Oriented Object Programming in Python

Olivier Mattelaer
UCLouvain
CP3 & CISM

Program of this lecture

- Programation Style
- Definition of many concept
  - ➡ Object, instance, class, attribute, …
- Dataclass
  - ➡ Equivalent of Fortran Structure
- Class
- Inheritance
- super function
- Multiple inheritance

Programming paradigm

Paradigm = style of computer programming

- Procedural languages:
  - ➡ Describe step by step the procedure that should be followed to solve a specific problem.
- Object-oriented programming:
  - ➡ Data and methods of manipulating data are kept as single unit called object
  - ➡ A user can access the data via the object's method
  - ➡ The internal working of an object maybe changed without affecting any code that uses the object

Procedural Example

```python
score = 0

question1 = """What is the meaning of HPC:
            1. High Performance Computing
            2. Higgs Portal Channel
            3. High Portability Cluster
            4. Hello Peter Charles?"""

answer = input(question1)

if answer == "1":
    score += 1

question2 = """What is the meaning of CECI:
            1. Centre des equipes de calcul interactifs
            2. Consortium des Equipements de calculs intensifs
            3. This is the french word for "this"
            4. Cool Equipe Connaissant Ipython?"""

answer = input(question2)

if answer == "2":
    score += 1


print("You have %d/2 correct answer" % score)
```

Idea: a program doing some trivia test

Pretty simple and very readable code

Procedural Example

```python
score = 0

question1 = """What is the meaning of HPC:
            1. High Performance Computing
            2. Higgs Portal Channel
            3. High Portability Cluster
            4. Hello Peter Charles?"""

answer = input(question1)

if answer == "1":
    score += 1

question2 = """What is the meaning of CECI:
            1. Centre des equipes de calcul interactifs
            2. Consortium des Equipements de calculs intensifs
            3. This is the french word for "this"
            4. Cool Equipe Connaissant Ipython?"""

answer = input(question2)

if answer == "2":
    score += 1


print("You have %d/2 correct answer" % score)
```

Idea: a program doing some trivia test

Pretty simple and very readable code

Issue:
- formatting of the question is done by hand
- some repetition in the logic
  (Does not really hurt here)

Better Procedural Example

In procedural case, those issues are solved with Function:
This avoid to repeat itself when adding functionality like
- check that answer is a valid number
- consistent formatting of the questions

```
def ask(question, answers):

    ans = "\n".join([f"{i+1}: {a}" for i, a in enumerate(answers)])
    valid = [str(i+1) for i in range(len(answers))]
    print_valid = ",".join(valid)
    text = f"{question}\n{ans}\nReply by {print_valid}\n>"

    while True:
        receive = input(text)
        if receive in valid:
            return int(receive)
        else:
            print(f"invalid answer: Please Retry. Valid answer are {print_valid}")
```

Better Procedural Example

In procedural case, those issues are solved with Function:
This avoid to repeat itself when adding functionality like
- check that answer is a valid number
- consistent formatting of the questions

```
def ask(question, answers):

    ans = "\n".join([f"{i+1}: {a}" for i, a in enumerate(answers)])
    valid = [str(i+1) for i in range(len(answers))]
    print_valid = ",".join(valid)
    text = f"{question}\n{ans}\nReply by {print_valid}\n>"

    while True:
        receive = input(text)
        if receive in valid:
            return int(receive)
        else:
            print(f"invalid answer: Please Retry. Valid answer are {print_valid}")
```

Better Procedural Example

Better code:
- a function allows to avoid to repeat itself
- data are a bit more structured

```
if __name__ == "__main__":
    score = 0
    if ask("What is the meaning of HPC", ["High Performance Computing",
                                          "Higgs Portal Channel",
                                          "High Portability Cluster",
                                          "Hello Peter Charles"]) == 1:
        score += 1
    if ask("What is the meaning of CECI", ["Centre des equipes de calcul interactifs",
                                          "Consortium des Equipements de calculs intensifs",
                                          "This is the french word for \"this\"",
                                          "Cool Equipe Connaissant Ipython"]) == 2:
        score += 1
    print("You have %d/2 correct answer" % score)
```

Issue:
- Data is not well structured… (correct answer check)
- still repetition for the score handling
  (problematic if this move to a more complex handling)

Let's create a data structure

```
3   class Question:
4
5       def __init__(self, question, answers, correct):
6           self.question = question
7           self.answers = answers
8           self.correct = correct
9
```

- "Question" is a class/data structure
  - ➡ __init__ is the "constructor"
    - ✦ It is called after an "OBJECT" (named self) is created
    - ✦ Allow to setup initial value

```
q = Question("what is my name?", ["Olivier", "Hal", "Damien", "Bernard"], correct=1)
```

[2] ✓ 0.1s Python

```
q.correct
```

[3] ✓ 0.2s Python

```
1
```

```
q.correct = 2
print(q.correct)
```

[4] ✓ 0.2s Python

```
2
```

Let's create a data structure

```python
class Question:

    def __init__(self, question: str, answers: list[str], correct:int):
        self.question = question
        self.answers = answers
        self.correct = correct
```

- "Question" is a class/data structure
  - ➡ __init__ is the "constructor"
    - ✦ It is called after an "OBJECT" (named self) is created
    - ✦ Allow to setup initial value

```
[2]  q = Question("what is my name?", ["Olivier", "Hal", "Damien", "Bernard"], correct=1)
     ✓ 0.1s                                                                    Python

[3]  q.correct
     ✓ 0.2s                                                                    Python
...  1

[4]  q.correct = 2
     print(q.correct)
     ✓ 0.2s                                                                    Python
...  2
```

Naming convention

```
3    class Question:
```

```
    q1 = Question('
```

```
q.correct
```

```
self
```

- Question is called a class
- q is named object or instance of the class Question
- “correct” is an attribute of the class/object
- This is the current instance

➡ Convention is to use self (but not enforced)

Function can take data-structure

```
def ask(question, answers):
    ans = "\n".join([f"{i+1}: {a}" for i, a in enumerate(answers)])
    valid = [str(i+1) for i in range(len(answers))]
    print_valid = ",".join(valid)
    text = f"{question}\n{ans}\nReply by {print_valid}\n>"

    while True:
        receive = input(text)
        if receive in valid:
            return int(receive)
        else:
            print(f"invalid answer: Please Retry. Valid answer are {print_valid}")
```

Style comment:
- typically do not put shortcut at the beginning of the function but directly call object.attribute

Function can take data-structure

```
def ask(onequestion):

    question = onequestion.question
    answers = onequestion.answers

    ans = "\n".join([f"{i+1}: {a}" for i, a in enumerate(answers)])
    valid = [str(i+1) for i in range(len(answers))]
    print_valid = ",".join(valid)
    text = f"{question}\n{ans}\nReply by {print_valid}\n>"

    while True:
        receive = input(text)
        if receive in valid:
            return int(receive)
        else:
            print(f"invalid answer: Please Retry. Valid answer are {print_valid}")
```

Style comment:
- typically do not put shortcut at the beginning of the function but directly call object.attribute

Example code

creation of the data-structure (now ready to be read from external file/ database/…)

```
all_questions = []

q1 = Question("What is the meaning of HPC", ["High Performance Computing",
                                             "Higgs Portal Channel",
                                             "High Portability Cluster",
                                                   "Hello Peter Charles"],
              correct=1)
q2 = Question("What is the meaning of CECI", ["Centre des equipes de calcul interactifs",
                                              "Consortium des Equipements de calculs intensifs",
                                              "This is the french word for \"this\"",
                                              "Cool Equipe Connaissant Ipython"],
       correct=2)

all_questions = [q1,q2]
```

Running the code

```
score = 0
for q in all_questions:
    if ask(q) == q.correct:
        score +=1
    else:
        print(f"No the correct answer was {q.correct}")
print("You have %d/2 correct answer" % score)
```

Quite nice encapsulation

Function and object

```
print(q.correct)
def change_correct(question, new_correct):
    question.correct = new_correct
change_correct(q, 3)
print(q.correct)
```

✓ 0.2s

```
1
3
```

- Remember that you can modify any attribute of an object within a function (python specific)

Dataclass

- Let assume, we want to avoid the issue and be sure that the data are untouched during the all program
  - ➡ This is a perfect example for using the "dataclass" of python

```python
import dataclasses
from dataclasses import dataclass

@dataclass(frozen=True)
class Question:
    """Class for keeping track of one question"""
    question: str
    answers: list[str] = dataclasses.field(default_factory=list)
    correct: int = 1
```

@ is called a **decorator,** that allow to define "for you", the __init__, __str__, __repr__ of the class for you.
The frozen=True allows to ensure that data are readonly

Dataclass example

```
change_correct(q, 3)
```

⊗ 0.6s

```
---------------------------------------------------------------------------
FrozenInstanceError                       Traceback (most recent call last)
Cell In [19], line 1
----> 1 change_correct(q, 3)

Cell In [16], line 3, in change_correct(question, new_correct)
      2 def change_correct(question, new_correct):
----> 3     question.correct = new_correct

File <string>:4, in __setattr__(self, name, value)

FrozenInstanceError: cannot assign to field 'correct'
```

```
print(q)
```

✓ 0.2s

```
Question(question='what is my name?', answers=['Olivier', 'Hal', 'Damien', 'Bernard'], correct=1)
```

Before (hand made data-structure) was:

```
print(q)
```

✓ 0.2s

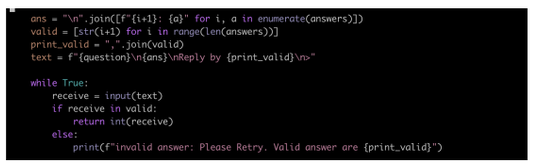
```
<__main__.Question object at 0x106605ff0>
```

Class

- A Class is a data-structure with function/method

```python
@dataclass(frozen=True)
class Question:
    """Class for keeping track of one question"""
    question: str
    answers: list[str] = dataclasses.field(default_factory=list)
    correct: int = 1
```

```python
def ask(onequestion):

    question = onequestion.question
    answers = onequestion.answers
```



```python
ans = "\n".join([f"{i+1}: {a}" for i, a in enumerate(answers)])
valid = [str(i+1) for i in range(len(answers))]
print_valid = ",".join(valid)
text = f"{question}\n{ans}\nReply by {print_valid}\n>"

while True:
    receive = input(text)
    if receive in valid:
        return int(receive)
    else:
        print(f"invalid answer: Please Retry. Valid answer are {print_valid}")
```

Content of the function

Class

- A Class is a data-structure with function/method

```python
@dataclass(frozen=True)
class Question:
    """Class for keeping track of one question"""
    question: str
    answers: list[str] = dataclasses.field(default_factory=list)
    correct: int = 1
```

```python
    def ask(self):

        question = self.question
        answers = self.answers
```

```python
ans = "\n".join([f"{i+1}: {a}" for i, a in enumerate(answers)])
valid = [str(i+1) for i in range(len(answers))]
print_valid = ",".join(valid)
text = f"{question}\n{ans}\nReply by {print_valid}\n>"

while True:
    receive = input(text)
    if receive in valid:
        return int(receive)
    else:
        print(f"invalid answer: Please Retry. Valid answer are {print_valid}")
```

Content of the function

Class

- A Class is a data-structure with function/method

```python
@dataclass(frozen=True)
class Question:
    """Class for keeping track of one question"""
    question: str
    answers: list[str] = dataclasses.field(default_factory=list)
    correct: int = 1
```

```python
    def ask(self):

        question = self.question
        answers = self.answers
```

```python
ans = "\n".join([f"{i+1}: {a}" for i, a in enumerate(answers)])
valid = [str(i+1) for i in range(len(answers))]
print_valid = ",".join(valid)
text = f"{question}\n{ans}\nReply by {print_valid}\n>"

while True:
    receive = input(text)
    if receive in valid:
        return int(receive)
    else:
        print(f"invalid answer: Please Retry. Valid answer are {print_valid}")
```

Content of the function

Note:

- the indentation
  - this is now a function of the class
  - gain in clarity

Class

- A Class is a data-structure with function/method

```python
@dataclass(frozen=True)
class Question:
    """Class for keeping track of one question"""
    question: str
    answers: list[str] = dataclasses.field(default_factory=list)
    correct: int = 1
```

```python
    def ask(self):

        question = self.question
        answers = self.answers
```

```python
ans = "\n".join([f"{i+1}: {a}" for i, a in enumerate(answers)])
valid = [str(i+1) for i in range(len(answers))]
print_valid = ", ".join(valid)
text = f"{question}\n{ans}\nReply by {print_valid}\n>"

while True:
    receive = input(text)
    if receive in valid:
        return int(receive)
    else:
        print(f"invalid answer: Please Retry. Valid answer are {print_valid}")
```

Content of the function

Note:

- the indentation
  - this is now a function of the class
  - gain in clarity

- the use of self
  - the first attribute of ALL class function is the instance itself (self)

Class

You can have many function/method, with additional argument

```
    def print_correct(self, prefix=""):
        if not prefix:
            t= "T"
        else:
            t="t"
        print(f"{prefix} {t}he correct answer was {self.correct}: {self.answers[self.correct-1]}")
```

Class

You can have many function/method, with additional argument

```
    def print_correct(self, prefix=""):
        if not prefix:
            t= "T"
        else:
            t="t"
        print(f"{prefix} {t}he correct answer was {self.correct}: {self.answers[self.correct-1]}")
```

Main code was before:

```
score = 0
for q in all_questions:
    if ask(q) == q.correct:
        score +=1
    else:
        print(f"No the correct answer was {q.correct}")
```

Class

You can have many function/method, with additional argument

```
    def print_correct(self, prefix=""):
        if not prefix:
            t= "T"
        else:
            t="t"
        print(f"{prefix} {t}he correct answer was {self.correct}: {self.answers[self.correct-1]}")
```

Main code was before:

```
score = 0
for q in all_questions:
    if ask(q) == q.correct:
        score +=1
    else:
        print(f"No the correct answer was {q.correct}")
```

Syntax change from ask(q) -> q.ask()

Class

You can have many function/method, with additional argument

```
    def print_correct(self, prefix=""):
        if not prefix:
            t= "T"
        else:
            t="t"
        print(f"{prefix} {t}he correct answer was {self.correct}: {self.answers[self.correct-1]}")
```

Main code was before:

```
score = 0
for q in all_questions:
    if ask(q) == q.correct:
        score +=1
    else:
        print(f"No the correct answer was {q.correct}")
```

Syntax change from ask(q) -> q.ask()

```
score = 0
for q in all_questions:
    if q.ask() == q.correct:
        score +=1
    else:
        q.print_correct(prefix="No,")
```

Advance possibility

- Allow to sum object

```python
from __future__ import annotations

import dataclasses
from dataclasses import dataclass

@dataclass
class Color:
    """Class for keeping track of one question"""
    red: float = 0.
    green: float = 0.
    blue: float = 0.

    def __add__(self, other: Color) -> Color:
        r = (self.red+other.red)/2
        g = (self.green+other.green)/2
        b = (self.blue+other.blue)/2
        return Color(r,g,b)


blue = Color(blue=1)
red = Color(red=1)
print(blue+red)
```

- You can customise all python operator (assignment, addition, multiplication, …)

Class inheritance

- Extend what a class can do by inheritance

```
class QuestionStatus(Question):
```

Mother

Child

- An object of type QuestionStatus is also of type Question
- By default all attributes/functions/method of the mother class are present in the child class.
  - The child can overwrite the mother behaviour

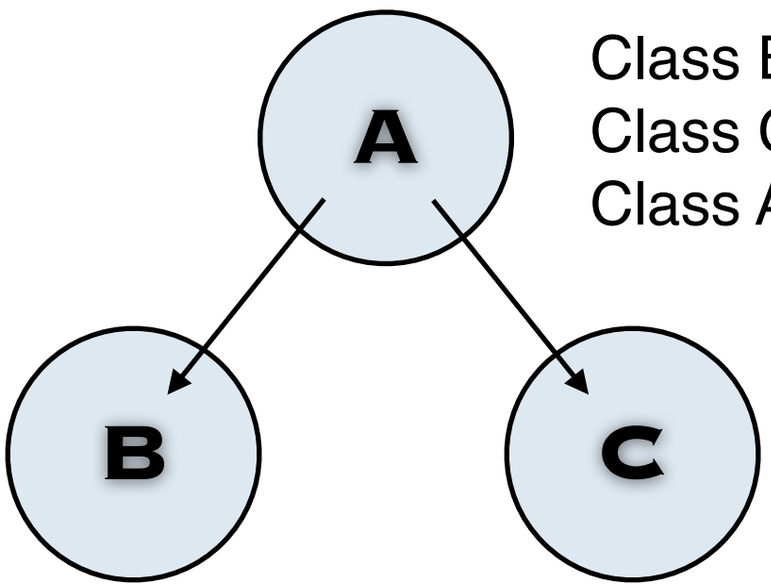
```
class QuestionStatus(Question):

    def __init__(self, *args, **opts):
        Question.__init__(self, *args, **opts)
        self.attempt = 0
        self.success = 0
```

Overwrite the constructor ..

… But explicitly call the it to expand it

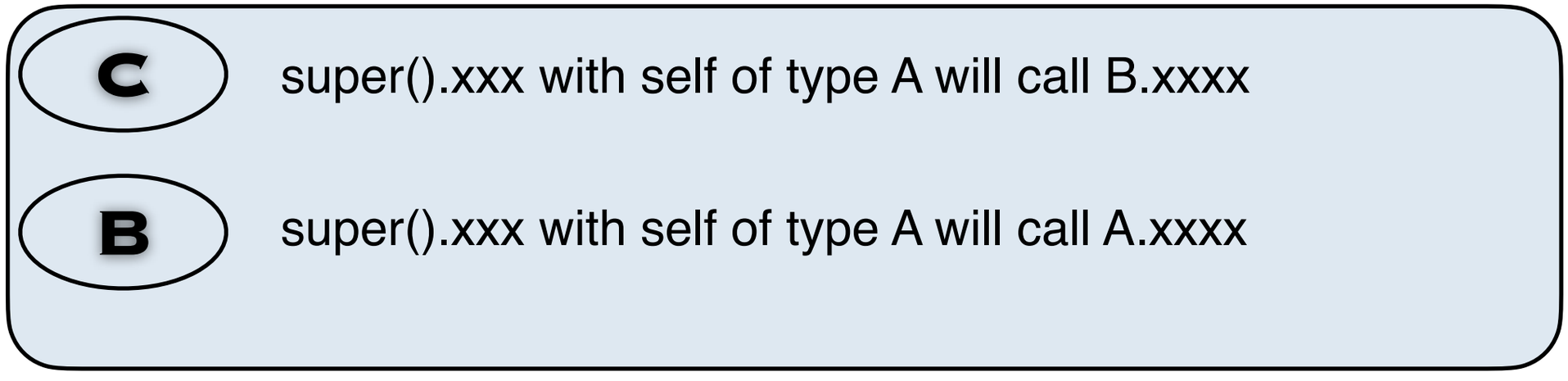
Multiple Class inheritance

- Mulitiple class inheritance possible



Class B:
Class C:
Class A (B,C):

- If both B and C defines a function
  - Use the one in B
- Mainly use for cooperative feature
  - In that case do not use
    - A.__init__(self, …)
    - But super().__init__(—no self—)



Conclusion

- Structuring your data is Essential
  - ➡ Both in procedural / object-oriented programming
- Dataclasses allows simple and powerful way to create data-structure (read-only, slots,…)
- A class is a data-structure with functions
  - ➡ A function is associated to the data-structure
- Inheritance allows to recycle code between different class
  - ➡ A class should do one (and only one) task
  - ➡ Then you can compose object