CS1302 - Lecture 7 Objects
(Dr Helena WONG)
==========

[Objects.ipynb]

- OOP
  - Motivation
  - Classes and Objects
  - Attributes
  - Object Aliasing

- Using Methods
  - File Objects
  - String Objects

- Operator Overloading
  - Dispatch on type
  - Data-directed programming

==========

[Helena's supplements]
Supplements for Contents
Exercises/Examples for Introduction

==========

==========
Supplements for Contents
==========

1. Objects

Why object-oriented programming? Manim examples: encapsulate implementation details

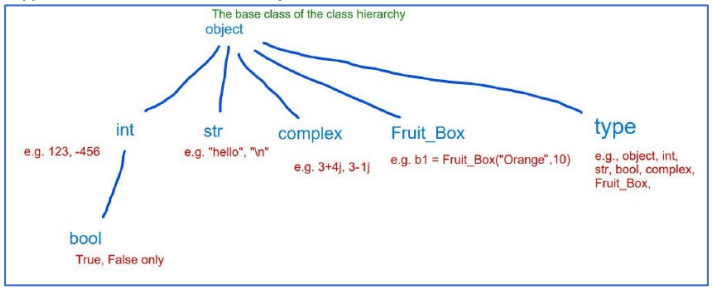
Definitions - **object, class, type, subclass, base class, member, attribute**

Python is a class-based object-oriented programming (OOP) language:
Each object is an instance of a class/type, which can be a subclass of one or more base classes.
An object is a collection of members/attributes, each of which is an object.

Basic concepts
(see Lecture_Wk08_Lecture07_supp_obj_and_cl.pdf)

Each object is essentially a *closure*.
(see Lecture_Wk06_Lecture06a_Supp.pdf)

Class/type: int, float, str, bool, object



isinstance(*obj*, *class*)
functions are "first-class" objects *(see Lecture_Wk05_Lecture05_Contents_andSupp.pdf)*
Almost everything in Python is an object What's not: e.g. keywords (if, for..) and syntax elements(+,*,whitespace..)

issubclass(*c1*, *c2*)
the *__bases__* attribute: bool.__bases__, int.__bases__

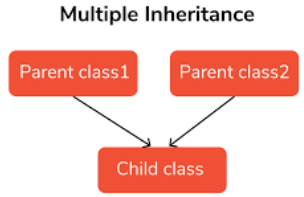
hasattr(obj, attribute_name)
objects have attributes (member), which can be a property (value) or a method (member function)

Others:

Member operator (.), object.variable_name, object.function_name()

bool.**mro()** (bool is subclass of int, every class is a subclass of the object class); **mro**: *method resolution order*

A subclass can inherit attributes from multiple base classes and define its own attributes.



Example: complex number, .**imag**, .**real, complex.conjugate**(x), x.conjugate()

dir: dir(1), dir(1+2j), dir(int), dir(complex)

callable (i.e., some kind of function)

2. Object Aliasing

==, is, id(x)

is simply checks whether two objects occupy the same memory, but

== calls the method (__eq__) of the operands to check the equality in value.

3. File Obejcts

!more - "!more" means running command 'more' in the shell

(For what is "shell command", google "geeksforgeeks terminal console shell command line")

Read a text file: **f=open(..), f.read(), f.close(), f.closed**

```
with open(..) as f:    # automatically call f.__enter__ and f.__exit__
                       # Note: "with" is not a loop!!
for line in f

f = open("...", 'r') # 'r', 'w', 'a'

f.write(..)

import os, os.path.dirname(..), os.makedirs(__, exist_ok=True)
os.remove(..), os.path.exists(..)
```

4. String Objects

```
string.find(substring)
string.split(separator, maxsplit) #rsplit
delimiter.join(substrings)
string.strip(character) #lstrip, rstrip
string.upper(), string.lower()
```

5. Operator Overloading

+, __add__

Dispatch on type, Data-directed programming

==========

Exercises/Examples for Introduction

==========

==========

[Exercise / Example 1] **type** and **dir**

==========

Run the following in OPTMentor. Compare!

```
print(type(1))
print(dir(1))

print(type(1 + 2j))
print(dir(1 + 2j))

print(type(int), type(complex))

print(dir(int))
print(dir(float))
print(dir(object))
```

==========

[Exercise / Example 2] **complex** , .**conjugate**

==========

(i) Fill in the blanks with these words (Guess!):

True, False, class, object

```
x1 = 1 + 2j
x2 = 1 - 2j
x3 = 123
print(isinstance(x1, complex)) # result: _____
print(isinstance(x3, complex)) # result: _____
print(isinstance(x3, int)) # result: _____
```

complex is a ____; x1 and x2 are ____ of complex

int is a ___; x3 is an ___ of int

(ii) Try / Guess the results

```
X1 = complex(1,2)    # same as X1 = 1 + 2j
print(X1)

X2 = X1.conjugate()
X3 = complex.conjugate(X1)

print(X2)
print(X3)
```

X2 = X1.conjugate() means to pass X1 to run complex.conjugate. Like this:

X3 = complex.conjugate(X1)

(iii) Understand more

"method" ~ "function"

```
print(type(X1.conjugate()))
print(type(complex.conjugate(X1)))
print(type(X1.conjugate))
print(type(complex.conjugate))
```

(iv) attributes: hasattr, __self__

```
X1 = complex(1,2)    # same as X1 = 1 + 2j
print(hasattr(X1,"conjugate"), hasattr(X1,"imag"))    # also try: import math with print(hasattr(math,"isclose"))
print(callable(X1.conjugate), callable(X1.imag))

X2 = X1.conjugate()
X3 = X1.conjugate.__self__
```

==========
[Exercise / Example 3] Almost every Python's things are **objects** (**isinstance**, **issubclass**)
==========

Try to digest and fill in the blanks

```
# 1) there are many objects; type, print, "hello" are objects
print(
    isinstance(type, object), # output _____
    isinstance(print, object), # output _____
    isinstance("hello", object) # output _____
)

# 2) there are many types; object, int, etc are types
print(
    isinstance(object, type), # output _____
    isinstance(int, type), # output _____
)

# 3) True is a value of bool; bool is a subclass of int, as well as a subclass of object
# actually, all classes are subclasses of object
print(
    isinstance(True, bool),    # output _____
    isinstance(True, int),     # output _____
    isinstance(True, float)    # output _____
)
print(
    issubclass(bool, int), # output _____
    issubclass(bool, object) # output _____
)

# 4)
# We can say 123 is an instance of int
# We cannot say 123 is subclass of int

# Similarly we can say object is an _____ of type
# We cannot say object is _____ of Type

# 5)
isinstance(type, object), issubclass(type, object) # output _____
isinstance(object, type), issubclass(object, type) # output _____
```

==========
[Exercise / Example 4] **complex** , **is** and **id**
==========

Use OPTMentor:

(1) Run the following. Guess what it does:

```
x1 = complex(1,2)
x2 = 1+3j
x3 = complex('1+4j')
print(x1, x1.real, x1.imag)
print(x2, x2.real, x2.imag)
print(x3, x3.real, x3.imag)
```

(2) Run the following. Guess what it does:

```
x = y = complex(1, 0)
z = complex(1, 0)
print(x == y == z == 1.0)
x_id = id(x)
y_id = id(y)
z_id = id(z)
print(x is y)    # id(x) == id(y)
print(x is z)    # id(x) != id(z)
```

```
# (3) (i)
        x = y = "abc"
        print(x is y)
        print(y is "abc")

        sum1 = x + y
        sum2 = x + "abc"

        print(sum1 == sum2)
        print(sum1 is sum2)

# (3) (ii) Note: the behaviour in OPTMentor is different from ipynb
        a = 10**10
        b = 10**10
        c = 10**100
        d = 10**100

        result1 = a is b
        result2 = c is d

        result3 = a == b
        result4 = c == d

        result1, result2, result3, result4

# (4)(i)
        # Try the following.    What do you expect and what is the output?
                x=1234567
                y=1234567
                print(id(x), id(y))
                print(x is y)

                # When you assign integers in the range of (-5, 256),
                # the result is different.

        # What about the following?
                x=123
                y=123
                print(id(x), id(y))
                print(x is y)

        # https://www.laurentluce.com/posts/python-integer-objects-implementation/
        # https://en.wikipedia.org/wiki/Interning_(computer_science)
        # https://levelup.gitconnected.com/optimization-in-python-the-interning-technique-for-improved-performance-3ff14d376176

# (4)(ii) [interning] in .jpynb: see the warning: SyntaxWarning (when using is with a literal)
        result1, result2 = 10 is 10, "abc" is "abc"
```

> **Caution**
>
> When using `is` with a literal, the behavior is not entirely predictable because
>
> - Python tries to avoid storing the same value at different locations by *interning* but
> - interning is not always possible/practical, especially when the same value is obtained in different ways.
>
> Hence, `is` should only be used for built-in constants such as `None` because there can only be one instance of each of them.

==========
[Exercise / Example 5] **File I-O**
==========

Study the following: Introduction programs for file I-O

(1) read from contact.csv

```
f = open('contact.csv') # create a file object for reading
contents = f.read()     # get the entire content
print(contents)
f.close()              # close the file
print(f.closed)         # file closed or not?
```

(2) read from contact.csv line by line; write to output.csv

```
f1 = open('contact.csv')
f2 = open('output.csv', 'w')

for line in f1:
    if line[0]>'E':
        f2.write(line)
f1.close()
f2.close()
```

try **!more contact.csv**
try **!more output.csv**

(3) using with

```
with open('contact.csv') as f1:
    with open('output.csv', 'w') as f2:
        for line in f1:
            if line[0]=='C':
                f2.write(line)

# checking
with open('contact.csv') as f1:
    with open('output.csv', 'w') as f2:
        print(f1.closed)
        print(f2.closed)
        for line in f1:

            if line[0]=='C':
                f2.write(line)
    print(f1.closed)
    print(f2.closed)
print(f1.closed)
```

(4) using 'a' for append

```
with open('contact.csv') as f1:
    with open('output.csv', 'a') as f2:
        for line in f1:
            if line[0]=='C':
                f2.write(line)

!more output_folder/new_contact.csv

next:
new_data = "Effie Douglas,galnec@naowdu.tc, (888) 311-9512"
f2.write(new_data) # doesn't work: f2 has been closed!

next:
with open('output.csv', 'a') as f2:
    f2.write(new_data)
    f2.write("\n") # why needed? ^_^
```

```
# (5) folders
        # (5.1) creating a folder
        import os
        destination = 'output_folder/new_contact.csv'
        dir_string = os.path.dirname(destination) # gets "output_folder"
        os.makedirs(dir_string) # if run the second time => Error "File exists"

        # try below:
        # os.makedirs(dir_string)
        # os.makedirs(dir_string, exist_ok=True)

        # (5.2) write to the file
        with open('contact.csv') as f1:
            with open(destination, 'a') as f2:
                for line in f1:
                    if line[0]=='C':
                        f2.write(line)

        !more output_folder/new_contact.csv

# (6) delete a file
        print(os.path.exists(destination)) # output_folder/new_contact.csv
        os.remove(destination)

# (7) delete a folder
        print(os.path.exists(dir_string)) # output_folder
        os.rmdir(dir_string) # occasionally not workingk: possible hidden things like ".ipynb_checkpoints"
        # try !ls output_folder -al


==========================================
 [Exercise / Example 6] String
==========================================
        # Study the following:

        record = "Tai Ming Chan,tmchan@cityu.edu.hk,(634) 234-7294\n"

        # learn find
        print(record.find("cityu"))
        print(record.find("@"))
        print(record.find("#"))

        # learn split
        x1=record.split(",")

        # learn join: show the data line by line
        x2="\n".join(x1)

        # learn strip, rstrip, remove whitespace (space, tab, newline)
        a1=record.split(",")[-1]
        a2=record.split(",")[-1].rstrip()

        # learn more (rsplit with maxsplit, upper), get the person's name like CHAN, Tai Man
        name = record.split(",")[0]
        parts = name.rsplit(" ",maxsplit=1)
        print(parts[1].upper() + ", " + parts[0])
```

==========
[Exercise / Example 7] We write a class called **Fruit_Box**
==========

Run the following. Guess what it does:

```
# (1) We create our own class
class Fruit_Box:
    def __init__(self, f_n, cnt):
        self.fruit_name = f_n
        self.count = cnt

    def print_all(self):
        print((self.fruit_name + " ") * self.count)

    def compare(self, another_box):
        if self.fruit_name == another_box.fruit_name:
            result = "more" if self.count > another_box.count else "not more"
            print(result)
        else:
            print("Cannot compare")


b1 = Fruit_Box("Orange", 10)
b2 = Fruit_Box("Orange", 20)
b3 = Fruit_Box("Lemon", 20)

b1.print_all()
Fruit_Box.print_all(b3)

b1.compare(b2)
b2.compare(b1)
b1.compare(b3)

# print(type(b1.print_all))
# print(type(Fruit_Box.print_all))

# help(b1.print_all)
# help(Fruit_Box.print_all)
```

```
# (2) Operator Overloading
class Fruit_Box:
    def __init__(self, f_n, cnt):
        self.fruit_name = f_n
        self.count = cnt

    def __add__(self, another_box):
        if self.fruit_name == another_box.fruit_name:
            return self.fruit_name + " x " + str(self.count + another_box.count)
        else:
            return (
                self.fruit_name + " x " + str(self.count)
                + " and "
                + another_box.fruit_name + " x " + str(another_box.count)
            )


b1 = Fruit_Box("Orange", 10)
b2 = Fruit_Box("Orange", 20)
b3 = Fruit_Box("Lemon", 20)
print(b1 + b2) # operator overloading is done!
print(b1 + b3)
```

```
# example (before learning operator overloading)

class Fruit_Box:
    def __init__(self, f_n, cnt):
        self.fruit_name = f_n
        self.count = cnt

    def print_all(self):
        print((self.fruit_name+" ")*self.count, end=" ")

    def sum(self, another_box):
        if self.fruit_name == another_box.fruit_name:
            return self.fruit_name+" x "+str(self.count+another_box.count)
        else:
            return (self.fruit_name+" x "+str(self.count)
                + " and "
                + another_box.fruit_name+" x "+str(another_box.count)
                )

b1 = Fruit_Box("Orange", 10)
b2 = Fruit_Box("Orange", 20)
b1.print_all()
result = b1.sum(b2)
```

See More: **Dispatch on type** and **Data-directed programming** (Objects.ipynb)

==========

See additional contents from **Objects.ipynb**