Week 2

- [ ] 所有笔记都要打一遍
- [ ] ppt-introduce decorator (@)
- [ ] int float 正负储存
- [ ] 列表 99乘法表+print()格式化输出

▼ Decorator

decorator本质是函数闭包（function closure）的语法糖（syntactic sugar）

- 函数闭包（function closure）

  闭包函数本身是一个函数，他的参数以及返回值都是函数

version 1:

```
import time
'''
函数逻辑（查找奇数）和辅助功能（记录时间）耦合在一起
是否能将功能分开？
'''

def print_odds():

  '''
  print odd numbers between 0 to 100 and count the execute time
  '''

  start_time = time.clock()

  for i in range(100):
    if i % 2 == 1:
      print(i)
  end_time = time.clock()
  print("it takes {} s to find all the olds".format(end_time - start_time)

if __name__ == '__main__':
  print_odds()
```

version 2:

```
import time
'''
将辅助功能（记录时间）抽离成一个辅助函数count_time
```

```
在辅助函数中调用主要功能函数（查找奇数）print_odds
解耦，但辅助函数调用主要功能函数，不方便
是否可以在调用主要函数时完成时间统计？
'''

def count_time(func): #一切函数都可以当作object来看，一个函数的参数与返回值都可以是函数
'''
count the time
'''
  start_time = time.clock()
  func()
  end_time = time.clock()
  print("it takes {} s to find all the olds".format(end_time - start_time)

def print_odds():

  '''
  print odd numbers between 0 to 100
  '''
  for i in range(100):
    if i % 2 == 1:
      print(i)



if __name__ == '__main__':
  count_time(print_odds)
```

version 3 - 闭包

❗传入的函数与返回的函数关系:返回函数是功能增强了的原函数

```
import time
'''
但是需要显示进行闭包增强
能否调用print_odds就可以直接增强，隐式增强
'''

def print_odds():

  '''
  print odd numbers between 0 to 100
  '''
  for i in range(100):
    if i % 2 == 1:
      print(i)

#闭包函数
def count_time_wrapper(func):
'''
闭包，用于增强函数func:给函数func增加统计时间的功能
'''
  def improved_func():
    start_time = time.clock()
    func()
    end_time = time.clock()
    print("it takes {} s to find all the olds".format(end_time - start_time)

  return improved_func
```

```
if __name__ == '__main__':
  print_odds =  count_time_wrapper(print_odds)
  print_odds()
```

version 4：decorator

结构：

```
import time

def count_time_wrapper(func):...

# @后加闭包函数的函数名
# 作用：在第一次调用函数的时候进行函数增强
@count_time_wrapper
def print_odds():...

if __name__ == '__main__':
  # print_odds =  count_time_wrapper(print_odds)
  print_odds()
```

```
import time

def count_time_wrapper(func):
'''
闭包，用于增强函数func:给函数func增加统计时间的功能
'''
  def improved_func():
    start_time = time.clock()
    func()
    end_time = time.clock()
    print("it takes {} s to find all the olds".format(end_time - start_time)

  return improved_func


@count_time_wrapper
def print_odds():
  '''
  print odd numbers between 0 to 100
  '''
  for i in range(100):
    if i % 2 == 1:
      print(i)

if __name__ == '__main__':
  # 装饰器等价于在第一次调用函数时执行以下语句：
  # print_odds =  count_time_wrapper(print_odds)
  print_odds()
```

- 语法糖

计算机语言中添加的某种语法，对语言功能没有影响，但更方便使用

  - 语法糖没有增加新功能

```
@count_time_wrapper
def print_odds():

print_odds()
```

等价于：

```
def print_odds():

print_odds =  count_time_wrapper(print_odds)
print_odds()
```

装饰器在第一次调用被装饰函数时增强

❗在函数被调用之前增强

❗只增强一次，后续在调用时调用的都是已增强的函数

❗❗❗对于含有返回值的函数，调用闭包增强后，不能成功返回，但是成功增强了返回值

❗❗❗对于含有参数的函数，调用闭包增强后，不能成功接收参数，显示闭包函数是没有该参数的

❗❗❗原函数有返回值且有参数，但增强后的函数没有返回值与参数

```
  #增强函数的返回值应该是原函数的返回值
def improved_func(*args, **kwargs): #增强函数应该把所有接受到的参数传给原函数
    start_time = time.clock()
    ret = func()
    end_time = time.clock()
    print("it takes {} s to find all the olds".format(end_time - start_time)
return ret
```

▼ 99乘法表

```
for i in range(1,10):
    for j in range(1, i+1):
        print(f'{j} * {i} = {i*j}', end =" " )
# print('{} * {} = {}'.format(j, i, j*i), end = " ")
    print(" ")
```

```
# 列表
print('\n'.join([' '.join(['%s*%s=%-2s' % (y, x, x*y) for y in range(1, x+1)]) for x in range(1, 10)]))
```

▼ 类型表示

- int
  - 类型存储范围

    python2：整数的位数为64位（默认系统为64位），取值范围为-2**63～2**63-1

    python3：理论上长度是无限的（只要内存足够大）
  - 负数使用补码表示

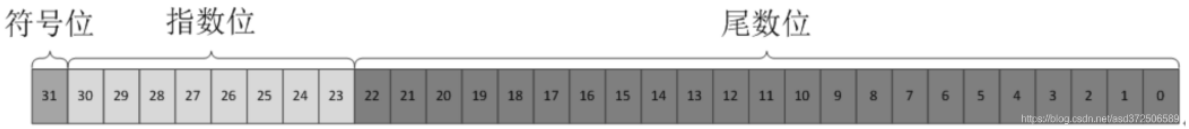
    补码正是基于反码的“-0”问题诞生

    补码的计算方法是：正数和+0的补码是其原码，负数则先计算其反码，然后反码加上1，得到补码。

- float

  float与int一样都占了四个字节也就是32个比特位。其中第一位代表正负，第2-9这八位代表指数，与后23位的底数。

  如图为单精度存储方式：



符号位（1bit）：0位正数，1位负数。

指数位（8bit）：指数位真正的取值范围是-127-127。

尾数位（23bit）

- double

double占了8个字节，64bit。

符号位（1bit）：0位正数，1位负数。

指数位（11bit）：指数范围为2^10（-1023-1024）

尾数位（底数位）（52bit）：记录科学计数法中的底数部分。