1811/2807/7001ICT
Programming Principles

School of Information and Communication Technology
Griffith University

Trimester 1, 2024

25 Modules and Classes

New data types are defined in Python by defining new classes.

25.1 Defining modules

Defining your own module is the same as creating a script.

Mostly in a module you will define functions and classes that will be used by the script that imports it.

The first statement should be a docstring.

25.2 Members

A class defines the members of the objects that will be made as instances of the class.

A class has two main kinds of members:

- attributes (or fields) for storing information, the state of the object; and
- methods, functions that define what can be done to any by the object.

Members may be either:

- class (or static) members that belong to and exist within the class itself and may be used even before any objects are created (instantiated); and
- instance members that belong to and exist only in the objects after they have been instantiated.

Each object has its own instance members, so each object may have different values saved in their attributes.

But all objects share the same values in their class attributes.

Python defines private members as follows.

> If a member name starts with two underscores, it is a private member.

Don't write code that uses and therefore relies on values in private attributes, as your code will break if those attributes are removed or used differently in a new version of the class you are using.

25.3 Class template

A typical class might be defined like this:

```
class name:
    statement(s)
```

where the statements are any kind of Python statements, but usually are statements that create variables (attributes) or functions (methods).

25.4 Empty classes

A class can be defined with no members at all.

```
# file: empty1.py
# Define an empty class.

class Empty:
   """An empty class."""
   pass
```

This seems useless until you know that attributes are created on the fly like any variable.

After the definition of the class above, we can create a variable with a new instance of the class, by calling its constructor.

```
mt = Empty()
```

Then we can give that object new attributes, and use them.

```
mt.x = 1
mt.y = 2
print(mt.x, mt.y)
```

So objects can be useful, general-purpose containers, as in JavaScript.

25.5 Special members

Some members of a class are predefined and special.

Their names all start and end with two underscores.

They can be redefined in your class definition to give your objects customised behaviours.

The most important is the `__init__` method.

Redefine this to change the way the class's constructor works.

In this example, we define a class for storing a person's name.

This is likely to be useful and reusable, so we'll put it in a module.

```
# file: name1.py

"""A module that defines class Name."""
```

The first statements of the module and the class are docstrings.

```
class Name:
   """A person's name."""
```

Now within the class we redefine `__init__` to have parameters.

```
def __init__(self, familyName, otherNames, title,
             famFirst = False):
    """Makes a person's name.

    For example: Name("Trump", "Donald John", "Mr") or
    Name("Kim", "Jong-un", "Mr", famFirst = True)."""
    self._familyName = familyName
    self._otherNames = otherNames
    self._title = title
    self._famFirst = famFirst
```

Notes:

- The first parameter of any method must be `self`, which is a reference to the current object, like `this` in, Java, etc.
- Instance attributes are created by assigning them values.
- These attributes start with underscores, so they are intended to be private.
- Within the method, all uses of the attributes must be qualified with `self`.

Give our class some more methods.

```
def fullName(self):
   """Return person's full name in preferred format."""
   if self._famFirst:
      return self._title + " " + self._familyName + \
         " " + self._otherNames
   else:
      return self._title + " " + self._otherNames + \
         " " + self._familyName

def canonName(self):
   """Return person's full canonical name."""
   return self._familyName + ", " + self._otherNames \
      + ", " + self._title
```

```
    def firstName(self):
        """Return person's first other name."""
        others = self._otherNames.split()
        return others[0]

    def __str__(self):
        return self.fullName()
```

`__str__` is a predefined method that is used by the string constructor, `str()` to make a string from an object.

If you redefine it, you can easily print your objects.

This program tests our class.

```
# file: test-name1.py
# Test driver for name1.py.

from name1 import *

def test(name):
   print(name)
   print(name.firstName())
   print(name.canonName())

andrew = Name("Rock", "Andrew B.", "Dr")
test(andrew)
shinji = Name("Ikari", "Shinji", "Mr", famFirst = True)
test(shinji)
```

Run it.

```
$ python3 test-name1.py
Dr Andrew B. Rock
Andrew
Rock, Andrew B., Dr
Mr Ikari Shinji
Shinji
Ikari, Shinji, Mr
$
```

See also files `address1.py` and `test-address1.py`, which similarly define an address class.

25.6 Association

Of course the attributes of an object may be object references.

This class defines a person, that *has* a name and an address.

```
# file: person1.py

"""A module that defines class Person."""

class Person:
   """A Person has a name and an address."""
```

```
def __init__(self, name, address):
    """Makes a Person.

    name is a Name.
    address is an Address."""
    self.name = name
    self.address = address

def __str__(self):
    return str(self.name)
```

25.7 Inheritance

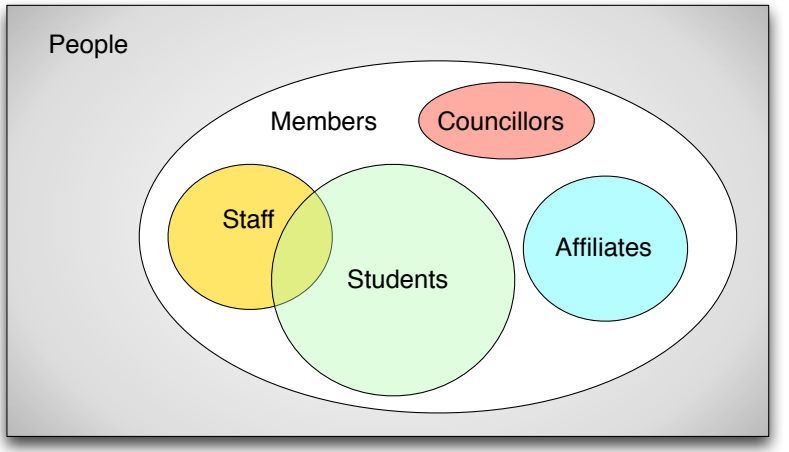
An existing class may be *extended* by defining a new class that *inherits* all of the members of the existing class, adding more or redefining some of them.

The new class is called a *subclass* of the old class, and the old class is called its *superclass*.

Consider this situation:

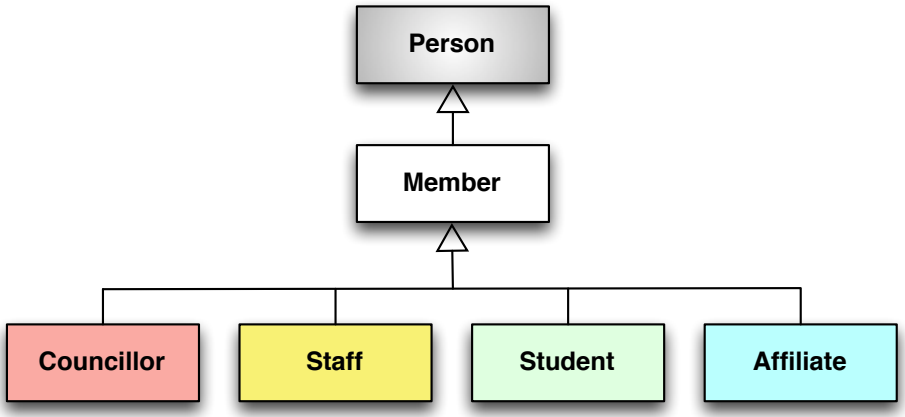
> A university has a mailing list of people it must write letters to.
>
> Some are just people external to the university, but others are members of the university and have an ID.
>
> There are many different kinds of members within the university.

The situation may be represented as sets with a Venn diagram:



A student *is* a member, and a member *is* a person.

Viewing the situation as classes:



A `Member` is a subclass of `Person` and a `Student` is a subclass of `Member`.

This module defines classes for members, staff, and students.

```
# file: members1.py

"""This module defines the members of the university."""

from person1 import *

class Member (Person):
   """A Member is a Person with an ID that letters can be
   written to."""
```

Class `Member` has its superclass in parentheses after its name.

The subclass redefines its `__init__` method with one more parameter than a `Person`.

It can reuse the superclass's `__init__` method by calling it, qualified by the name of the superclass.

```
def __init__(self, name, address, id):
    """Makes a Member.

    name is a Name.
    address is an Address.
    id is a uni ID."""
    Person.__init__(self, name, address)
    self.id = id
```

Redefine (override) `__str__` to include the ID.

```
def __str__(self):
    return self.id + " " + self.name.fullName()
```

Make a new method for members.

```
def writeALetter(self, body):
    """Write a letter to this Member with the given
    body text."""
    print(self.name)
    print(self.address)
    print()
    print("Dear " + self.name.firstName(), end = ",\n\n")
    print(body)
    print("\nRegards,")
    print("your University.")
```

Make a new subclass of `Member` for a student.

```
class Student (Member):
   """A Student is a Member that can be excluded."""

   def exclude(self):
      """Write an exclusion letter."""
      self.writeALetter("""Your grades are rubbish.
You are forthwith excluded.
Pay your fees on the way out.""")
```

And for staff.

```
class Staff (Member):
   """A Staff is a Member that can be fired."""

   def fire(self):
      """Write a termination letter."""
      self.writeALetter("""Your SET scores are rubbish.
You are forthwith fired.
Hand in your keys on the way out.""")
```

Testing.

```
# file: test-members1.py
# Test driver for members1.py.

from name1 import *
from address1 import *
from members1 import *

andrew = Staff(
   Name("Rock", "Andrew B.", "Dr"),
   Address("10 Fred Street", "Lower Grunge",
           "Kingsland", "9999"),
   "S1234567")
andrew.fire()
print("-------------------------------------------------")
```

```
shinji = Student(
   Name("Ikari", "Shinji", "Mr", famFirst = True),
   Address("1034/23 Suwaru Road", "New Tokyo", "Tokyo",
           "ABZX"),
   "S7654321")
shinji.exclude()
```

```
$ python3 test-members1.py
Dr Andrew B. Rock
10 Fred Street
Lower Grunge
Kingsland 9999

Dear Andrew,

Your SET scores are rubbish.
You are forthwith fired.
```

```
Hand in your keys on the way out.

Regards,
your University.
-------------------------------------------------
Mr Ikari Shinji
1034/23 Suwaru Road
New Tokyo
Tokyo ABZX

Dear Shinji,

Your grades are rubbish.
You are forthwith excluded.
Pay your fees on the way out.

Regards,
```

```
your University.
$
```

Section summary

This section covered:

- how to create your own classes in Python.