh, that makes sense”.
- Do write the code from a blank page.
- The syntax of every line of code must make sense to you.
- The algorithm that’s being implemented needs to make sense to you.
- Use questions from labs/exams to test yourself with limited/no access to the solution/internet/LLM.


Does your python bite?
No, but it can hurt you in other ways.
Indentation Error:
Expected an indented block
He doesn’t.

Programming 2

Focus:

- Design data structures using **class**es.
- Apply **inheritance** and polymorphism.
- Enhance designs with decorators and **special methods**.



Control Flow

Data Types

Functions

File Handling

Modules

Object & Class

Special Methods

Inheritance

Polymorphism

Decorators

Algorithms

~~Testing~~

~~AI, ML, DL~~

Focus:

- Grasp **Python syntax** and **logic**.
- Use **data types**, **control flow**, and **functions**.
- Apply **file handling** and **modules**.

Focus:

- Implement and test **algorithms**.
- ~~Evaluate code performance and reliability.~~
- ~~Recognize the role of AL/ML in research.~~

Control Flow

See:
Lab 1, Lab 3 (Task 3), Lab 5

- `if` and `while` must be followed by an expression whose value can be evaluated as a **Boolean** (`True` or `False`).

`if i == 1:`	Evaluate `i==1`, outcome is `True` or `False`
`if board.update_row_col():`	A function that returns `True` or `False`
`while game:`	A Boolean variable: `True` or `False`

- `break` (stop looping), `continue` (skip this round)

- `for` must be followed by an **iterable** object (*e.g.*, string, list, tuple, dictionary).

`for i in range(10, 0, -1):`	Decrement of an index using `range()`
`for idx, val in enumerate([2, 5, 8]):`	Loop over a `list` with index and value
`for key, value in my_dict.items():`	Loop over a `dict` with `key:value` pairs

Functions

See:
Lab 1, Lab 3 (Task 1,2), Lab 8 (Task 2)

Input(s) → Function → Output(s)

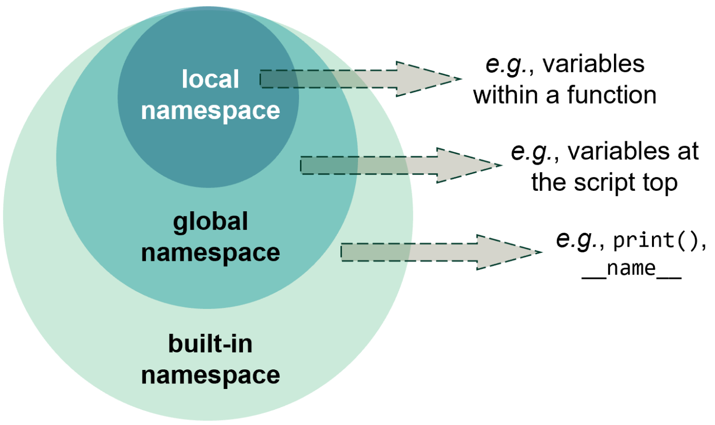
- Optional in function definition
- Pass variables into a function
- Pass functions into a function (wrapper)

- Function argument types:
  1) Positional argument
  2) Non-keyword argument (`*arg`)
  3) Keyword argument (`**kwarg`)

- Optional in function definition
- **Use `return` to output**

- When multiple `return` defined, the function terminates once the first `return` is triggered.

- Catch/omit the returned result in function call.


local namespace
e.g., variables within a function
global namespace
e.g., variables at the script top
built-in namespace
e.g., print(), __name__

- **Namespace** matters!
- Local variables: if you create a variable within a function, that variable *only* exists in that function.

Declare and Print a 2D List

See:
Lab 3 (Task 3), Lab 5, Lab 8 (Task 1)

Example: **declare** the 2-D nested list structure

```
N_ROW = 3
N_COL = 4
board = []
for i in range(N_ROW):
    r = []
    for j in range(N_COL):
        r.append(' . ')
    board.append(r)
```

1. **Create a row**: Start with an empty list and append the same element to it `N_COL` times. (imagine this is a row vector with N_{col} elements, represents N_{col} columns per row.)
2. **Build the board**: Append the initialized row to the main board structure for `N_ROW` times. (so, you obtain a $N_{row} \times N_{col}$ matrix.)

⇩

```
[[' . ', ' . ', ' . ', ' . '], [' . ', ' . ', ' . ', ' . '], [' . ', ' . ', ' . ', ' . ']]
```

⇩

Example: **print** the 2-D nested list structure

```
for i in range(N_ROW):
    print(f"{i:<5}", end='')
    for j in range(N_COL):
        print(board[i][j], end='')
    print()
```

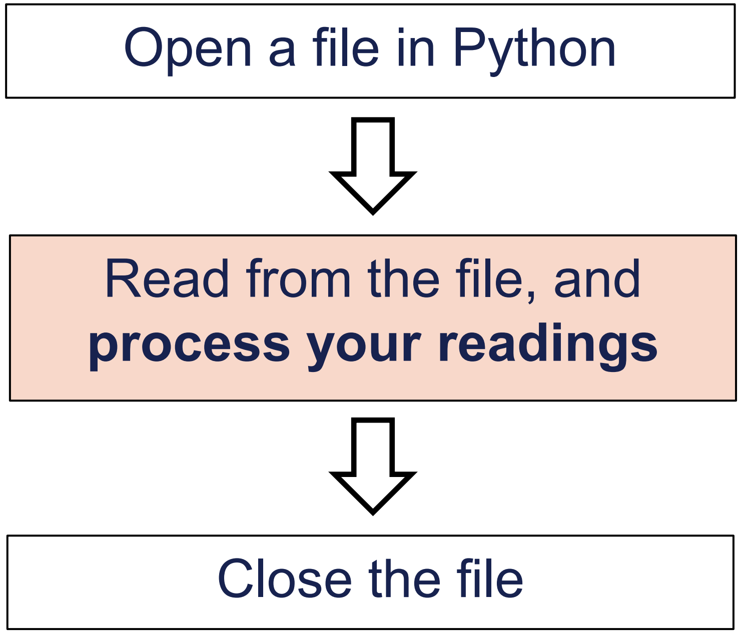
The key trick here is `"{i:<5}"`

- `:<` tells Python to **left-align** the text
- `5` is the **width** of the space allocated for the text.

`end=''` prevents line breaks introduced by `print()`

File I/O

See:
Lab 2 (Tasks 2.2, 3, 5), Lab 6



Open a file in Python

Read from the file, and **process your readings**

Close the file

- After reading data from a file, the contents are saved in a structure such as a **string** or a **list**.
- It is your task to **process the raw readings** before using them for further analysis: clean, transform, sort, organise…
- These operations can be done with **string and list methods**, *e.g.*,
  - Use `.split()` or `.strip()` to process text strings.
  - Use `.append()` or `.sort()` to manage lists.

- How to use a loop to read files line by line.
- If you are tasked to read a file (*e.g.*, `.txt`, `.csv`, `.fasta`, `.fna`), **always scrutinize the file contents first in a text file editor** (NOT Microsoft Excel).

Error Catching

See:
Lab 6

- In Python, exceptions can be handled using the **`try…except…`** clause:

Example: use of `try…except…` clause

```
while True:
    try:
        x = int(input("Please enter a number (1-9): "))
        break
    except ValueError:
        print("That was not valid number. Try again...")
```

`ValueError` raises due to the failure of typecasting, *e.g.*, `int("hello!")`

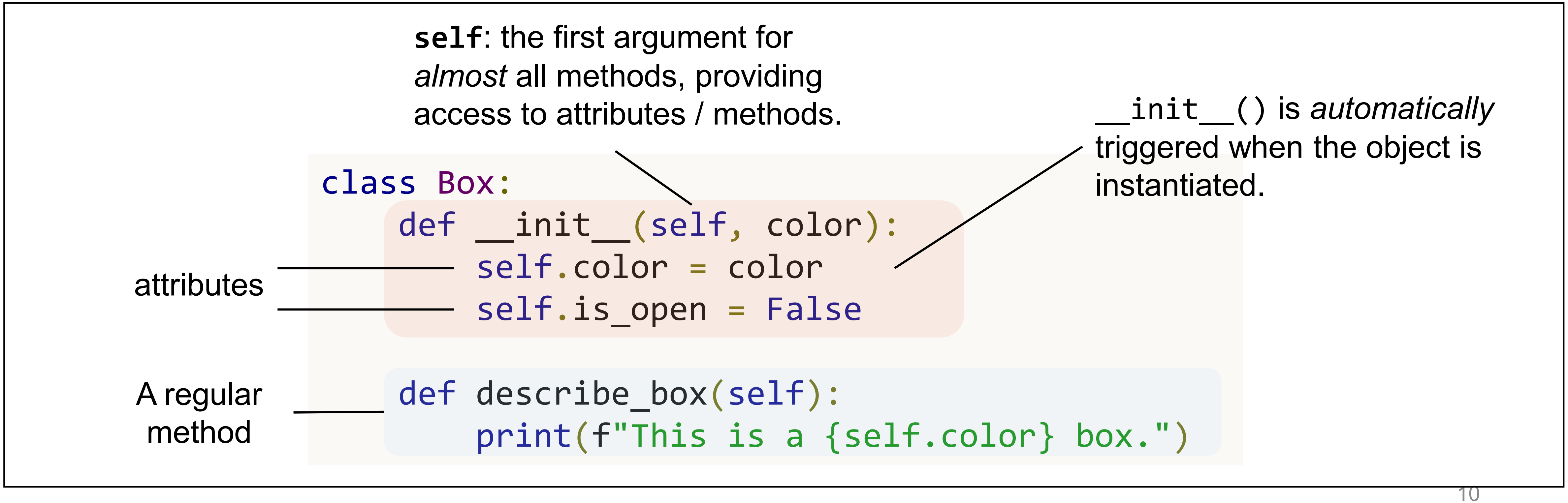
- Understand common errors types, *e.g.*,

`IndexError`	Accessing an out-of-range index in a **list** or **tuple**.
`NameError`	Using a **variable** that hasn’t been defined.
`TypeError`	Performing an operation on an inappropriate **data type**.
`ValueError`	Passing a valid type but **invalid value**.
`SyntaxError`	Code contains a **syntax** error.

Object-Oriented Programming

See:
Lab 4, Lab 5, Lab 6

- **Object-oriented**: programs based on the objects, where *data* (*attributes*) and *functions* (*methods*) are encapsulated into a user-defined data type.
- **Class**: The blueprint for creating an object. Does not contain actual data.
- **Objects** (instances): when actual data are sent into the class (instantiation).



`self`: the first argument for *almost* all methods, providing access to attributes / methods.

`__init__()` is *automatically* triggered when the object is instantiated.

attributes

A regular method

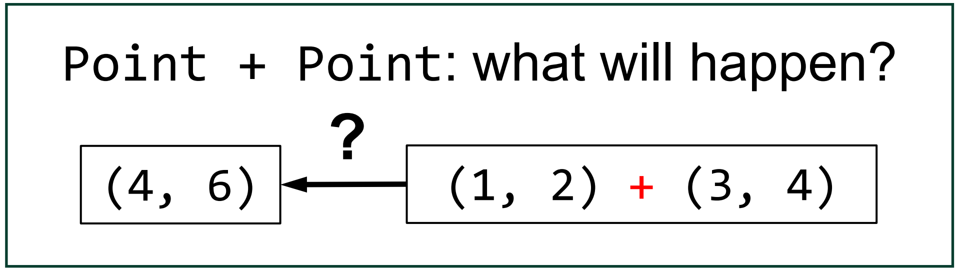
```python
class Box:
    def __init__(self, color):
        self.color = color
        self.is_open = False

    def describe_box(self):
        print(f"This is a {self.color} box.")
```

OOP: Special Methods

See:
Lab 4, Lab 6

- **Operator overloading** enables users to define the rules of an operator when it is applied to the user-defined data types. *e.g.*, **+**, **-**, *****, **==**, **<=**



`Point + Point`: what will happen?

(4, 6) ← **?** (1, 2) + (3, 4)

- In this situation, the rule(s) for '+' need to de defined with the special (magic) method **`__add__`** in `Point`

- **`pt + 2`** triggers `__add__()`, **`2 + pt`** triggers `__radd__()`.

- **Other magic methods** we have seen:

`__init__`, `__str__`, `__getitem__`, `__mul__`, `__rmul__`, `__sub__`, `__rsub__`.

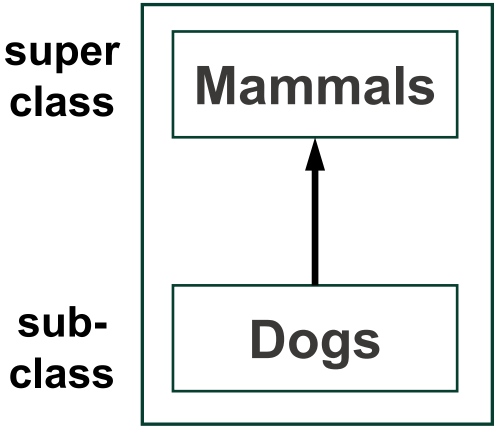
- **Functions associated with classes**:

`isinstance()`, `hasattr()`, `getattr()`

OOP: Inheritance

See:
Lab 5

- OOP allows a child class to inherit features from the parent class(es)



```
class Mammal:
    def __init__(self, name):
        self.name = name

    def warm_blooded(self):
        return f"{self.name} is warm-blooded."

    def speak(self):
        return "Grrrr!"

class Dog(Mammal):
    def __init__(self, name):
        super().__init__(name)

    def speak(self):
        return "Bark!"
```

speak() method in Dog overrides the speak() method in Mammal

warm_blooded() method in Dog is **inherited** from the Mammal class

Tips for Exam

1. **You MUST use Python IDLE 3.13.5 to answer questions.**
   - Make sure you are familiar with Python IDLE interface before exam.

2. **You will be supplied with a code skeleton to start your work.**
   - If you download a file, where is it? Is it named correctly (*e.g.*, `q1.py`, `q2input.txt`)?
   - Do **NOT** declare anything else in the global space.
   - If you want to test, you **MUST** use the `main` guard: `if __name__ == "__main__"`

3. **You MUST submit your work before the exam finishes; You will NOT be given extra time for code submission.**
   - Allocate your time wisely. Save your work frequently.
   - Submit as instructed strictly – submitted filenames **MUST** adhere to the instructions.

4. **Non-technical PC login failures will NOT be forgiven.**
   - Memorise your college login credentials (username, password). Test it beforehand!

The End.