TAIJI LABORATORY
FOR GRAVITATIONAL WAVE UNIVERSE

ICTP-AP
International Centre
for Theoretical Physics Asia-Pacific
国际理论物理中心-亚太地区

University of Chinese Academy of Sciences

引力波数据探索：编程与分析实战训练营

第 2 部分 基于 Python 的数据分析基础

数据科学语言 Python 入门到熟悉

主讲老师：王赫

ICTP-AP, UCAS

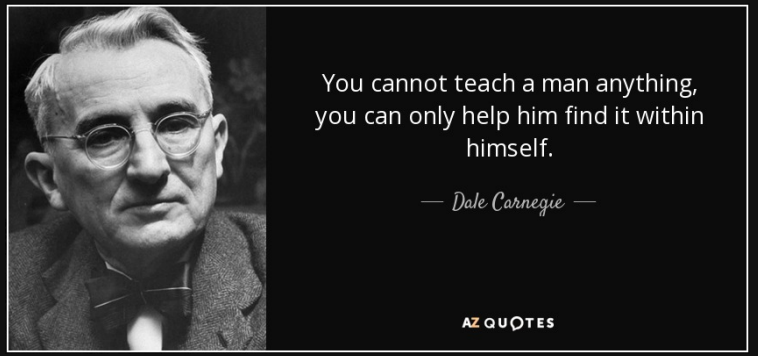
2023/11/29

Taiji Program in Space
Gravitational wave physics
空间太极计划

关于上一讲的学员反馈

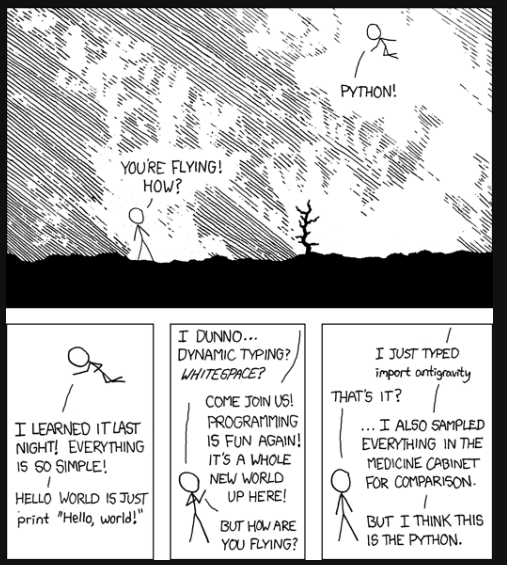
- (上次选做作业的问题)windows在搭载gpu的容器构建中碰到问题是nvidia的相关工具是仅在linux上搭载的，为此必须装wsl2，装完，整体搭载完之后碰到问题是是否需要在容器内再安装docker，（因为nvidia教程中包含docker命令）但是这有种双层嵌套的感觉，像是在wsl2中的container中的container
- 能否别一节课时间太长，因为课业压力确实很重，只能分段学习……

- 授课主要面向的是国科大以引力波数据处理为主要研究方向的研究生和高年级本科生。
- 不管“包学包会”，但会尽可能弥补和完善听课体验和授课内容。



数据科学语言 Python 入门到熟悉

- 初识 Python
- Python 基本数据类型：变量、运算、表达式
  - 数值精度
  - 可变对象 vs 不可变对象
  - List comprehensions
- Python 流程控制与异常处理
- Python 面相对象编程
- Python 中内存回收机制
- Python 编程语言的进阶途径



“ Talk is cheap. Show me the code. ”

Linus Torvalds

Python 编程语言

- Python语言是一种解释型、面向对象、动态数据类型的高级程序设计语言
- Python语言是数据分析师的首选数据分析语言

游戏开发

机器学习

数据分析

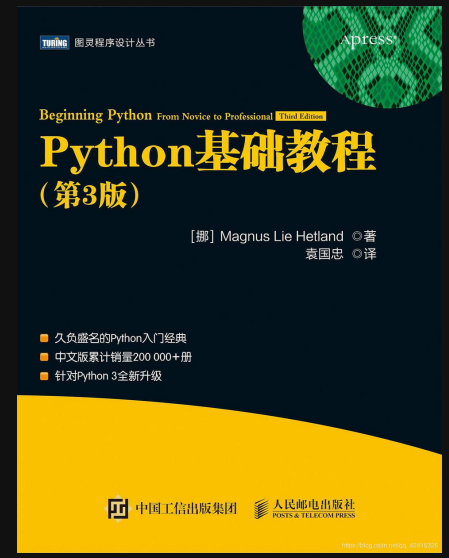
动 画 电 影 效 果

创建复杂的 Web 应用程序

网站开发

...

- 个人感觉很不错的入门材料：
  - Python 3.x 官方中文文档：https://docs.python.org/zh-cn/3/index.html
  - 《Python 基础教程》(第3版) —— Magnus Lie Hetland
  - Real Python: https://realpython.com
  - 廖雪峰的《小白的 Python 新手教程》：https://www.liaoxuefeng.com/wiki/1016959663602400
  - ...



Python 的基本数据类型/结构

数字类型

- **Int** 整型
- **Float** 浮点型
- **String** 字符串
- *Bool* *布尔型*
- **List** 列表
- Tuple 元组
- Set 集合
- **Dict** 字典
- **...**
-

and more complex, range, bytes, ... (内置类型)

- Int 和 Float 啥区别？小数点的区别！
- 用作**计算器**用的 +、-、*、/、//、%、**、()、...

Python 的基本数据类型/结构

数字类型

- **Int** 整型
- **Float** 浮点型

文本序列类型

- **String** 字符串
- *Bool* *布尔型*
- **List** 列表
- Tuple 元组
- Set 集合
- **Dict** 字典
- **...**
-

and more complex, range, bytes, ... (内置类型)

- Int 和 Float 啥区别？小数点的区别！
- 用作**计算器**用的 +、-、*、/、//、%、**、()、...
- ' ' 和 " " 有啥区别？没区别！
- String：用 **+** 来拼接，用 * 来重复
- 关于“**转义**”你起码该晓得的：\n，\t，\r 与 r'...'
- 理解序列之**切片**
- 关于 String 的常见的**方法**：in, not in、format、find、strip、lstrip、rstrip、split、upper、lower

Python 的基本数据类型/结构

数字类型

- **Int** 整型
- **Float** 浮点型

文本序列类型

- **String** 字符串
- *Bool* *布尔型*

序列类型

- **List** 列表
- Tuple 元组
- Set 集合
- **Dict** 字典

- ...
-

and more complex, range, bytes, ... (内置类型)

- 序列类型：可任何类型元素的序列结构
- 序列类型：用 + 来拼接，用 * 来重复
- 关于 List 的常见的**方法**
- 关于 List 直接赋值、浅拷贝和深度拷贝的内涵
- List 和 Tuple 有啥区别？括号或逗号的区别！
- Set 是由不重复元素组成的无序的集：爱做逻辑运算

Python 的基本数据类型/结构

数字类型

- **Int** 整型
- **Float** 浮点型

文本序列类型

- **String** 字符串
- *Bool* *布尔型*

序列类型

- **List** 列表
- Tuple 元组
- Set 集合

映射类型

- **Dict** 字典

- ...
-

and more complex, range, bytes, ... (内置类型)

- 序列类型：可任何类型元素的序列结构
- 序列类型：用 + 来拼接，用 * 来重复
- 关于 List 的常见的**方法**
- 关于 List 直接赋值、浅拷贝和深度拷贝的内涵
- List 和 Tuple 有啥区别？括号或逗号的区别！
- Set 是由不重复元素组成的无序的集：爱做逻辑运算
- Dict 乃“key-value”的数据结构，所谓“键值对”、“联合数组”
- Dict 的属性和方法

Python 的基本数据类型/结构

数字类型

- **Int** 整型
- **Float** 浮点型

文本序列类型

- **String** 字符串
- *Bool* *布尔型*

序列类型

- **List** 列表
- Tuple 元组
- Set 集合

映射类型

- **Dict** 字典

- ...
-

and more complex, range, bytes, ... (内置类型)

- 神乎其神的 List Comprehensions！ （”**列表推导式**“）
  - 序列类型 → 序列/集合/映射类型
- One-lined Python: Powerful Python One-Liners
  - List Comprehensions
  - lambda (map、filter、reduce)

小结：Python 中所有基本数据类型的特点总结与归纳、变量的内存机制、熟练表达式的用法提高编程效率

Python 的流程控制与异常处理

```python
# if
if condition1:
    statement1
elif condition2:
    statement2
else:
    statment3
```

```python
# for
for i in range(5):
    my_func(i)

# while
i = 0
while i < 5:
    my_func(i)
    i += 1
```

```python
# try ... except ...
try:
    my_func()
except Exception as e:
    print(e)
    raise
```

- break vs continue (pass)

```python
for num in range(2, 10):
    if num % 2 == 0:
        print("Found an even number", num)
        continue
    print("Found a number", num)
'''
Found an even number 2
Found a number 3
Found an even number 4
Found a number 5
Found an even number 6
Found a number 7
Found an even number 8
Found a number 9
'''
```

```python
for n in range(2, 10):
    for x in range(2, n):
        if n % x == 0:
            print(n, 'equals', x, '*', n//x)
            break
    else:
        # loop fell through without finding a factor
        print(n, 'is a prime number')

'''
2 is a prime number
3 is a prime number
4 equals 2 * 2
5 is a prime number
6 equals 2 * 3
7 is a prime number
8 equals 2 * 4
9 equals 3 * 3
'''
```

Python 的流程控制与异常处理

```python
# if
if condition1:
    statement1
elif condition2:
    statement2
else:
    statment3
```

```python
# for
for i in range(5):
    my_func(i)

# while
i = 0
while i < 5:
    my_func(i)
    i += 1
```

```python
# try ... except ...
try:
    my_func()
except Exception as e:
    print(e)
    raise
```

- break vs continue (pass)
- for ... (else) ...

```python
for i in range(3):
    print(i)


else:
    print("end")
'''
0
1
2
end
'''
```

```python
for i in range(3):
    print(i)
    if i % 2 == 0:
        break
else:
    print("end")
'''
0
'''
```

- 只有当循环里没有遇到 break 时，else 块才会在循环结束后执行
- break 和 else 是两个互斥的条件

```python
# Source: https://stackoverflow.com/a/9980752/1392860
for i in mylist:
    if i == target:
        break
    process(i)
else:
    raise ValueError("List argument missing terminal flag.")
# 所以无需专门建立一个临时标记变量来标记是否已经找到了 target
```

Python 的流程控制与异常处理

```python
# if
if condition1:
    statement1
elif condition2:
    statement2
else:
    statment3
```

```python
# for
for i in range(5):
    my_func(i)

# while
i = 0
while i < 5:
    my_func(i)
    i += 1
```

```python
# try ... except ...
try:
    my_func()
except Exception as e:
    print(e)
    raise
```

- break vs continue (pass)
- for ... (else) ...
- while/for ... try ... except ...

```python
while True:
    try:
        x = int(input("Please enter a number: "))
        break
        # Don't forget stop the Loop!
    except ValueError:
        print("Oops!  That was no valid number.  Try again...")
```

Python 的流程控制与异常处理

```python
# if
if condition1:
    statement1
elif condition2:
    statement2
else:
    statment3
```

```python
# for
for i in range(5):
    my_func(i)

# while
i = 0
while i < 5:
    my_func(i)
    i += 1
```

```python
# try ... except ...
try:
    my_func()
except Exception as e:
    print(e)
    raise
```

- break vs continue (pass)
- for ... (else) ...
- while/for ... try ... except ...
- try ... except ... (else) ... (finally) ...

```python
def divide(x, y):
    try:
        result = x / y
    except ZeroDivisionError:
        print("division by zero!")
    else:
        print("result is", result)
    finally:
        print("executing finally clause")
```

```
>>> divide(2, 1)
'''
result is 2.0
executing finally clause
'''
>>> divide(2, 0)
'''
division by zero!
executing finally clause
'''
>>> divide("2", "1")
'''
executing finally clause
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 3, in divide
TypeError: unsupported operand type(s) for /: 'str' and 'str'
'''
```

Python 的流程控制与异常处理

```python
# if
if condition1:
    statement1
elif condition2:
    statement2
else:
    statment3
```

```python
# for
for i in range(5):
    my_func(i)

# while
i = 0
while i < 5:
    my_func(i)
    i += 1
```

- break vs continue (pass)
- for ... (else) ...
- while/for ... try ... except ...
- try ... except ... (else) ... (finally) ...

```python
from random import sample as love_from_boss

while luck not in my_life:
    if holiday and love_from_boss([0]*364+[1], 1)[0]:
        print('Happiness!')
        My_life_my_choice(max_power=True, sleep_24h=True)

    wakeup()
    breakfasts()
    working()

    lunch()
    try:
        sleep(1)
        lifespan += 1
        bonus -= 100
    except:
        lifespan -= 1
    working()

    try:
        bodyshaping()
        bodyweight -= 20
    except:
        bodyweight += 20
    dinner()

    while not luck:
        try:
            submit(working.output, stress=True)
        except Exception as e:
            print('Boss: 这是什么鬼? ', e)
            print('Boss: 滚去加班! ')
            overworking()
        else:
            sleeping(until = 7)
            break
        finally:
            lifespan -= 10
        if lifespan <= 0:
            dead()
            print('Game over')
            break
```

Python 面向对象编程

- 简易入门教程：（Source: ID 王大伟）
  - Python面向对象编程从零开始（1）——从没对象到有对象
  - Python面向对象编程从零开始（2）—— 与对象相互了解
  - Python面向对象编程从零开始（3）—— 小姐姐请客上篇
  - Python面向对象编程从零开始（4）—— 小姐姐请客下篇
  - Python面向对象编程从零开始（5）—— 小姐姐要买房
- 官方教程：https://docs.python.org/3.10/tutorial/classes.html
- 面向过程 vs 面向对象
- 类与实例（对象）、属性与方法

```python
std1 = { 'name': 'Michael', 'score': 98 }
std2 = { 'name': 'Bob', 'score': 81 }

# 处理学生信息可以通过函数实现，比如打印学生的成绩：

def print_core(std):
    print('%s : %s'%(std['name'],std['score']))

print_core(std1)
print_core(std2)
```

```python
class Student(object): # 类
    def __init__(self,name,score):
        self.name = name    # 类的属性
        self.score = score    # 类的方法
    def print_score(self):
        print('%s : %s'%(self.name,self.score))

amy = Student('amy',120)  # 类对象的实例化
jack = Student('jack',108)

amy.print_score()
jack.print_score()
```

Python 面向对象编程

- 简易入门教程：（Source: ID 王大伟）
  - Python面向对象编程从零开始（1）——从没对象到有对象
  - Python面向对象编程从零开始（2）—— 与对象
  - Python面向对象编程从零开始（3）—— 小姐姐
  - Python面向对象编程从零开始（4）—— 小姐姐
  - Python面向对象编程从零开始（5）—— 小姐姐
- 官方教程：https://docs.python.org/3.10/tutorial/clas
- 面向过程 vs 面向对象
- 类与实例（对象）、属性与方法

```python
std1 = { 'name': 'Michael', 'score': 98 }
std2 = { 'name': 'Bob', 'score': 81 }

# 处理学生信息可以通过函数实现，比如打印学生的成绩：

def print_core(std):
    print('%s : %s'%(std['name'],std['score'
print_core(std1)
print_core(std2)
```

```python
import random as r

class Fish():    # 父类
    def __init__(self):
        self.x =r.randint(0,10) # 类属性
        self.y =r.randint(0,10)
    def move(self):             # 类方法
        self.x -=1    # 一直向西移动
        print("我的位置是：",self.x, self.y)
# 利用继承演示鱼游动方向位置。
class Goldfish(Fish):    # 子类
    pass
class Salmon(Fish):        # 子类
    pass
class Shark(Fish):
    # 这里重写了__init__方法，就会覆盖掉父类的方法了，
    # 用到super函数后就可以继续使用父类的方法。
    def __init__(self):
        # super函数不用给定任何基类的名字(如下)，它会一层层找出代码所有父类里面对应的方法，
        # 要改变该类的继承关系时只需修改这个类的父类就行就是括号里面的Fish。
        super().__init__()    # super().重写的属性或方法
        self.hungry = True
    def eat(self):
        if self.hungry:
            print("我要吃了。。。")
            self.hungry = False
        else:
            print('好饱了。。。')
```

Python 中的常见标准库（模块）

- os / sys （操作系统接口/与解释器交互的接口）
  - `os.getcwd()/os.listdir()/os.mkdir()/os.remore()/os.system()/...`
  - `os.sys.path/sys.path/sys.argv/sys.stdout/sys.stdin/sys.stderr/...`
- math / random / statistics（数学/随机数/统计）
  - `random.random()/random.sample()/random.seed()/...`
- glob（文件通配符）
- urllib.request / smtplib（互联网访问/邮件发送）
- time / datetime（日期与时间的格式化）
  - ```
    from datetime import datetime, timedelta
    now = datetime.today()
    now.strftime("%m-%d-%y %H:%M:%S. %d %b %Y is a %A on the %d day of %B.")
    ```
- logging（日志）→ Delgan/loguru
- re（字符串的模式匹配 - 正则表达式）
- ...
    ```

Python 编程语言的进阶途径

Python progression path - From apprentice to guru

Asked 9 years, 3 months ago Active 5 years, 9 months ago Viewed 354k times

471 votes

I thought the process of Python mastery went something like:

1. Discover list comprehensions
2. Discover generators
3. Incorporate map, reduce, filter, iter, range, xrange often into your code
4. Discover Decorators
5. Write recursive functions, a lot
6. Discover itertools and functools
7. Read Real World Haskell (read free online)
8. Rewrite all your old Python code with tons of higher order functions, recursion, and whatnot.
9. Annoy your cubicle mates every time they present you with a Python class. Claim it could be "better" implemented as a dictionary plus some functions. Embrace functional programming.
10. Rediscover the Strategy pattern and then all those things from imperative code you tried so hard to forget after Haskell.
11. Find a balance.

share edited Jan 28 '13 at 10:28 community wiki 3 revs, 2 users 96% wheaties

Source: https://stackoverflow.com/a/2576240/8656360

Python 编程语言的进阶途径

Python progression path - From apprentice to guru

Asked 9 years, 3 months ago Active 5 years, 9 months ago Viewed 354k times

471 votes

I thought the process of Python mastery went something like:

1. Discover list comprehensions 列表推导式
2. Discover generators 生成器
3. Incorporate map, reduce, filter, iter, range, xrange often into your code
4. Discover Decorators 装饰器
5. Write recursive functions, a lot 递归函数
6. Discover itertools and functools 高阶函数库
7. Read Real World Haskell (read free online) 中文版 huangz1990/real-world-haskell-cn
8. Rewrite all your old Python code with tons of higher order functions, recursion, and whatnot.
9. Annoy your cubicle mates every time they present you with a Python class. Claim it could be "better" implemented as a dictionary plus some functions. Embrace functional programming.
10. Rediscover the Strategy pattern and then all those things from imperative code you tried so hard to forget after Haskell.
11. Find a balance.

编程思想 与 代码规范

重构

share edited Jan 28 '13 at 10:28 community wiki 3 revs, 2 users 96% wheaties

Source: https://stackoverflow.com/a/2576240/8656360

Python_cheatsheet:

My Simple Tips:

1. Code every day
2. Write it out / making notes
3. Teach
4. Ask "good" questions
5. Pair programing
6. Contribute to open source

2023 开发者生态系统现状
https://www.jetbrains.com/lp/devecosystem-2023/