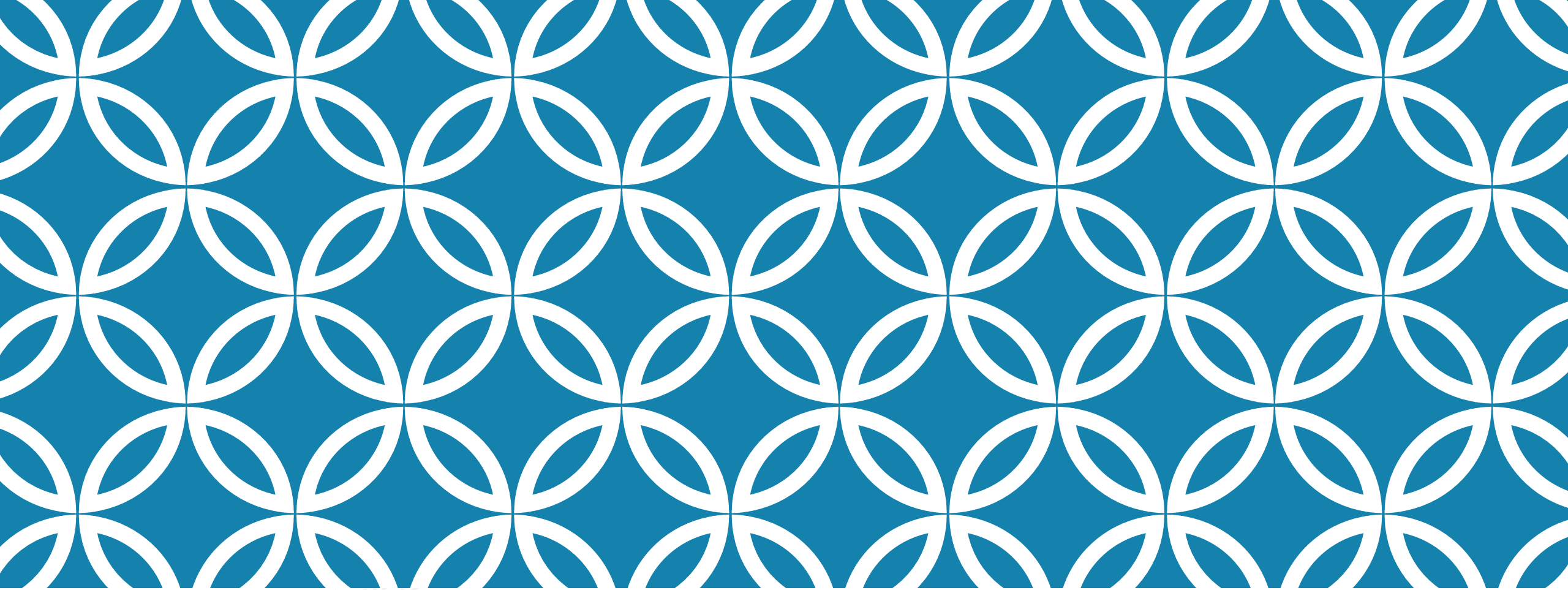


# PYTHON文件、面向对象和异常等若干问题探讨

用Python玩转数据

Dazhuang@NJU 2020春

# 文件

如何向文件头部插入一个字符串或新行?

```
with open('a.txt', 'r+') as f:
        lines = f.readlines()
        for i in range(0, len(lines)):
                lines[i] = str(i+1) + ' ' + lines[i]
        lines.insert(0, 'Information\n')
        f.seek(0)
        f.writelines(lines)
```

# 面向对象

```
class A(object):
    def __init__(self, t):
        self.m, self.s = [int(x) for x in t.split(':')]
    def p(self):
        print(self.m, ":", self.s, sep = '')
class B(A):
    def p(self):
        if self.s // 60 !=0:
            self.m += self.s // 60
            self.s %= 60
        A.p(self)

if __name__ == "__main__":
    x = B("3:75")
    x.p()
```

阅读程序

# 面向对象

调用父类方法super()

```
class p_father(object):
    …
    def pnt(self, a, b):
        self.a = a
        self.b = b
        print(self.a+self.b)

class p_child(p_father):
    def pnt(self, a, b):
        # p_father.pnt(self, a, b)
        super().pnt(a, b)
        print(self.a*self.b)

calc = p_child()
calc.pnt(3, 5)
```

# 面向对象

Python是否支持传统函数重载(overloading)方法？

```
class printf:
    def pnt(self, a, b):
        self.a = a
        self.b = b
        print(self.a+self.b)

    def pnt(self, a):
        self.a = a
        print(self.a)

e = printf()
e.pnt(3, 5)
e.pnt(8.5)
```

# 异常

**筛选数字**

编写程序，输入字符串（可能分成若干段），将字符串中所有的数字字符串(除去尾部标点符号外可构成一个整数或浮点数，例如100times不是，句子中的56.78.23不是，句尾的45.78.是，或是一个单独的数字)转换成浮点数并输出。如果没有数字字符串，则输出‘Not Found!'。字符串中可能包括以下标点符号：","、"."、“"”、“?”和"!"，标点符号不会连续出现。

```
def isfloat(s):
    if s[-1] == '.':
        s = s.strip('.')
    try:
        float(s)
    except ValueError:
        return False
    return float(s)

if __name__ == "__main__":
    text = input()
    for ch in text:
        if ch in ',?"!':
            text = text.replace(ch, '')
    words = text.split()
    found = 0
    for word in words:
        if isfloat(word) != False:
            print(isfloat(word))
            found = 1
    if found == 0:
        print('Not Found!')
```

# 异常 - RAISE语句

- for语句的实现

```
lst = [1,2,3,4]
it = iter(lst)
while True:
    try:
        x = next(it)
        print(x) # 或其他操作
    except StopIteration:
        break
```

- sqrt()函数的实现

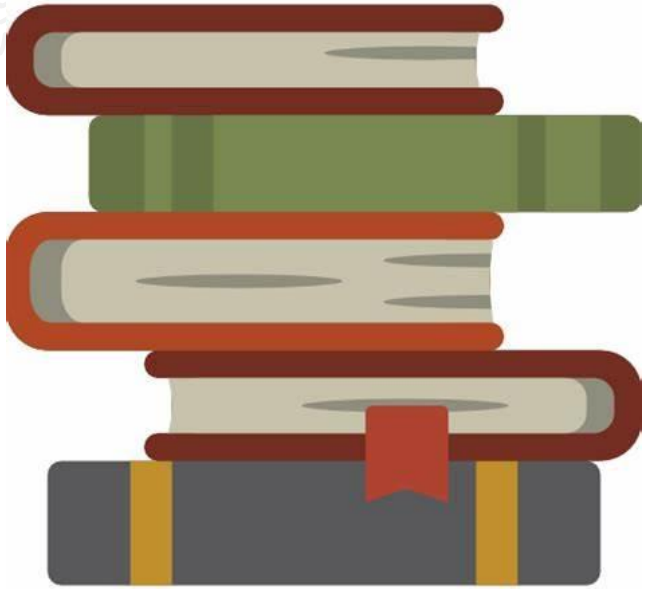
```
def sqrt(x):
    if not isinstance(x, (int, float)):
        raise TypeError('must be real number, not str')
    elif x < 0:
        raise ValueError('math domain error')
```

计算平方根的功能模块

# 面向对象和异常示例—栈的实现

用list类实现一个栈（stack）

| 栈方法 | 列表实现 |
| --- | --- |
| S.push(e) | L.append(e) |
| S.pop() | L.pop() |
| S.top() | L[-1] |
| S.is_empty() | len(L)==0 |
| len(S) | len(L) |



From《数据结构与算法 Python语言实现》

```
class Empty(Exception):
    pass

class Stack:
    def __init__(self):
        self.data = []
    def length(self):
        return len(self.data)
    def is_empty(self):
        return len(self.data) == 0
    def push(self, e):
        self.data.append(e)
    def top(self):
        if self.is_empty():
            raise Empty('Stack is empty!')
        return self.data[-1]
    def pop(self):
        if self.is_empty():
            raise Empty('Stack is empty!')
        return self.data.pop()
```

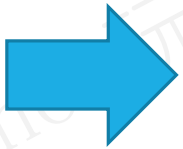
```
S = Stack()
S.push(3)
S.push(5)
S.push(8)
print(S.length())
print(S.top())
print(S.length())
print(S.pop())
print(S.length())
```

```
def f1():
    print('f1 was called')

def f2():
    print('f2 was called')

...
```

增加计时功能



```
import time
def f1():
    start_time = time.process_time()
    print('f1 was called')
    end_time = time.process_time()
    print('time=', end_time-start_time)

def f2():
    start_time = time.process_time()
    print('f2 was called')
    end_time = time.process_time()
    print('time=', end_time-start_time)
```

# PYTHON装饰器

功能：拓展原函数功能的一种函数，这个函数的返回值也是一个函数

优势：不更改原函数代码给函数增加新的功能

应用：缓存、日志和权限校验等

```
import time

def deco(fun):
    def wrapper():
        start_time = time.process_time()
        fun()
        end_time = time.process_time()
        print('time=', end_time-start_time)

    return wrapper

@deco
def f1():
    print('f1 was called')

@deco
def f2():
    print('f2 was called')

f1()
f2()
```

```python
from functools import wraps

def logit(func):
    @wraps(func)
    def with_logging(*args, **kwargs):
        print(func.__name__ + " was called")
        return func(*args, **kwargs)
    return with_logging

@logit
def addition_func(x):
   """Do some math."""
   return x + x


result = addition_func(4)
# Output: addition_func was called
```

- “Logging is another area where the decorators shine”
- https://book.pythontips.com/en/latest/decorators.html

- https://realpython.com/primer-on-python-decorators/
- https://python101.pythonlibrary.org/chapter25_decorators.html