

Classes And Objects

Encapsulation/information hiding, inheritance, modular design in python/multi-file programs.

James Tam

Decomposing Large Programs: By File

- Because real life programs are large, they are not only divided into functions but also split into multiple files.
- Example: There's so many files for the game RPG Icewind Dale that they are distributed among several folders (this is not unique to this game).

Cache	2023-07-02 5:57 PM	File folder	
CD2	2023-07-02 6:00 PM	File folder	
Characters	2023-07-02 6:12 PM	File folder	
Data	2023-07-02 6:00 PM	File folder	
Music	2023-07-02 5:59 PM	File folder	
Override	2023-07-02 5:59 PM	File folder	
Portraits	2023-07-02 6:13 PM	File folder	
Scripts	2023-07-02 5:57 PM	File folder	
Sounds	2023-07-09 1:32 AM	File folder	
Temp	2023-07-02 6:08 PM	File folder	
Temprave	2023-07-02 5:59 PM	File folder	
3dfs.dll	1998-06-02 6:32 AM	Application exten...	689 KB
CHITINKEY	2000-06-22 1:45 AM	KEY File	212 KB
Config.exe	2000-06-27 6:37 PM	Application	713 KB
Dialog.tlk	2000-06-22 11:39 PM	TLK File	2,838 KB
Icewind.ini	2023-07-02 6:08 PM	Configuration sett...	1 KB
Icewind_Dale_I_Manual.pdf	2023-07-03 3:20 AM	Adobe Acrobat D...	1,532 KB
IDMain.exe	2000-08-14 1:33 PM	Application	6,140 KB
Keymap.ini	2000-06-22 9:42 PM	Configuration sett...	8 KB
Language.ini	2000-06-22 11:03 PM	Configuration sett...	12 KB
README_ENG.TXT	2000-08-09 1:53 PM	Text Document	44 KB
Uninst.isu	2023-07-02 6:03 PM	ISU File	617 KB

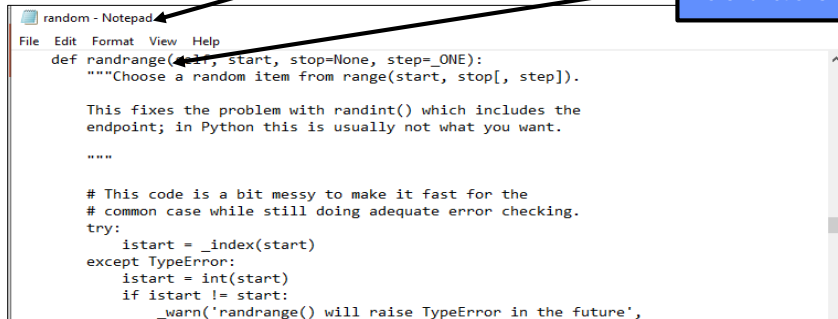
James Tam

Python File Decomposition: Modules

- Each module is a separate library of python features (functions, class definitions).
- Recall: the 'Random' module:

Name of file:
random.py

File contains 1 or
more functions



```
random - Notepad
File Edit Format View Help
def randrange(start, stop=None, step=_ONE):
    """Choose a random item from range(start, stop[, step]).

    This fixes the problem with randint() which includes the
    endpoint; in Python this is usually not what you want.

    """

    # This code is a bit messy to make it fast for the
    # common case while still doing adequate error checking.
    try:
        istart = _index(start)
    except TypeError:
        istart = int(start)
        if istart != start:
            _warn('randrange() will raise TypeError in the future',
```

James Tam

Review: Using The Code In A Module

- Add the name 'Random' to your program
 - Format:
`import <Module/filename>`
 - Example:
`import random`
- Running a function/method from this module
 - Format:
`<Module/filename>.<function/method call>`
 - Example:
`print(random.randrange(1,6))`

James Tam

Defining Your Own Module

- Name of the folder containing the full online example:
1st_module_example
- To start the whole program run the module with the 'start' function (in this case it is Driver.py).

Filename: Draw.py

```
def rectangle():
    print("""
    #####
    #####
    #####
    #####
    #####
    #####
    #####
    #####""")

def right():
    print("""
    #
    ##
    ###""")

def triangle():
    print("""
    #
    ##
    ###""")
```

James Tam

Defining Your Own Module (2)

- **Filename: Driver.py**

```
import Draw

def start():
    Draw.rectangle()
    Draw.right()
    Draw.triangle()

start()
```

Adding the name
'Draw' to your
program

Running functions from
the 'Draw' module
(similar to running the
random methods this
requires <module
name>.<method name>)

Filename: Draw.py

```
def rectangle():
    print("""
    #####
    #####
    #####
    #####
    #####
    #####
    #####
    #####""")

def right():
    print("""
    #
    ##
    ###""")

def triangle():
    print("""
    #
    ##
    ###""")
```

Naming The Starting Module

- Recall: The function that starts a program (first one called) should have a good self-explanatory name e.g., “`start()`” or follow common convention e.g., “`main()`”
- Similarly the file module that contains the ‘`start()`’ or ‘`main()`’ function should be given an appropriate name e.g., “`Driver.py`”, “`Start.py`”, “`Main.py`” (it’s the ‘driver’ of the program or the starting point)

Filename: “`Driver.py`”

```
def start():  
    #Instructions  
  
start()
```

James Tam

Importing Modules Containing Class Definitions

- **Approach 1:** import just the name of the file containing the class definition.
 - Example: `import PersonFile` (similar this previous approach: `import Random`)
 - Advantage:
 - Allows access to other ‘names’ in the file e.g. other utility methods, constants etc.
 - Recall `Random.py` contains: `Random.randrange(0,6)`, `Random.randint(1,6)`
 - Disadvantage:
 - The name of the file must be included along with the name of the function/method/attribute e.g. `Random.randint(1,6)`

File: `Random.py`

```
def randint(start,end):  
def randrange(start,end):
```

James Tam

Importing Modules Containing Class Definitions

- **Approach 2:** import just the name of the file containing the class definition.
 - Example: `from PersonFile import Person` (or using this approach with an existing library `from Random import randint`)
 - Advantage:
 - Imports only the names needed (reduced conflicts between the names in the module being imported and the file where the import is included).
 - `from Random import randint`
 - `#only imports the randint name`

James Tam

Approach 1: An Example

- **Name of the folder complete online example:**
`2nd_oo_module_example`
 - Only the **module/filename** is imported so **other names** (e.g. **class name**) must be prefaced by the module name ('context' needed).
 - Note: `sayName` is accessed via a **reference name** not class name.

Filename:

`PersonFile.py`

```
class Person:
    def __init__(self):
        self.name = "I have no name :("
    def sayName(self):
        print("My name is...", self.name)

def fun():
    print("called fun")
```

Filename:

`Driver.py`

```
import PersonFile

def start():
    #Only filename imported
    aPerson = PersonFile.Person()
    aPerson.sayName()
    PersonFile.fun()

start()
```

James Tam

Approach 2: An Example

- Name of the folder complete online example:
3rd_oo_module_example
 - Only imports the **class name**, other names (e.g. function name) cannot be accessed.
 - The **class name** doesn't need to be prefaced by the **module name**.
 - References to the imported name are shorter.
 - Reduces the possibility of naming conflicts e.g. if there's already a function 'fun' in Driver.py (won't conflict with 'fun' in Person).

Filename:

PersonFile.py

```
class Person:
    def __init__(self):
        self.name = "I have no name :("
    def sayName(self):
        print("My name is...", self.name)

def fun():
    print("called fun")
```

Filename:

Driver.py

```
from PersonFile import Person

def start():
    aPerson = Person()
    aPerson.sayName()

start()
```

James Tam

Object-Oriented Design: Advantage Over Procedural Decomposition

- Procedural approach: functions can allow for nonsensical behaviors e.g. “**flying pigs**”
- E.g.

```
def fly(aFlyer):
    ...

pigs = list["pig1", "pig2"]
fly(pigs)
```

James Tam

Recall: Objected Approach **Ties Behaviors** (Methods) To Classes

- Capabilities are defined in a class (in this case it's the **Flyer()** which is a **parent class**) and all classes that inherit all the **abilities of a flyer** (in this cases the child class: **Airplane**).

```
class Flyer():  
    def fly(self):  
        ...
```

- Via inheritance:** class definitions be extended by specifying that **'child' classes** (derived from the **parent**) **inherit** (are able to access) the attributes and methods of the parent.

```
class Airplane(Flyer):
```

In python this allows an Airplane object to 'fly' by inheriting the abilities of the 'Flyer'.

Alternative example: Java

```
public class Airplane extends  
Flyer  
{  
  
}
```

James Tam

(A Very Simple) Inheritance Example

- Name of the folder containing the full online example:
4th_inheritance

Filename:
Flyer.py

```
class Flyer():  
    def fly(self):  
        print("Engage flying mode")
```

Filename:
Airplane.py

```
from Flyer import Flyer  
  
class Airplane(Flyer):  
    def beginTrip(self):  
        print("seat belts")  
        self.fly()
```

Filename:
Start.py

```
from Airplane import Airplane  
def start():  
    p = Airplane()  
    p.beginTrip()  
  
start()
```

```
Passengers fasten your seat belts  
Engage flying mode  
'
```

James Tam

New Terms: Previous Example

- **Parent class**: a class from which another class is derived.
 - `class Flyer():`
- **Child class**: a class that is derived from the parent class.
 - `class Airplane(Flyer):`
- **Inheritance**: the relationship between a parent and child class.
 - **Format**:
`class <ChildClass>(<Parent class>):`
 - **Example (python)**:
`class Airplane(Flyer):`
 - **Example (java – not needed for 231 but needed for 233)**:
`public class Airplane extends Flyer`

James Tam

New Term: Encapsulation

- **Definition 2 for encapsulation**: hiding the variable attributes of a class so they can only be accessed and changed in a specific fashion (via the class methods).
- Alternative term, **information hiding**: the private attributes are hidden behind the public methods of a class.
- Enforcing encapsulation/information hiding:
 - **Approach 1** (better approach – many languages do it this way): It's a built in rule of the syntax i.e. violating it results in syntax/translation error and the program not translate let alone run.
 - **Approach 2** (python employs this approach):
 - Programming **stylistic conventions** specify an attribute as private.
 - It's analogous to how named constants are declared:
`GST = 0.05`
`GST = 0.1 #Bad style but unfortunately it only results in a #logic error`

James Tam

Stylistically Demonstrating **Something Should Not Accessible**

- The python convention is to preface the attribute with two underscores).
- But the stylistic approach merely indicates to other programmers that the attribute should not be changed.

James Tam

Encapsulation/Information Hiding: Python

- Name of the folder containing the full online example:
5th_information_hiding
- **Changes/access** are possible for **private attributes** outside of the class.
 - These parts are **bad** and should not be allowed outside of the class (but python allows it).
 - o Private attributes (e.g. `__friends`) or even methods should never be directly accessible outside of a class definition.

```
class Person:
    __friends = []

    def __init__(self):
        self.__friends = ["no friends"]

jim = Person()
print(jim._Person__friends) #Attribute accessed!
jim._Person__friends = [] #Attribute changed!
jim._Person__friends.append("Stacey Hearn")
print(jim._Person__friends)
```

James Tam

Encapsulation/Information Hiding Vs. Abstraction

- Don't mix the two up!
- Encapsulation/information hiding: protecting variable attributes of a class by restricting access via controlled mechanisms (methods that prevent misuse).
- Class Person:

```
def __init__(self,newAge):
    if(newAge>0):
        self.__age = newAge
    else:
        print("Age can't be negative")
```

James Tam

Enforcing Information Hiding

- Many languages (e.g. Java) enforce this with the rules of the language i.e. directly trying to change or access a **private attribute** results in a **syntax error**.
- Example (unless you are later told this isn't needed for a CPSC 217/231 exam but is more than fair game for CPSC 219/233).\

```
public class Person
{
    public Person()
    {
        //this ~self
        this.age = 0;
    }
    private int age;
    public void setAge(int newAge)
    {
        this.age = newAge;
    }
}

public class Driver
{
    public static void main
    (String [] args)
    {
        //Calling constructor
        Person aPerson = new Person();
        //Syntax error:
        aPerson.age = 12;
    }
}
```

James Tam

In Case You're Wondering: Why Bother?

- Information hiding can prevent in appropriate access (a part of a program has access to information that it should not) or **preventing information being set to incorrect values** (e.g. outside an allowable range).
- Example: again for your reference this is CPSC 233 material.

```
public class Person {  
    private int age;  
    public void setAge(int newAge) {  
        //Valid age in range from 0-118.  
        if((age >=0)&&(age<=118)) {  
            this.age = newAge;  
        }  
        else {  
            System.out.println("Age outside 0-118");  
        }  
    }  
}
```

- Properly implemented: information hiding is a unique advantage of Object-Oriented programming.

James Tam

New Term: Method/Function Signature

- **Method signature**: the name and parameter lists of the methods of class.
- It's how you use (call) the method.
- Example for a Person class:

Actions:

- Eat
- Sleep
- Secrete
- Multiply

Method signatures for class Person could be (parameter name, parameter type):

- eat(howMuch:String)
- sleep(duration:integer)
- secrete() #I don't want to know! ;)
- multiply(partner:Person)

James Tam

New Concept For O-O: Abstraction (2)

- Abstraction in O-O:
 - The inner details of the code in a class is not necessarily knowledge for other programmers (it is a 'black box' or an abstraction to any except for the programmer who implemented this class).
 - Instead all that is needed are the details as to how the class is used e.g. how methods are called.

You just have to know how to properly use the class methods or function not how they are implemented.

- Example (procedural): random module
 - Function randint
 - `randint(<min value>, <max value>):`
 - This function returns a random integer value within the range of the two parameters.
 - How this function generates the value is not relevant to other programmers.
 - The function is a 'black box' to outside programmers (inner details are blacked out or hidden).
 - Other programmers simply need to understand an abstraction of the function (e.g. how to use it, what value it produces etc.)

James Tam

New Concept For O-O: Abstraction

- Abstraction (Object-Orientation):
 - Much like with the procedural example (random library/module) other programs simply have to know how to use your class (what are the method signatures).
 - They do not have to know the "inner workings" of the class (the code inside each method).

Method signatures for class Person could be (parameter name, parameter type):

- `eat(howMuch:String)`
- `sleep(duration:integer)`
- `secrete() #I don't want to know! ;)`
- `multiply(partner:Person)`

- Example documentation for class Random (java).
 - <https://docs.oracle.com/javase/8/docs/api/java/util/Random.html>
 - Method: `public int nextInt(int bound)`
 - Returns a "...value between 0 (inclusive) and the specified value (exclusive)"
 - i.e. it behaves just like python's `Random.randrange(1,6)` #1-5

James Tam

After This Section You Should Now Know

- How to decompose a program (functions or classes) into modules.
- What are the benefits of Object-Oriented programming:
 - Inheritance
 - Information hiding/encapsulation (if properly implemented).
- New terminology:
 - abstraction,
 - information hiding/encapsulation,
 - How information hiding is implemented in python vs. other some other languages.
 - What are some of the benefits of information hiding.
 - inheritance
 - parent class
 - child class

James Tam

After This Section You Should Now Know (2)

- How to divide your program into different modules
- How inheritance can allow access to group of derived classes.

James Tam

Copyright Notification

- Unless otherwise indicated, all images in this presentation were provided courtesy of James Tam.