GCU
Glasgow Caledonian University

Classes

Overview

- **Classes & Objects**
  - General ideas
- **Classes & Objects**
  - In code
  - ‘classes’, ‘objects’ ‘instances’
- **Python Classes**
  - `self`, `class`,
  - `__init__`,
  - `__str__`
- **class scope variables**
- **public v private methods**



GCU
Glasgow Caledonian University

Classes & Objects I

- Creating different objects with a common tool

Analogy

- **Pasty cutter**
  - single thing (various designs)
- **Pasty**
  - *resource* that can be cut by any pastry cutter
- **Together**
  - the pastry can be made into an object
  - the type of object depends on the pastry cutter
- **Re-use**
  - you re-use the cutter
  - you can also re-use the pastry



Analogy

- **Pasty cutter (Class)**
  - single thing (various designs)
  - one cutter can stamp out many objects
- **Pasty (Memory)**
  - *resource* that can be cut by any pastry cutter
- **Together**
  - the pastry can be made into an **object**
  - the type of **object** depends on the pastry cutter
- **Re-use**
  - you re-use the cutter
  - you can also re-use the pastry, once it is no longer needed



Classes & Objects

- **Class: like a template for…**
  - creating something (an ‘object’)
- **Object**
  - exists in memory, once created
  - created by the class
- **Example**
  - seahorse cutter
  - seahorse objects







Classes, Objects, Instances

- **'Class':**
  - creates objects
- **'Object':**
  - 'thing' created from a class
- **'Instance'**
  - an object seen in relation to it's class







classes v instances



class properties:

```
name
colour
```

classes used to create instances



instance

```
name: 'Mairi'
colour: 'red'
```



instance

```
name: 'Sunny'
colour: 'golden'
```

instances use-up RAM when created

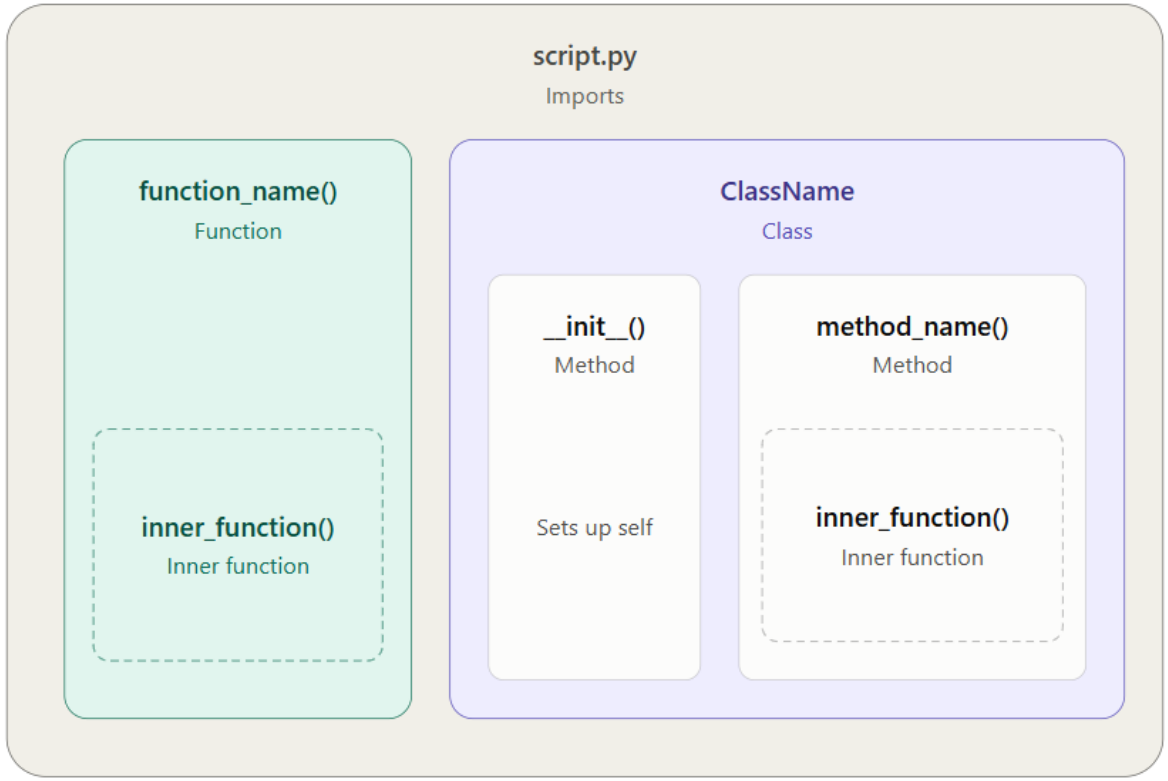
GCU
Glasgow Caledonian University

Classes & Objects II

- Creating different objects with a common tool (in code)

Classes (in a file)

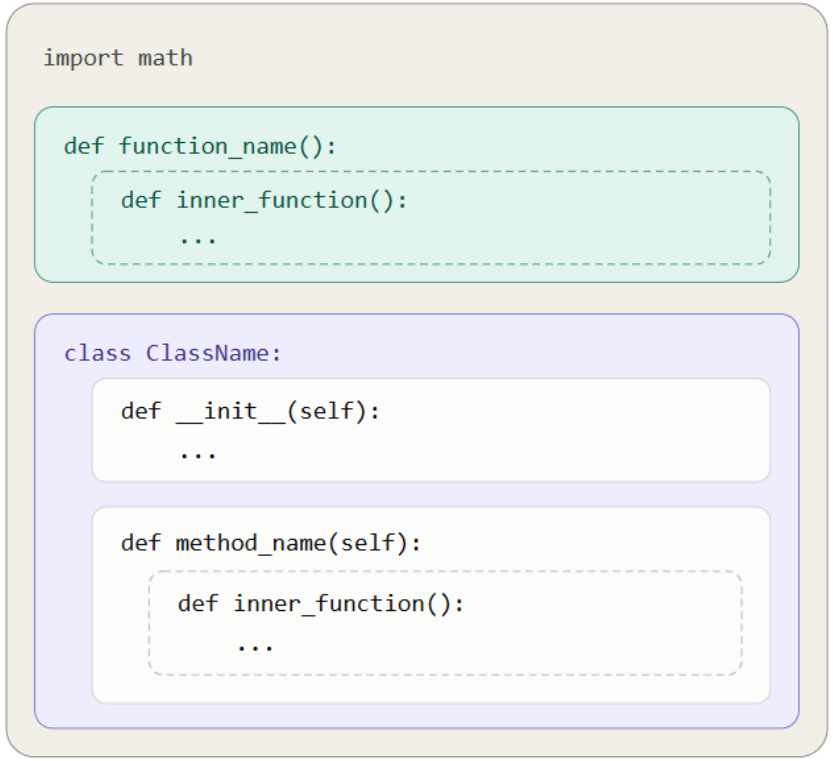
- **Files** **.py**
  - in a folder (maybe a package?)
- **Classes**
  - exist in files (maybe a module)
- **Methods**
  - exists in classes
- **Functions**
  - can exist inside functions
- **Variables**
  - can exists at all levels: file, class, function (class method), inner function



Dashed = only callable from inside the block around it

Classes (in a file)

- **Files** `.py`
  - in a folder (maybe a package?)
- **Classes**
  - exist in files (maybe a module)
- **Methods**
  - exists in classes
- **Functions**
  - can exist inside functions
- **Variables**
  - can exists at all levels: file, class, function (class method), inner function



Grey = script, teal = function, purple = class, white = method, dashed = inner function

Classes (example)

- **Classes**
  - exist in file
- **Methods**
  - exists in classes
- **`Seahorse`**
  - `species` is a class level variable
  - `name` is attached to the object

```python
class Seahorse:
    # Class variable — shared by all Seahorse objects
    species = "Hippocampus"

    def __init__(self, name, color):
        # Instance variables — unique to each Seahorse
        self.name = name
        self.color = color

    def __str__(self):
        # Defines how the object is displayed when printed
        return f"{self.name} is a {self.color} seahorse of species {Seahorse.species}."


def main():
    # example usage of
    s1 = Seahorse("Sunny", "golden")
    print(s1)


if __name__ == "__main__":
    main()
```

Classes (in a file)

- `__init__`
  - initialization
- `__str__`
  - string representation
- `self`
  - not a python keyword
  - conventional first-argument
  - refers to object method was called on

```python
class Seahorse:
    # Class variable — shared by all Seahorse objects
    species = "Hippocampus"

    def __init__(self, name, color):
        # Instance variables — unique to each Seahorse
        self.name = name
        self.color = color

    def __str__(self):
        # Defines how the object is displayed when printed
        return f"{self.name} is a {self.color} seahorse of species {Seahorse.species}."


def main():
    # example usage of
    s1 = Seahorse("Sunny", "golden")
    print(s1)


if __name__ == "__main__":
    main()
```

Naming conventions

🧱 Class Naming Convention

Rule	Example	Notes
✅ Use **PascalCase**	`class MyClass:`	Each word starts with a capital letter, no underscores.
🚫 Don't use snake_case	`class my_class:` ❌	That's for functions and variables.
🚫 Don't use ALL_CAPS	`class MYCLASS:` ❌	That's for constants.
✅ Keep names **clear and descriptive**	`class BankAccount:`	Should reflect what the class represents.
✅ Use **singular nouns**	`class Dog:` not `class Dogs:`	A class defines *one type* of object.

class v instance variables

- **Class variable**
  - defined in the class body
  - one copy, shared by every instance
- **Instance variables**
  - set on `self`, usually in `__init__`
  - each object has its own, seperate
- **Lookup order**
  - Python checks the object first, then the class
- **tip**
  - to change a class variable, use the class name

```
class Seahorse:
    species = "Hippocampus"   # class var

    def __init__(self, name):
        self.name = name      # instance var

mairi = Seahorse("Mairi")
sunny = Seahorse("Sunny")

Seahorse.species = "H. kuda"  # change class
print(mairi.species)          # H. kuda
print(sunny.species)          # H. kuda

sunny.species = "H. erectus"  # new instance var!
print(sunny.species)          # H. erectus
print(mairi.species)          # H. kuda
```

GCU
Glasgow Caledonian University

Private v Public Methods

- Creating different objects with a common tool (in code)

Public v private methods

- **Public methods**
  - plain name, e.g. `feed()`
  - the class's interface: meant to be called from outside
- **Private (by convention)**
  - leading underscore, e.g. `_digest()`
  - internal helper: “don't call this from outside”
- **Not enforced**
  - Python has no `private` keyword
  - `_name` still works, it's a signal to other programmers
- **tip**
  - keep the public interface small; hide the details

```
class Seahorse:
    def __init__(self, name):
        self.name = name
        self._hunger = 5        # internal

    def feed(self):             # public
        self._digest()
        return self.name + " is fed"

    def _digest(self):          # private
        self._hunger = 0

sunny = Seahorse("Sunny")
print(sunny.feed())     # Sunny is fed
sunny._digest()         # works... but don't!
```

Conclusions

- **Types**
  - we can create out own
  - with classes
- **‘classes’, ‘objects’ ‘instances’**
  - often-used terms in object orientated programming
- **Python Classes**
  - `self`, `class`,
  - `__init__`,
  - `__str__`
- **class scope**
  - variables
- **public v private methods**



GCU
Glasgow Caledonian University

Thanks!