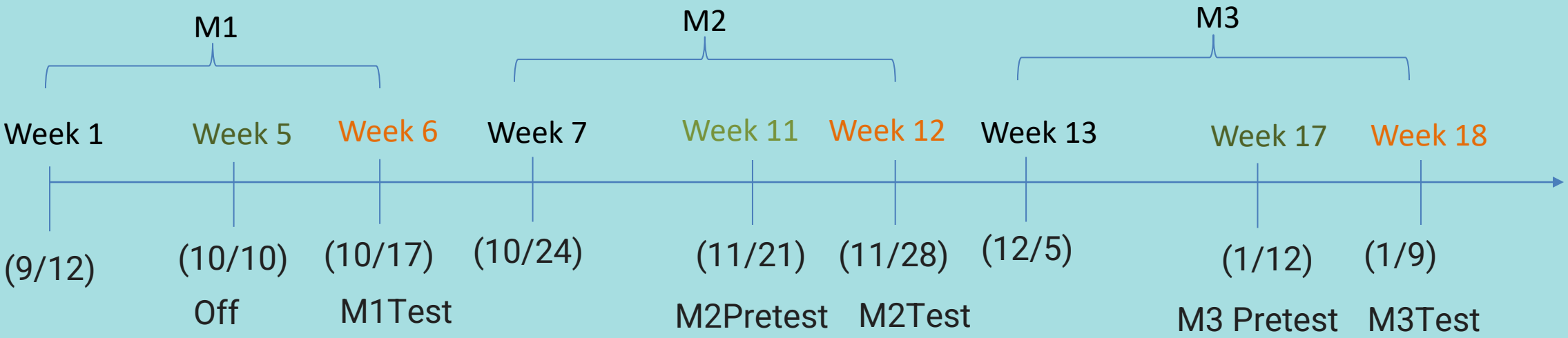
Fundamental Programming Course

Week 15

Huseh-Ting Chu@Asia University, 2022

Schedule

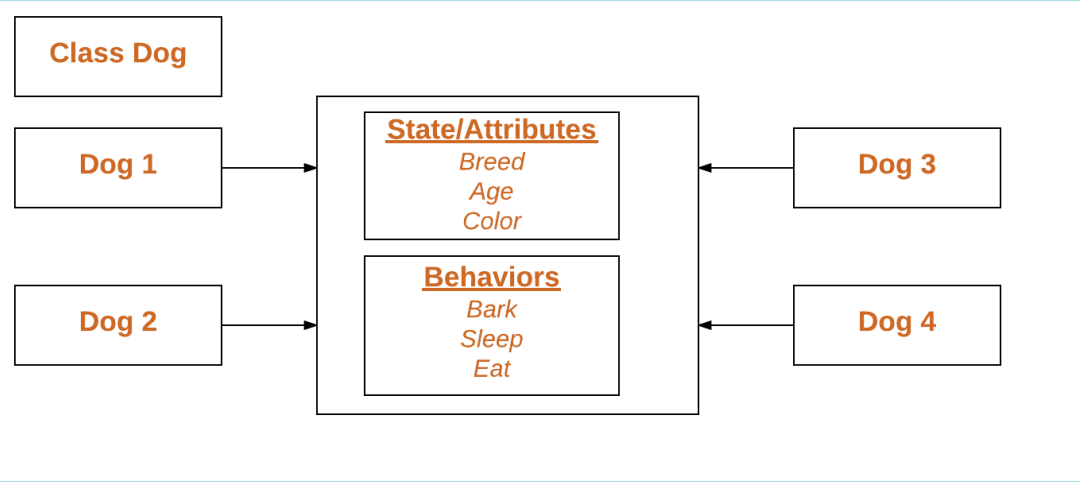

M1
M2
M3
Week 1
Week 5
Week 6
Week 7
Week 11
Week 12
Week 13
Week 17
Week 18
(9/12)
(10/10)
Off
(10/17)
M1Test
(10/24)
(11/21)
M2Pretest
(11/28)
M2Test
(12/5)
(1/12)
M3 Pretest
(1/9)
M3Test

Syllabus

- W1-Python Introduction and Programming Tools
- W2-Variables and Operations
- W3-Loop and formatted output
- W4-Condition and Containers
- W5-String and built-in functions
- W6-M1 test
- W07-Dictionary Container
- W08-File I/O
- W09-Function
- W10-Advanced flow control
- W11-Advanced operations and generators
- W12-M2 test

- W13-Advanced functions (definitions and calls, Recursion)
- W14-Class fundamentals (classes, objects, properties, constructors, methods)
- W15-Advanced Classes (Static methods, class Methods and class decorators)
- W16-Modules and Packages
- W17-Advanced programming(Argparse and Venv)
- W18-M3 test

Class definition




```
class Dog:
    # Class Variable
    animal = 'dog'
    # The init method or constructor
    def __init__(self, breed):
        # Instance Variable
        self.breed = breed
    # Adds an instance variable
    def setColor(self, color):
        self.color = color
    # Retrieves instance variable
    def getColor(self):
        return self.color

# Driver Code
Rodger = Dog("pug")
Rodger.setColor("brown")
print(Rodger.getColor())
```

Advanced Classes

Python 三大器:
迭代器 iterator
生成器 generator (comprehension)
裝飾器 decorator

- Various methods of class
- Topic 1- instance method/category method
- Topic 2- static method
- Topic 3- decorator

Topic 1-instance methods

- derived class

```
class Person:
  def __init__(self, fname, lname):
    self.firstname = fname
    self.lastname = lname

  def printname(self):
    print(self.firstname, self.lastname)


class Student(Person):
  pass
```

super()

Python 還有一個 super() 函數，可以讓子類繼承其父類的方法和屬性

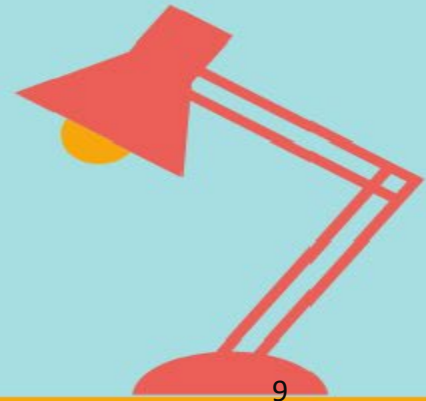
```
class Person:
  def __init__(self, fname, lname):
    self.firstname = fname
    self.lastname = lname

  def printname(self):
    print(self.firstname, self.lastname)

class Student(Person):
  def __init__(self, fname, lname):
    super().__init__(fname, lname)
```

Topic 1- class method

```
class Person:

    def __init__(self, name):
        self.name = name

    def greet(self):
        print('Hello, my name is {}.'.format(self.name))

    @classmethod
    def info(cls):
        print('調用了類方法')
```

Topic 1- class method

```
Person.calculate(1, 2)

person_c = Person('Pang Hu')
person_c.calculate(1, 2)
```



Topic 2- static method

```
class Person:
    def __init__(self, name):
        self.name = name
    def greet(self):
        print('Hello, my name is {}.'.format(self.name))

    @classmethod
    def info(cls):
        print('調用了類方法')

    @staticmethod
    def calculate(a, b):
        print('調用靜態方法計算兩數之和')
        print('{}+{}={}'.format(a, b, a+b))
```

Topic 2- usage of static method

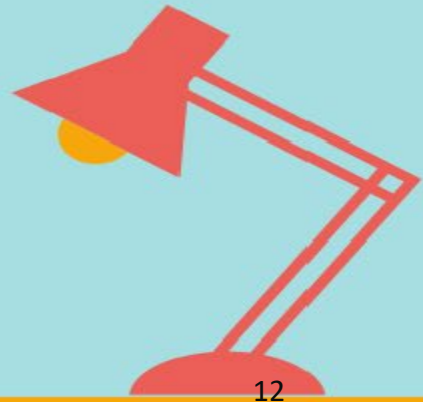
```
Person.info()

person_b = Person('Xiao Hong')
person_b.info()
```

Topic 3- decorator

Because the basic syntax of Python is simple, easy to use and concise, the syntax of Python becomes more and more complicated. A decorator is a function that returns another function, usually it is applied as a function transformation using the @wrapper syntax. Common examples of decorators are classmethod() and staticmethod().
The decorator syntax is just syntactic sugar.

```
def f(...):
    ...
f = staticmethod(f)

@staticmethod
def f(...):
    ...
```

```
def my_decorator(func):
    print('In A')
    return func

@my_decorator
def my_func():
    print('In B')

my_func()
```



Thanks!

Q&A