

作业

1、实现Base64解码

思路：

首先判断bytes的长度是否是4的倍数，如果是，继续。

a编码后是YQ==，先查Y找到24，这个数要放到最高6位，Q为16放次高6位，=找不到就是0凑6位，最终凑成一个24位数，最后追加到一个bytearray中去。

```
# www.magedu.com
# base64解码实现

alphabet = b"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/"

def base64decode(src:bytes):
    ret = bytearray()
    length = len(src)

    step = 4 # 对齐的，每次取4个
    for offset in range(0, length, step):
        tmp = 0x00
        block = src[offset:offset + step]

        # 反查表，从字符到index a YQ==
        for i,c in enumerate(reversed(block)):
            index = alphabet.find(c)
            if index == -1:
                continue # 找不到说明是=，就是0，不用移位相加了
            tmp += index << i*6

        ret.extend(tmp.to_bytes(3, 'big'))
    return bytes(ret.rstrip(b'\x00')) # 把最右边的\x00去掉

# base64的decode
txt = "TWFu"
txt = "TWE="
txt = "TQ=="
txt = "TWFuTWE="
txt = "TWFuTQ=="
txt = txt.encode()
print(txt)

print(base64decode(txt).decode())

# base64实现
import base64
```

```
print(base64.b64decode(txt).decode())
```

改进

1. reversed可以不需要
2. alphabet.find效率低

```
# www.magedu.com
# base64解码实现
from collections import OrderedDict

base_tbl = b"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/"
# 用字典查询提升效率,有序字典只是为了顺便记录顺序
alphabet = OrderedDict(zip(base_tbl, range(64)))

def base64decode(src: bytes):
    ret = bytearray()
    length = len(src)

    step = 4 # 对齐的,每次取4个
    for offset in range(0, length, step):
        tmp = 0x00
        block = src[offset:offset + step]

        # 反查表,从字符到index
        for i in range(4):
            index = alphabet.get(block[-i-1])
            if index is not None: # 注意不能是0
                tmp += index << i*6
            # 找不到,不用移位相加了
        ret.extend(tmp.to_bytes(3, 'big'))
    return bytes(ret.rstrip(b'\x00')) # 把最右边的\x00去掉,不可变

# base64的decode
txt = "TWFu"
# txt = "TWE="
# txt = "TQ=="
# txt = "TWFuTWE="
# txt = "TWFuTQ=="
txt = txt.encode()
print(txt)

print(base64decode(txt).decode())

# base64实现
import base64
print(base64.b64decode(txt).decode())
```

2、完善命令分发器

完善命令分发器，实现函数可以带任意参数（可变参数除外），解析参数并要求用户输入
即解决下面的问题

```
# 自定义函数
@reg('mag')
def foo1(x,y):
    print('magedu', x, y)

@reg('py')
def foo2(a,b=100):
    print('python', a, b)
```

思路：

可以有2种方式

1、注册的时候，固定死，@reg('py',200,100)

可以认为@reg('py',200,100)和@reg('py',300,100)是不同的函数，可以用partial函数。

2、运行时，在输入cmd的时候，逗号或空格分割，获取参数。

至于函数的验证，以后实现。

一般用户都喜欢使用单纯一个命令如mag，然后直接显示想要的结果，所以采用第一种方式

```
from functools import partial

# 自定义函数可以有任意参数，可变参数、keyword-only除外
def command_dispatcher():
    # 构建全局字典
    cmd_tbl = {}

    # 注册函数
    def reg(cmd,*args,**kwargs):
        def _reg(fn):
            func = partial(fn,*args,**kwargs)
            cmd_tbl[cmd] = func
            return func
        return _reg

    # 缺省函数
    def default_func():
        print('Unknown command')

    # 调度器
    def dispatcher():
        while True:
            cmd = input('Please input cmd>>')
            # 退出条件
            if cmd.strip() == '':
                return
            cmd_tbl.get(cmd, default_func)()
```

```
    return reg, dispatcher

reg, dispatcher = command_dispatcher()

# 自定义函数
@reg('mag', z=200, y=300, x=100)
@reg('mag1', z=300, y=300, x=300)
def foo1(x, y, z):
    print('magedu', x, y, z)

@reg('py', 300, b=400)
def foo2(a, b=100):
    print('python', a, b)

# 调度循环
dispatcher()
```

