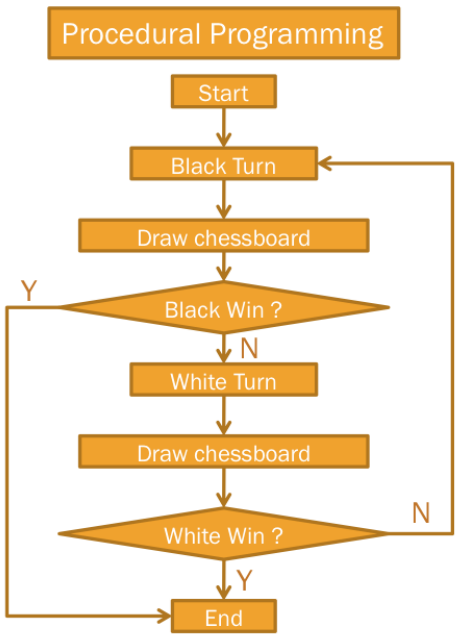
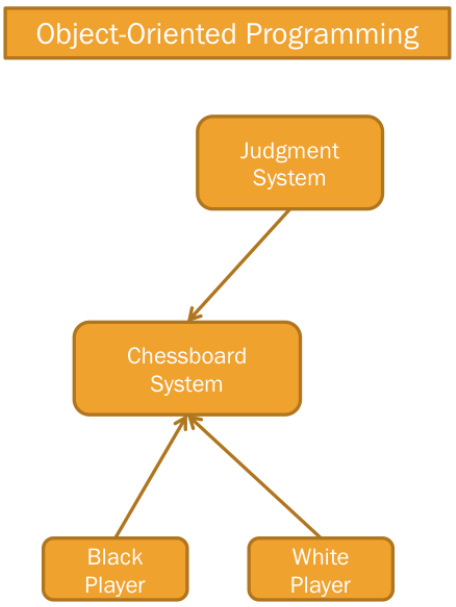
Lecture 6 Introduction to Object Oriented Programming (2021.10.19 & 21)

§1. Object-oriented Programming





§2 Object

1. Object

1° An object represents an entity in the real world that can be distinctly identified.

2° An object has a unique identity, state, and behaviours.

2. Key elements of object

1° Unique identity

① like a person's ID

2° State

① a set of data fields with their current values (variables)

② also know as properties / attributes

3° Behavior

① defined by a set of methods (functions)

Examples:

CEO

- name
- gender
- age
- company
- buildStrategy()
- makePresentations()

name	Tim Cook
gender	Male
age	57
company	Apple

buildStrategy(Apple);
makePresentation(iPad);

name	Elon Musk
gender	Male
age	47
company:	Tesla

buildStrategy(Tesla);
makePresentation(Mode X);

Car

- brand
- mode
- color
- drive()
- brake()
- playMusic()

brand	Audi
mode	A4
color	red

drive(speed);
brake();
playMusic(CD);

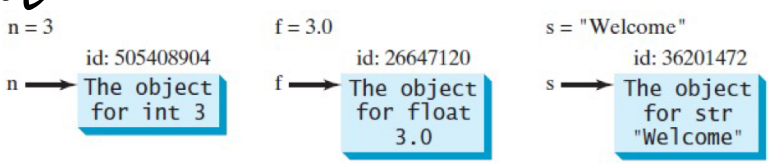
brand	BMW
mode	328i
color	white

drive(speed);
brake();
playMusic(bluetooth);

3 Variable

1° In python, everything is an object.

2° A variable in Python is actually a reference to an object.



4. The id of an object

1° a unique integer

2° automatically assigned by Python when the program is executed

3° will not be changed during the execution of the program

4° id( ) function

5. The type for the object

1° determined by Python according to the value of the object.

2° type( ) function

b. Methods

1° You can perform operations on an object

2° The operations are defined using functions.

3° The functions for the objects are called methods in Python.

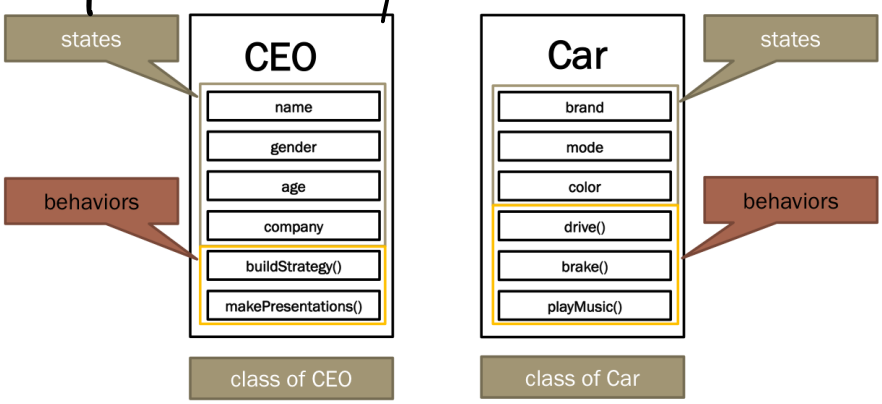
4° Methods can only be invoked from a specific object.

§3 Class

1. Class

1° Classes are constructs defining objects of the same type

2° A class is a contract: also sometimes called a template or blueprint



Example:

- Class name: Human
- Data fields: Height, body weight, IQ, EQ, education level ...
- Methods:

  Eat()

  Sleep()

  Marry()

  Work()

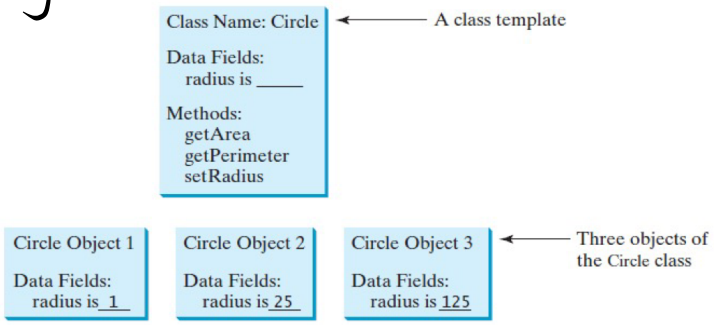
  .....

2. Object V.S. Class

1° An object is an instance (实例) of a class, and you can create many instances of a class.

2° Creating an instance of class is referred to as instantiation

3° "object" = "instance"



3. Define class

1° Python uses the following syntax to define a class

```
class ClassName:
    initializer
    methods
```

- a special method __init__()
- to initialize a new object's state when it is created

2° Data fields are also called instance variables, because each object (instance) has a specific value for a data field.

3° Methods are also called instance methods, because a method which is invoked by an object (instance) will perform actions based on the data fields of that object.

EXAMPLE

All methods, including the initializer, have the first parameter self

This parameter refers to the object that invokes the method

```
class Car:
    def __init__(self, brand="Audi", mode="A4", color="white"):
        self.brand = brand
        self.mode = mode
        self.color = color
    def drive(self):
        print("A", self.color, self.brand, self.mode, "is driving...")
    def brake(self):
        print("A", self.color,self.brand, self.mode, "is braking...")
    def playMusic(self):
        print("A", self.color,self.brand, self.mode, "is playing music...")
```

The self parameter in the __init__() method is automatically set to reference the object that was just created

```
myCar = Car()
myCar.drive()
```

Create new Car object

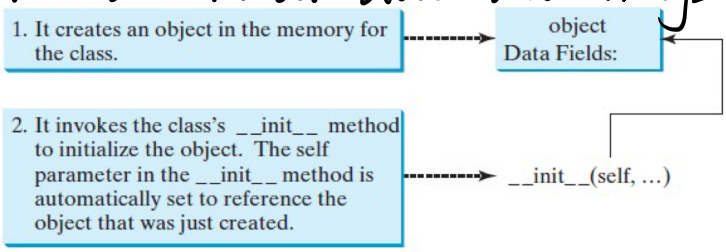
Invoke myCar's drive method

Object member access operator
- dot operator (.)
- Access the object's data fields
- Invoke its methods

4. Constructing objects

1° You can create objects from defined class with a constructor

2° The constructor does two things:



3° Constructor arguments

① The argument of constructor match the parameters in the __init__() method without self

② If the initializer in the class has a default values, then the constructor without arguments will assign the default values to data fields.

- myCar = Car()
- myCar = Car("Audi", "A4", "white")

```
def main():

        myCar1 = Car()
        myCar1.drive()

        myCar2 = Car("BMW", "328i", "red")
        myCar2.brake()

        myCar3 = Car("Tesla", "Model X", "grey")
        myCar3.playMusic()

        myCar2.color = "blue"        # modify a property
        myCar2.drive()

main()
```

5. Slope of self

1° The slope of an instance variable is the entire class once it is created.

2° You can also create local variables in a method.

3° The slope of a local variable is within the method.

```
def ClassName:

    def __init__(self, ...):
        self.x = 1  # Create/modify x
        ...

    def m1(self, ...):
        self.y = 2  # Create/modify y
        ...
        z = 5 # Create/modify z
        ...

    def m2(self, ...):
        self.y = 3  # Create/modify y
        ...
        u = self.x + 1 # Create/modify u
        self.m1(...) # Invoke m1
```

Scope of self.x and self.y

Scope of z

```
class ClassName:

    def __init__(self):
        self.x = 1

    def m1(self):
        self.y = 2
        z=5
        print('y=',self.y,'z=',z)

    def m2(self):
        self.y = 3
        z = self.x+1
        print('y=',self.y,'z=',z)
        self.m1()

myObj = ClassName()
myObj.m2()
```

§4 Module and Import System

1. Module and import

1° Put definition in a file and use predefined code later.
Such a file is called a module

2° Definitions from a module can be imported
① into other module
② into the main module
(1) in a script executed at the top level and in calcator mode
(2) the collection of variables that you have access to.

Examples:

mymodule.py

```
def compare(a, b):
        if a>b:
                print(b, a)
        else:
                print(a, b)

def compare2(a, b):
        if a>b:
                return [b, a]
        else:
                return [a, b]
```

test.py

```
import mymodule

mymodule.compare(3,2)
result = mymodule.compare2(2,1)
print(result)
```

```
2, 3
[1,2]
```

You can assign a function to a local name:

test2.py

```
import mymodule

compare = mymodule.compare
result = compare(2,1)
print(result)
```

mymodule.py

```
def compare(a, b):
        if a>b:
                print(b, a)
        else:
                print(a, b)

def compare2(a, b):
        if a>b:
                return [b, a]
        else:
                return [a, b]
```

test3.py

```
from mymodule import compare, compare2

result = compare2(100,4)
print(result)
```

```
[4,100]
```

test4.py

```
from mymodule import *  (all)

result = compare(4,1)
```

```
1, 4
```

2. The module search path *

When a module is imported, the interpreter searches:

1° built-in module

2° In a list of directories given by the variable sys.path from these locations:

① directory containing the input script, or the current directory when no file is specified.

② PYTHONPATH
a list of directory names, with the same syntax as the shell variable PATH

③ installation-dependent default

3. Math module: access to the mathematical functions

Trigonometric functions

`math.`**`acos`**`(x)`
Return the arc cosine of *x*, in radians.

`math.`**`asin`**`(x)`
Return the arc sine of *x*, in radians.

`math.`**`atan`**`(x)`
Return the arc tangent of *x*, in radians.

`math.`**`cos`**`(x)`
Return the cosine of *x* radians.

`math.`**`sin`**`(x)`
Return the sine of *x* radians.

`math.`**`tan`**`(x)`
Return the tangent of *x* radians.

Number-theoretic and representation functions

`math.`**`ceil`**`(x)`
Return the ceiling of *x*, the smallest integer greater than or equal to *x*. If *x* is not a float, delegates to `x.__ceil__()`, which should return an `Integral` value.

`math.`**`copysign`**`(x, y)`
Return *x* with the sign of *y*. `copysign` copies the sign bit of an IEEE 754 float, `copysign(1, -0.0)` returns *-1.0*.

`math.`**`fabs`**`(x)`
Return the absolute value of *x*.

`math.`**`factorial`**`(x)`
Return *x* factorial. Raises `ValueError` if *x* is not integral or is negative.

`math.`**`floor`**`(x)`
Return the floor of *x*, the largest integer less than or equal to *x*. If *x* is not a float, delegates to `x.__floor__()`, which should return an `Integral` value.

Power and logarithmic functions

`math.`**`exp`**`(x)`
Return `e**x`.

`math.`**`log`**`(x[, base])`
Return the logarithm of *x* to the given *base*. If the *base* is not specified, return the natural logarithm of *x* (that is, the logarithm to base *e*).

`math.`**`pow`**`(x, y)`
Return `x` raised to the power `y`.

`math.`**`sqrt`**`(x)`
Return the square root of *x*.

Constants

`math.`**`pi`**
The mathematical constant *pi*.

`math.`**`e`**
The mathematical constant *e*.

Module *cmath*:
Complex number versions of many of these functions

EXAMPLE

```
import math

class Circle:
    # Construct a circle object
    def __init__(self, radius = 1):
        self.radius = radius

    def getPerimeter(self):
        return 2 * self.radius * math.pi

    def getArea(self):
        return self.radius * self.radius * math.pi

    def setRadius(self, radius):
        self.radius = radius
```

```
>>> circle1 = Circle()
>>> circle1.radius
1
>>> circle1.getPerimeter()
6.283185307179586
>>> circle1.getArea()
3.141592653589793
>>> circle1 = Circle(2)
>>> circle1.radius
2
>>> circle1.radius = 10
>>> circle1.getArea()
314.1592653589793
```

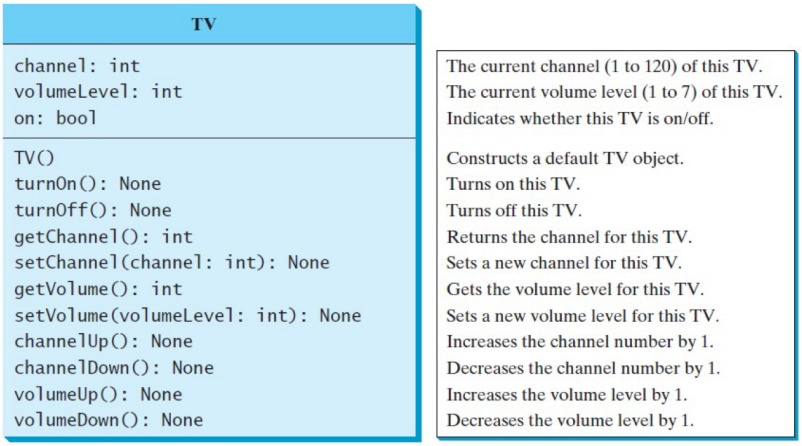
EXAMPLE: USING SELF-DEFINED CIRCLE CLASS

```
def main():
    # Create a circle with radius 1
    circle1 = Circle()
    print("The area of the circle of radius",
        circle1.radius, "is", circle1.getArea())

    # Create a circle with radius 25
    circle2 = Circle(25)
    print("The area of the circle of radius",
        circle2.radius, "is", circle2.getArea())

    # Create a circle with radius 125
    circle3 = Circle(125)
    print("The area of the circle of radius",
        circle3.radius, "is", circle3.getArea())

    # Modify circle radius
    circle2.radius = 100 # or circle2.setRadius(100)
    print("The area of the circle of radius",
        circle2.radius, "is", circle2.getArea())

main() # Call the main function
```

RESULTS:

```
The area of the circle of radius 1.0 is 3.141592653589793
The area of the circle of radius 25.0 is 1963.4954084936207
The area of the circle of radius 125.0 is 49087.385212340516
The area of the circle of radius 100.0 is 31415.926535897932
```

EXAMPLE: TV CLASS




```
class TV:
    def __init__(self):
        self.channel = 1  # Default channel is 1
        self.volumeLevel = 1  # Default volume level is 1
        self.on = False  # Initially, TV is off

    def turnOn(self):
        self.on = True

    def turnOff(self):
        self.on = False

    def getChannel(self):
        return self.channel

    def setChannel(self, channel):
        if self.on and 1 <=      channel <= 120:
            self.channel = channel

    def getVolumeLevel(self):
        return self.volumeLevel

    def setVolume(self, volumeLevel):
        if self.on and \
             1 <=      volumeLevel <= 7:
            self.volumeLevel = volumeLevel

    def channelUp(self):
        if self.on and self.channel < 120:
            self.channel += 1

    def channelDown(self):
        if self.on and self.channel > 1:
            self.channel -= 1

    def volumeUp(self):
        if self.on and self.volumeLevel < 7:
            self.volumeLevel += 1

    def volumeDown(self):
        if self.on and self.volumeLevel > 1:
            self.volumeLevel -= 1
```

```
from TV import TV

def main():
    tv1 = TV()
    tv1.turnOn()
    tv1.setChannel(30)
    tv1.setVolume(3)

    tv2 = TV()
    tv2.turnOn()
    tv2.channelUp()
    tv2.channelUp()
    tv2.volumeUp()

    print("tv1's channel is", tv1.getChannel(),
        "and volume level is", tv1.getVolumeLevel())
    print("tv2's channel is", tv2.getChannel(),
        "and volume level is", tv2.getVolumeLevel())

main() # Call the main function
```

```
tv1's channel is 30 and volume level is 3
tv2's channel is 3 and volume level is 2
```

§5 String Class

1. Functions to deal with String

Python has a number of string functions which are built-into String class

1° We invoke them with object member access operator/ .(dot) operator

2° We call the function in the String object.

FUNCTIONS IN STRING CLASS

```
>>> stuff = 'hello world'
>>> type(stuff)
<class 'str'>
>>> dir(stuff)
['__add__', '__class__', '__contains__', '__delattr__', '__dir__', '__doc__', '_
_eq__', '__format__', '__ge__', '__getattribute__', '__getitem__', '__getnewargs
__', '__gt__', '__hash__', '__init__', '__iter__', '__le__', '__len__', '__lt__'
, '__mod__', '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__r
epr__', '__rmod__', '__rmul__', '__setattr__', '__sizeof__', '__str__', '__subcl
asshook__', 'capitalize', 'casefold', 'center', 'count', 'encode', 'endswith', '
expandtabs', 'find', 'format', 'format_map', 'index', 'isalnum', 'isalpha', 'isd
ecimal', 'isdigit', 'isidentifier', 'islower', 'isnumeric', 'isprintable', 'issp
ace', 'istitle', 'isupper', 'join', 'ljust', 'lower', 'lstrip', 'maketrans', 'pa
rtition', 'replace', 'rfind', 'rindex', 'rjust', 'rpartition', 'rsplit', 'rstrip
', 'split', 'splitlines', 'startswith', 'strip', 'swapcase', 'title', 'translate
', 'upper', 'zfill']
```

HTTPS://DOCS.PYTHON.ORG/3/LIBRARY/STDTYPES.HTML#STRING-METHODS

4.7.1. String Methods

Strings implement all of the common sequence operations, along with the additional methods described below.

Strings also support two styles of string formatting, one providing a large degree of flexibility and customization (see `str.format()`, Format String Syntax and String Formatting) and the other based on C `printf` style formatting that handles a narrower range of types and is slightly harder to use correctly, but is often faster for the cases it can handle (printf-style String Formatting).

The Text Processing Services section of the standard library covers a number of other modules that provide various text related utilities (including regular expression support in the `re` module).

`str.` **`capitalize`**`()`
Return a copy of the string with its first character capitalized and the rest lowercased.

`str.` **`casefold`**`()`
Return a casefolded copy of the string. Casefolded strings may be used for caseless matching.

Casefolding is similar to lowercasing but more aggressive because it is intended to remove all case distinctions in a string. For example, the German lowercase letter `'ß'` is equivalent to `"ss"`. Since it is already lowercase, `lower()` would do nothing to `'ß'`; `casefold()` converts it to `"ss"`.

The casefolding algorithm is described in section 3.13 of the Unicode Standard.

New in version 3.3.

`str.` **`center`**`(`*width*`[, `*fillchar*`])`
Return centered in a string of length *width*. Padding is done using the specified *fillchar* (default is an ASCII space). The original string is returned if *width* is less than or equal to `len(s)`.

`str.` **`count`**`(`*sub*`[, `*start*`[, `*end*`]])`
Return the number of non-overlapping occurrences of substring *sub* in the range [*start*, *end*]. Optional arguments *start* and *end* are interpreted as in slice notation.

2. find()

1° We can use the find() function to search for a substring in a string.

2° find() finds the first occurence of the target sub-string.

3° If the sub-string is not found, it returns -1.

4° Important: the string position starts from 0.

EXAMPLE

```
>>> data = 'From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2016'
>>> atpos = data.find('@')
>>> print(atpos)
21
>>> sppos = data.find(' ',atpos)
>>> print(sppos)
31
>>> host = data[atpos+1:sppos]
>>> print(host)
uct.ac.za
```

substring, start position, end position

MAKING EVERYTHING UPPER OR LOWER CASE

- You can convert a string into upper case or lower case

```
>>> myStr = 'I am the one, I will beat Matrix'
>>> newStr = myStr.upper()
>>> print(newStr)
I AM THE ONE, I WILL BEAT MATRIX
>>> newStr = myStr.lower()
>>> print(newStr)
i am the one, i will beat matrix
```

- Hint: often when we use find() to find a substring, we convert the original string into lower case first, so that we don't need to worry about case

3. replace()

1° The replace() function is like a "search and replace" operation in a word processor.

2° It replace all occurrences of the search string with the replacement string.

```
>>> greet = 'Hello, Bob'
>>> newStr = greet.replace('Bob', 'Jane')
>>> print(newStr)
Hello, Jane
>>> newStr = greet.replace('o','X')
>>> print(newStr)
HellX, BXb
>>> newStr = greet.replace('z','X')
>>> newStr
'Hello, Bob'
```

4. Strip()

1° When we need to remove whitespaces at the beginning and/or end of a string.

2° lstrip() and rstrip() to the left and right only.

3° strip() removes both beginning and ending whitespace.

```
>>> greet = '   Hello Bob   '
>>> greet.lstrip()
'Hello Bob   '
>>> greet.rstrip()
'   Hello Bob'
>>> greet.strip()
'Hello Bob'
```

5. Startwith()

startwith() function checks whether a string is starting with

a given string.

```
>>> line = 'Please submit your application'
>>> line.startswith('Please')
True
>>> line.startswith('p')
False
```

b. String formatting operators: old style and new style

Old	`'%s %s' % ('one', 'two')`
New	`'{} {}'.format('one', 'two')`
Output	`one two`
Old	`'%d %d' % (1, 2)`
New	`'{} {}'.format(1, 2)`
Output	`1 2`