



# 实验7 函数1

ANDYWWW

LAST MODIFIED: 23APR 2019

1. 合法标识符：定义一个函数 CheckId(), 函数从\_\_main\_\_模块中接收参数 s, 判断 s 是否为合法标识符

---

思路：

- (1) 明确合法标识符的特征：首字符必须是下划线或字母，  
其他字符必须是下划线、字母或数字
- 参考实验5第2题
- (2) 首先判断首字符，然后利用切片循环后面字符  
进一步判断

1. 合法标识符：定义一个函数 CheckId(), 函数从\_\_main\_\_模块中接收参数 s, 判断 s 是否为合法标识符

答案：

```
def CheckId(s):  
    alphas = 'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ_'  
    nums = '0123456789'  
    firstChar = s[0]  
    if firstChar not in alphas:  
        print('Error. First char must be alphas or _.')  
    else:  
        otherChar = s[1:]  
        alphasnums = alphas + nums  
        for c in otherChar:  
            if c not in alphasnums:  
                print('Error. Other chars must be alphas number or _.')  
                break  
        else:  
            print('Valid identifier.')  
  
if __name__ == "__main__":  
    CheckId(input())
```

可以利用chr函数循环生成，也可后续利用类似实验5第2题的方法判断

判断两个表达式的值是否相等用"=="  
赋值用"=" "  
判断两个变量的id是否相等用"is"!

也可以利用flag进行判断，输出部分放在循环外。  
也可以直接用return，这样就不需要break，但是需要在主函数内print(CheckId(xxx))

1. 合法标识符：定义一个函数 CheckId(), 函数从\_\_main\_\_模块中接收参数 s, 判断 s 是否为合法标识符

```
x = int(input("Input the number: "))
```

```
def f(x):
```

```
    if x == 1:
```

```
        return 1
```

```
        print("x = {}".format(x))
```

```
    else:
```

```
        return 0
```

```
        print("x = {}".format(x))
```

函数内的变量（除了全局变量）只在函数内起作用，出了函数便会被“销毁”。且函数对外变量有“屏蔽”作用，比如这里函数中x不再是前面input语句的x

return后的语句不会再执行，因此这两个print语句是没用的。

如果f(xxx)为一个函数，只要代码内有f(xxx)出现就会执行这个函数。如果运行print(f(xxx))，如果这个函数有返回值（return），则print其返回值；如果没有返回值则会输出None

一般而言，将import部分全部放到代码最开始，自定义函数部分放到主函数之前，较理想的代码格式是：

```
import xxx
import xxx as xxx
```

```
def f(xxx):
```

```
    xxxx
```

```
def g(xxx):
```

```
    xxxx
```

```
if __name__ == "__main__":
```

```
    xxxx
```

左侧的代码格式不推荐！

1. 合法标识符：定义一个函数 CheckId(), 函数从\_\_main\_\_模块中接收参数 s, 判断 s 是否为合法标识符

Ex1. 定义函数find\_str(xxx), 使得其能实现类似字符串的find方法, 即寻找第一次出现子串s1的索引, 如果找不到返回-1 (要求: 不准使用字符串方法以及切片)

```
def find_str(string,sub):  
    for i in range(len(string)-len(sub)+1):  
        flag = 1  
        for j in range(len(sub)):   
            if string[i+j] != sub[j]:  
                flag = 0  
                break  
        if flag:  
            return i  
    return -1
```

防止下标越界

找到所有子串

跳出循环代表没找到

```
def find_str(string,sub):  
    index_lst = []  
    for i in range(len(string)-len(sub)+1):  
        flag = 1  
        for j in range(len(sub)):   
            if string[i+j] != sub[j]:  
                flag = 0  
                break  
        if flag:  
            index_lst.append(i)  
    return index_lst
```

1. 合法标识符：定义一个函数 CheckId(), 函数从\_\_main\_\_模块中接收参数 s, 判断 s 是否为合法标识符

Ex2. 定义函数 find\_replace\_str(xxx), 使得其能实现类似字符串的replace方法, 替换string中的所有子字符串old为new (要求: 不准使用字符串方法以及切片)

```
def find_replace_str(string,old,new):
```

```
    new_string = ""
```

```
    i = 0
```

```
    while i < len(string):
```

i用来控制原字符串

```
        flag = 1
```

```
        for j in range(len(old)):
```

```
            if i+j >= len(string):
```

```
                flag = 0
```

```
                break
```

```
            if string[i+j] != old[j]:
```

```
                flag = 0
```

```
                break
```

下标越界或对应元素不相等都代表无法找到对应子串

```
    if flag:
```

```
        new_string += new
```

```
        i += len(old)
```

```
        print(i)
```

```
    else:
```

```
        new_string += string[i]
```

```
        i += 1
```

```
    return new_string
```

1. 合法标识符：定义一个函数 CheckId(), 函数从\_\_main\_\_模块中接收参数 s, 判断 s 是否为合法标识符

Ex3. 定义函数split\_space\_str(xxx), 使得其能实现类似字符串的split方法, 这里只要求实现以空格为分割, 连续的空格视为一个空格 (要求: 不准使用字符串方法以及切片)

```
def split_space_str(string):  
    result_lst = []  
    split_str = ""  
    for i in range(len(string)):  
        if string[i] == " "  
            if split_str:  
                result_lst.append(split_str)  
                split_str = ""  
            else:  
                split_str += string[i]  
    if split_str:  
        result_lst.append(split_str)  
    return result_lst
```

split\_str用来储存目前分割出来的字符串

如果碰到空格, 有两种可能: ① 前面有非空格字符串, 此时需要将split\_str添加到结果的列表中而后再初始化; ② 前面是空格, 此时根据①以及最初初始化, split\_str为空, 不需要做任何操作即可

如果字符串不以空格结尾, 则会少append一个split\_str因此需要在循环后多加一个append。

Ex4. 定义函数split\_str(xxx), 使得其能实现字符串的split方法, 这里多个分隔符不视为一个 (相较于Ex3, 此题更简单)



1. 合法标识符：定义一个函数 CheckId(), 函数从\_\_main\_\_模块中接收参数 s, 判断 s 是否为合法标识符

Ex5. 给出一个由A和X组成的串, 统计得分。每个A的得分为目前连续出现的A的个数, X的得分为0。

法1:

```
string = input()
s, s_all = 0, 0
for ch in string:
    if ch == 'A':
        s += 1
        s_all += s
    else:
        s = 0
print(s_all)
```

s用来标记A连续出现的次数, 只要碰到X把s重置成0即可, 此方法与Ex3相似

法2:

```
lst = input().split('X')
grade = 0
for i in lst:
    if i:
        c = len(i)
        grade += ((c + 1) * c) / 2
print(int(grade))
```

In[3]: "AAAXXAAXAAAXXX".split("X")

Out[3]: ['AAA', '', 'AA', 'AAA', '', '', '']

计算列表元素即连续出现的A, 利用等差数列求和即可

A	A	X	A	A	A	X	A	A	A
1	2	0	1	2	3	0	1	2	3



2. 验证命题：如果一个三位整数是37的倍数，则这个整数循环左移后得到的另两个数也是37的倍数

---

思路：

循环数字从100-999，利用数学方法或切片方法算出左

移得到的两个数，验证它们是不是也是37的倍数

如果所有的37的倍数都满足条件，则说明命题成立；只要有一个不满足条

件则不成立（利用flag）

2. 验证命题：如果一个三位整数是37的倍数，则这个整数循环左移后得到的另两个数也是37的倍数

答案：

```
def thirty_seven(num):  
    num = (num % 100) * 10 + (num // 100)  
    if num % 37 != 0:  
        return False  
    num = (num % 100) * 10 + (num // 100)  
    if num % 37 != 0:  
        return False  
    return True
```

将某几位向左移动n位乘以 $10^n$ 即可

```
if __name__ == "__main__":  
    flag = True  
    for num in range(100,1000):  
        if num % 37 == 0:  
            if not thirty_seven(num):  
                flag = False  
                break  
    if flag:  
        print("It's a true proposition.")  
    else:  
        print("It's a false proposition.")
```

## 2. 验证命题：如果一个三位整数是37的倍数，则这个整数循环左移后得到的另两个数也是37的倍数

Ex1. 对于任意一个数num，试定义一个函数move\_num(xxx)，其返回值是该数循环左移得到的所有数构成的列表（假设数字中不含0）

```
def move_num(num):
    nnum = num; result_lst = []
    count = 0
    while nnum != 0:
        nnum //= 10
        count += 1
    index = 1
    while index < count:
        num_1 = num % (10 ** (count - 1))
        num_2 = num // (10 ** (count - 1))
        num = num_1 * 10 + num_2
        result_lst.append(num)
        index += 1
    return result_lst
```

确定数字的位数

循环左移一位

```
def move_num(num):
    str_num = str(num)
    length = len(str_num)
    index = 1; result_lst = []
    while index < length:
        str_num = str_num[1:] + str_num[0]
        result_lst.append(int(str_num))
        index += 1
    return result_lst
```

切片求解，即使  
数字中含有0也  
可以使用

## 2. 验证命题：如果一个三位整数是37的倍数，则这个整数循环左移后得到的另两个数也是37的倍数

Ex2. 验证关于黑洞数的命题

```
def operate_num(num):  
    max_num = int("".join(sorted(str(num), reverse=True)))  
    min_num = int("".join(sorted(str(num))))  
    return max_num - min_num
```

```
def is_same(num):  
    if len(set(str(num))) != 3:  
        return True  
    return False
```

如果有相同的数字则  
集合长度一定小于3

自己写较大程序时，尽可能把程序分解成多个过程，这样方便Debug

```
In[1]: list(sorted(["1","2","12","123"]))
```

```
Out[1]: ['1', '12', '123', '2']
```

单个数字字符串可以直接排序，但是  
多个数字字符串不可直接这么排序

```
def main(num):  
    count = 0; lst = [str(num)]  
    while count < 7:  
        num = operate_num(num)  
        lst.append(str(num))  
        if num == 495:  
            break  
        count += 1  
    if count == 7:  
        return -1  
    return lst
```

最多操作七次

返回值为-1代表不成功

## 2. 验证命题：如果一个三位整数是37的倍数，则这个整数循环左移后得到的另两个数也是37的倍数

Ex2. 验证关于黑洞数的命题

```
if __name__ == "__main__":  
    for i in range(100,1000):  
        if not is_same(i):  
            res_lst = main(i)  
            if res_lst == -1:  
                print("{} is False".format(i))  
                continue  
            print("-".join(res_lst))
```

如果是验证该命题，只需要加入flag即可

```
if __name__ == "__main__":  
    flag = True  
    for i in range(100,1000):  
        if not is_same(i):  
            res_lst = main(i)  
            if res_lst == -1:  
                flag = False  
                break  
    if flag:  
        print("It's a true proposition.")  
    else:  
        print("It's a false proposition.")
```

↑  
输出所有结果

验证命题是否成立 →

3. 咖啡名提取：编写一个函数 `clean_list()` 处理此咖啡列表，处理后列表中只含咖啡名称，并将此列表返回

---

**思路：**

对于某个“脏”字符串，先构建空字符串用来储存结果。

循环字符串中的每个字符，判断其是否是字母（参考第一题），将字母加到用来储存的字符串中



3. 咖啡名提取：编写一个函数 `clean_list()` 处理此咖啡列表，处理后列表中只含咖啡名称，并将此列表返回

答案：

```
def clean_list(lst):
    cleaned_list = []
    for item in lst:
        clean_item = ""
        for c in item:
            if c.isalpha():
                clean_item += c
        cleaned_list.append(clean_item)
    return cleaned_list

if __name__ == "__main__":
    coffee_list = eval(input())
    cleaned_list = clean_list(coffee_list)
    coffee_zip = list(zip(range(1, len(cleaned_list)+1), cleaned_list))
    i = int(input())
    print(coffee_zip[i-1][1])
```

Ex1. 如何将“脏”字符串的咖啡名和价格同时提取出来？

4. 哥德巴赫猜想：2000以内（包含2000）的正偶数（大于等于4）都能够分解为两个质数之和

---

**思路：**

对于一个偶数 $n$ ，从2开始循环到 $n$ ，如果 $m$ 是素数判断 $n-m$ 是否为素数，如果不是继续循环，如果循环到 $n$ 还没有找到 $m, n-m$ 对，则说明命题不成立

4. 哥德巴赫猜想：2000以内（包含2000）的正偶数（大于等于4）都能够分解为两个质数之和

答案：法1

```
from math import sqrt
```

```
def isprime(n):  
    if n == 2:  
        return True  
    else:  
        for i in range(2, int(sqrt(n)) + 1):  
            if n % i == 0:  
                return False  
        return True
```

for循环会大大降低运算速度，此代码有四个循环，如果后续使用Python写代码时尽量减少for循环的数目！  
① 改变算法； ② 如果可以划归成矩阵运算用numpy

```
if __name__ == "__main__":  
    for even in range(4, 2001, 2):  
        flag = 0  
        for factor in range(2, even):  
            if isprime(factor) and isprime(even-factor):  
                flag = 1  
                print("{}={}+{}".format(even, factor, even - factor))  
                break  
        if flag == 0:  
            print(even)  
            break
```

4. 哥德巴赫猜想：2000以内（包含2000）的正偶数（大于等于4）都能够分解为两个质数之和

答案：法2

```
def generate_prime_lst(num):  
    prime_lst = []  
    for i in range(2,num + 1):  
        if isprime(i):  
            prime_lst.append(i)  
    return prime_lst
```

由于素数都是正数，因此2000以内的偶数拆分成的两个质数都属于2000以内的质数，可以先生成2000以内质数列表，再判断是否在列表内即可减少循环次数

```
if __name__ == "__main__":  
    prime_lst = generate_prime_lst(2000)  
    for even in range(4,2001,2):  
        flag = 0  
        for factor in prime_lst:  
            if (even - factor) in prime_lst:  
                flag = 1  
                print("{}={}+{}".format(even,factor,even - factor))  
                break  
        if flag == 0:  
            print(even)  
            break
```

#### 4. 哥德巴赫猜想：2000以内（包含2000）的正偶数（大于等于4）都能够分解为两个质数之和

Ex1. 求1000以内的完全数（即除自身之外其所有引子的和时其本身）

```
def factor_sum(num):  
    sum_ = 0  
    for i in range(1,num):  
        if num % i == 0:  
            sum_ += i  
    return sum_
```

根据需求  
灵活选择  
返回值

```
if __name__ == "__main__":  
    for i in range(1,1000):  
        if i == factor_sum(i):  
            print(i)
```

```
def factor_sum(num):  
    factor_lst = []  
    for i in range(1,num):  
        if num % i == 0:  
            factor_lst.append(i)  
    return factor_lst
```

```
if __name__ == "__main__":  
    for i in range(1,1000):  
        factor_lst = factor_sum(i)  
        if i == sum(factor_lst):  
            plus_str = "+".join([str(x) for x in factor_lst])  
            print("{}={}".format(i,plus_str))
```

将结果输出成“6=1+2+3”的形式

5. 特殊数问题：对于两个不同的整数A和B，如果整数A的全部因子（不包括A本身）之和等于B；且整数B的全部因子（不包括B本身）之和等于A，则将A和B称为特殊数

---

**思路：**

参考第4题Ex1，计算所有A的因子（除了A本身）的

和B，再计算B所有因子的和C，看C是否与A相等。

可以考虑使用集合储存结果（要保证先小后大），这样可以保证不出现重

复数对



5. 特殊数问题：对于两个不同的整数A和B，如果整数A的全部因子（不包括A本身）之和等于B；且整数B的全部因子（不包括B本身）之和等于A，则将A和B称为特殊数

答案：

```
from math import sqrt

def fac(n):
    factor_lst = [1]
    for i in range(2, int(sqrt(n)) + 1):
        if n % i == 0:
            if i == sqrt(n):
                factor_lst.append(i)
            else:
                factor_lst.extend([i, int(n / i)])
    return sum(factor_lst)
```

这里只是为了减少运算次数，也可以直接用

```
for i in range(1, n):
    if n % i == 0:
        factor_lst.append(i)
```

```
if __name__ == "__main__":
    n = int(input())
    result_lst = []
    finished_lst = []
    for i in range(1, n + 1):
        if i in finished_lst:
            continue
        m = fac(i)
        if m == i:
            continue
        if fac(m) == i:
            result_lst.append(sorted([i, m]))
            finished_lst.extend([i, m])
    for a, b in result_lst:
        print("{}-{}".format(a, b))
```

为了减少运算次数，如果不在意运算次数可以忽略

A、B两整数是不同的

5. 特殊数问题：对于两个不同的整数A和B，如果整数A的全部因子（不包括A本身）之和等于B；且整数B的全部因子（不包括B本身）之和等于A，则将A和B称为特殊数

答案：

```
if __name__ == "__main__":
```

```
    n = int(input())
```

```
    result_set = set()
```

```
    for i in range(1, n + 1):
```

```
        m = fac(i)
```

```
        if m == i:
```

```
            continue
```

```
        if fac(m) == i:
```

```
            num_pair = sorted([i, m])
```

```
            result_set.add("{}-{}".format(num_pair[0], num_pair[1]))
```

```
    for res in result_set:
```

```
        print(res)
```

使用集合储存结果，如果用列表的话可以加一个if num\_pair in result\_lst的语句

6. 全数字问题：如果一个 $n$  ( $n \leq 9$ ) 位数刚好包含了1至 $n$ 中所有数字各一次则称它们是全数字 (pandigital) 的

---

**思路：**

对于一个 $n$  ( $n \leq 9$ ) 位数，将其转化为字符串，判断

1- $n$ 数字是否在字符串中即可。

6. 全数字问题：如果一个 $n$  ( $n \leq 9$ ) 位数刚好包含了1至 $n$ 中所有数字各一次则称它们是全数字 (pandigital) 的

答案：

```
def pandigital(nums):
    result_lst = []
    for num in nums:
        num = str(num)
        flag = True
        for i in range(1, len(num) + 1):
            if str(i) not in num:
                flag = False
                break
        if flag:
            result_lst.append(num)
    return result_lst
```

方法不唯一!

```
if __name__ == "__main__":
    lst = pandigital(eval(input()))
    if not lst:
        print("not found")
    else:
        for num in lst:
            print(num)
```