Python基础

Python简介

1.程序员：

程序设计人员。

2.程序：

一组计算机能识别和执行的指令，是实现某种需求的软件。

3.操作系统

管理和控制计算机软件与硬件资源的程序；

隔离不同的硬件差异，使开发程序简单化。

例如，Windows，Linux，Unix。

4.硬件

主板--计算机的主要电路系统

CPU--主要负责执行程序指令，处理数据。

硬盘--持久化存储数据的记忆设备，容量大，速度慢。

内存--临时存储数据的记忆设备，容量小，速度快。

IO设备--键盘、鼠标、显示器。

Python定义

是一个免费、开源、跨平台、动态、面向对象的编程语言。

Python程序的执行方法

交互式

在命令行输入指令，回车即可得到结果

1.打开终端

2.进入交互式：python3

3.编写代码：print("hello world")

4.离开交互式：exit()

文件式

将指令编写到.py文件，可以重复运行程序。

1.编写文件

2.打开终端

3.进入程序所在目录：cd 目录

4.执行程序：python3 文件名

Linux常用命令

1.pwd：查看当前工作目录的路径

2.ls：查看指定目录的内容或文件信息

3.cd：改变工作目录（进入到某个目录）

练习：

1.在指定目录创建python文件

--目录：/home/teren1/1905/month01

--文件名：exercise01.py

2.在文件中写入：print("你好，世界！")

3.运行python程序

执行过程

计算机只能识别机器码(1010)，不能识别源代码（python）。

1.由源代码转变成机器码的过程分成两类:编译和解释。

2.编译:在程序运行之前,通过编译器将源代码变成机器码，例如：C语言

--优点:运行速度快

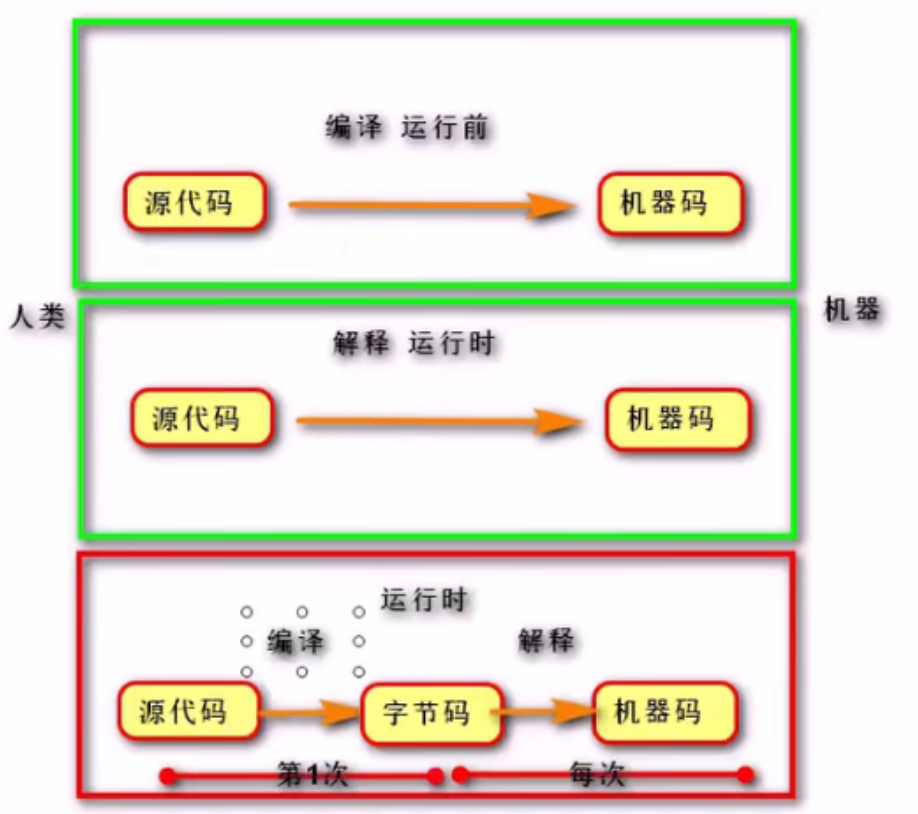
--缺点:开发效率低,不能跨平台。

3.解释:在程序运行之时,通过解释器对程序逐行翻译，然后执行。例如JavaScript

--优点:开发效率高，可以跨平台;

--缺点:运行速度慢

4.python是解释性语言，但为了提高运行速度，使用了一种编译的方法。得到pyc文件，存储了字节码（特定于Python的表现形式，不是机器码



解释器类型

1.CPython(C语言开发)

2.Jython(java开发)

3.IronPython(.net开发)

4.....

数据基本运算

pycharm常用快捷键

1.移动到本行开头：home键

2.移动到本行末尾：end键

3.注释代码：ctrl+/

4.复制行：ctrl+d

5.选择列：鼠标左键+alt

6.移动行：shift+alt+上下箭头

7.智能提示：ctrl+space

注释

给人看的，通常是对代码的描述信息。

1.单选注释：以#号开关。

2.多行注释：三引号开关，三引号结尾。

函数

表示一个功能，函数定义者是提供功能的人，函数调用者是使用功能的人。

例如：

> 1.print(数据) 作用：将括号中的内容显示在控制台中。
> 2.变量 = input("") 作用：将用户输入的内容赋值给变量。

变量

1.定义：关联一个对象的标识符

2.命名：必须是字母或下划线开头，后跟字母、数字、下划线。

> 不能使用关键字，否则发生语法错误：SyntaxError

3.建议命名：字母小写，多个单词以下划线隔开。

> class_name

4.赋值：创建一个变量或改变一个变量关联的数据。

5.语法

```
变量名 = 数据
变量名1 = 变量名2 = 数据
变量名1, 变量名2 = 数据1, 数据2
```

del语句

1.语法：

> del 变量名1, 变量名2

2.作用：

用于删除变量，同时解除与对象的关联。如果可能则释放对象。

3.自动化内存管理的引用计数：

每个对象记录被变量绑定（引用）的数量，当为0时被销毁。

核心数据类型

1.在python中变量没有类型，但关联的对象有类型

2.通过type函数可查看。

空值对象 None

1.表示不存在的特殊对象。

2.作用：占位和解除与对象的关联。

整形 int

1.表示整数，包含正数、负数、0。

2.字面值：

> 十进制：5
> 二进制：0b开头，后跟0或者1
> 八进制：0o开头，后跟0~7
> 十六进制：0x开头，后跟$0_{9,A}$F,A~F

3.小整数对象池：CPython中整数-5至256启动存在小整数对象，并可重复使用。

浮点型 float

1.表示小数，包含正数、负数、0.0

2.字面值：

> 小数：1.0 2.5
> 科学计数法：e/E(正负号)指数
> 1.23e-2（等同于0.0123）
> 1.23456e5（等同于123456.0）

字符串 str

是用来记录文本信息（文字信息）。

字面值：双引号

复数 complex

由实部和虚部组成的数字。

虚部是以j或者J结尾。

字面值：1j 1+1j 1-1j

布尔 bool

用来表示直和假的类型

True表示真（条件满足或成立），本质是1

False表示假（条件不满足或不成立），本质是0

数据类型转换

1.转换为整形：int(数据)

2.转换为浮点型：float(数据)

3.转换为字符串：str(数据)

4.转换为布尔型：bool(数据)

> 结果为False: bool(0) bool(0.0) bool(None)

5.混合类型自动升级

1+2.14 返回的结果是3.14
1+3.0 返回的结果是4.0

运算符

算数运算符

+加法
-减法
*乘法
/除法：结果为浮点数
//地板除：除的结果去掉小数部分
%求余
**幂运算
优先级从高到低：()

增强运算符

y += x 等同于 y = y + x
y -= x 等同于 y = y - x
y *= x 等同于 y = y * x
y /= x 等同于 y = y / x

y //= x 等同于 y = y // x
y **= x 等同于 y = y ** x

比较运算符

>大于
>=小于等于
<小于
<=小于等于
!=不等于
==等于

逻辑运算符

与and

表示并且的关系，一假俱假。

示例：

```
True and True  # True
True and False  # False
False and True  # False
False and False  # False
```

或or

表示或者的关系，一真俱真。

示例：

```
True or True  # True
True or False  # True
False or True  # True
False or False  # False
```

非not

表示相反的逻辑。

示例：

```
not True  # False
not False  # True
```

身份运算符

身份运算符是python用来判断的两个对象的存储单元是否相同的一种运算符号

身份运算符只有is和is not两个运算符，返回的结果都是TRUE或者FALSE。

用“”进行判断是，只要最终的值是一致的用“”进行比较运算结果就是TRUE，而is则必须引用同一对象返回结果才为TRUE，否则就是FALSE。

优先级

高到低：

- 算数运算符
- 比较运算符
- 快捷运算符
- 身份运算符
- 逻辑运算符

语句

行

1.物理行：程序员编写代码的行。

2.逻辑行：python解释器需要执行的指令。

3.建议一个逻辑行在一个物理行上。

4.如果一个物理行中使用多个逻辑行，需要使用分号;隔开。

5.如果逻辑行过长，可以使用隐式换行或显式换行。

- 隐式换行：所有括号的内容换行，成为隐式换行

> 括号包括：()[]{}三种

- 显式换行：通过折行符\（反斜杠）换行，必须放在一行的末尾，目的是告诉解释器，下一行也是本行的语句

pass语句

通常用来填充语法空白。

选择语句

if elif else语句

1.作用：

让程序根据条件选择性的执行语句。

2.语法：

```
if 条件1:
    语句块1
elif 条件2:
    语句块2
else:
    语句块3
```

3.说明：

elif子句可以有0个或多个。
else子句可以由0个或1个，且只能放在if语句的最后。

if语句的真值表达式

```
if 100:
    print("True")
```

等同于

```
if bool(100):
    print("True")
```

条件表达式

语法：

```
变量 = 结果1 if 条件 else 结果2
```

作用：根据条件（True/False）来返回结果1还是结果2。

循环语句

while语句

1.作用：

可以让一段代码满足条件，重复执行。

2.语法：

```
while 条件:
    满足条件执行的语句
else:
    不满足条件执行的语句
```

3.说明：

else子句可以省略。
在循环体内用break终止循环时，else子句不执行。

for语句

1.作用：

用来遍历可迭代对象的数据元素。

可迭代对象是指能依次获取数据元素的对象，例如：容器类。

2.语法：

```
for 变量列表 in 可迭代对象:
    语句块1
else:
    语句块2
```

3.说明:

else子句可以省略。
在循环体内用break终止循环时，else子句不执行。

range语句

1.作用:

用来创建一个生成一系列整数的可迭代对象（也叫整数序列生成器）。

2.语法:

```
range(开始点，结束点，间隔)
```

3.说明:

函数返回的可迭代对象可以用for去除其中的元素

跳转语句

break语句

在循环结构中，使用break语句即可跳出当前循环体，从而中断当前循环。

continue语句

在循环结构中，continue不是立即跳出循环体，而是跳过当次循环，回到条件测试部分，继续执行循环。

容器类型

通用操作

数学运算符

1.+：用于拼接两个容器

2.+=：用原容器和右侧容器拼接，并重新绑定变量

3.*：重复生成容器元素

4.*=：用原容器生成重复元素，并重新绑定变量

5.< <= > >= == != :一次比较两个容器中元素，一旦不同，则按编码比较大小。

成员运算符

1.语法：

```
数据 in 序列
数据 not in 序列
```

2.作用

如果在指定的序列中找到值，返回bool类型。

索引index

1.作用：访问容器元素

2.语法：容器[整数]

3.说明：

- 正向索引从0开始，第二个索引位1，最后一个为len()
- 反向索引从-1开始，-1代表最后一个，-2代表倒数第二个

切片slice

1.作用：

从容器中取出相应的元素重新组成一个容器。

2.语法：

```
容器[(开始索引):(结束索引)(:(步长))]
```

3.说明：

- 小括号()括起的部分代表可省略
- 结束索引不包含该位置元素
- 步长是切片每次获取完当前元素后移动的偏移量

内建函数

1.len(x)返回序列的长度

2.max(x)返回序列的最大值元素

3.min(x)返回序列的最小值元素

4.sum(x)返回序列中所有元素的和（元素必须是数值类型）

字符串

字符串定义

由一系列字符组成的不可变序列容器，存储的是字符的编码值。

编码

1.字节byte：计算机最小存储单位，等于8位bit。

2.字符：单个的数字，文字与符号。

3.字符集（码表）：存储字符与二进制序列的对应关系。

4.编码：将字符转换为对应的二进制序列的过程。

5.解码：将二进制序列转换为对应的字符的过程。

6.编码方式：

- ASCII编码：包含英文、数字等字符，每个字符1个字节。
- GBK编码：兼容ASCII编码，包含21003个汉字； 英文1个字节，汉字2个字节。
- Unicode字符集：国际统一编码，旧字符集每个字符2字节，新字符集4字节
- UTF-8编码：Unicode的存储与传输方式，英文1字节，汉字3字节。

相关函数

1.ord(字符串)：返回该字符串的Unicode码。

2.chr(整数)：返回该整数对应的字符串。

字面值

单引号和双引号的区别

1.单引号内的双引号不算结束符

2.双引号内的单引号不算结束符

三引号作用

1.换行会自动转换为换行符\n

2.三引号内可包含单引号和双引号

3.作为文档字符串

转义字符

1.改变字符的原始含义。

\' \" \""" \n \\ \t \0

2.原始字符串：取消转移。

a = r" C:\newfile\test.py"

字符串格式化

1.定义：

生成一定格式的字符串

2.语法：

```
字符串%(变量)
"我的名字是%s，年龄是%d"%(name, age)
```

3.类型码：

- %s 字符串
- %d 整数
- %f 浮点数

列表list

列表定义

由一系列变量组成的可变序列容器

列表基础操作

1.创建列表：

```
列表名=[]
列表名=list(可迭代对象)
```

2.添加元素：

```
列表名.append(元素)
列表名.insert(索引, 元素)
```

3.定位元素：索引、切片

4.遍历列表：

正向：

```
for 变量名 in 列表名:
    变量名就是元素
```

反向：

```
for 索引名 in range(-1, len(列表名)-1, -1):
    列表名[索引名]就是元素
```

5.删除元素：

```
del 列表名[索引或切片]
```

深拷贝和浅拷贝

浅拷贝：复制过程中，只复制一层变量，不会复制深层变量绑定的对象和复制过程。

深拷贝：复制整个依赖的变量。

列表VS字符串

1.列和字符串都是列表，元素之间有先后顺序关系。

2.字符串是不可变的序列，列表是可变的序列。

3.字符串中每个元素只能存储字符，而列表可以存储任意类型。

4.列表和字符串都是可迭代对象。

5.函数：

- 将多个字符串拼接为一个。

```
result = "连接符".join(列表)

result = " ".join(["a", "b", "c"])  # "a b c"
```

- 将一个字符串拆分为多个。

```
列表 = "字符串".split("分隔符")

list01 = "a-b-c-d".split("-")  # ["a", "b", "c", "d"]
```

列表推导式

1.定义：

使用简易方法，将可迭代对象转换为列表。

2.语法：

```
变量 = [表达式 for 变量 in 可迭代对象]

变量 = [表达式 for 变量 in 可迭代对象 if 条件]
```

3.说明：

如果if真值表达式的布尔值为False，则可迭代对象生成的数据将被丢弃。

列表推导式嵌套

1.语法：

```
变量 = [表达式 for 变量1 in 可迭代对象1 for 变量2 in 可迭代对象2]
```

元组 tuple

元组定义

1.由一系列变量组成的不可变序列容器。
2.不可变是指一但创建，不可以再添加/删除/修改元素。

元组基础操作

1.创建空元组：

```
元组名 = ()
元组名 = tuple()
```

2.创建非空元组：

```
元组名 = (20,)
元组名 = (1, 2, 3)
元组名 = 100,200,300
元组名 = tuple(可迭代对象)
```

3.获取元素：

```
变量 = 元组名[索引]
变量 = 元组名[切片] # 赋值给变量的是切片所创建的新列表
```

4.遍历元组：

正向：

```
for 变量名 in 列表名:
```

反向：

```
for 索引名 in range(len(列表名)-1,-1,-1):
```

元组与列表

1.元组与列表都可以存储一系列变量，由于列表会预留内存空间，所以可以增加元素。

2.元组会按需分配内存，所以如果变量数量固定，建议使用元组，因为占用空间更小。

- 列表（可变）：预留空间
- 元组（只读）：按需分配

3.应用：

变量交换的本质就是创建元组：

```
x, y = (y, x)
```

格式化字符串的本质就是创建元祖：

```
"姓名:%s, 年龄:%d" % (“tarena”, 15)
```

字典dict

字典定义

1.由一系列键值对组成的可变散列容器。

2.散列：对键进行哈希运算，确定在内存中的存储位置，每条数据存储无先后顺序。

3.键必须惟一且不可变(字符串/数字/元组)，值没有限制。

字典基础操作

1.创建字典：

```
字典名 = {键1: 值1, 键2: 值2}
字典名 = dict(可迭代对象)
```

2.添加/修改元素：

语法:

```
字典名[键] = 数据
```

说明:

- 键不存在，创建记录。
- 键存在，修改值。

3.获取元素：

```
变量 = 字典名[键] # 没有键则错误
```

4.遍历字典：

```
for 键名 in 字典名:
字典名[键名]

for 键名,值名 in 字典名.items():
语句
```

5.删除元素：

```
del 字典名[键]
```

字典推导式

1.定义：

使用简易方法，将可迭代对象转换为字典。

2.语法:

```
{键:值 for 变量 in 可迭代对象}
{键:值 for 变量 in 可迭代对象 if 条件}
```

字典VS列表

- 字典

1.查找和插入的速度极快，不会随着key的增加而增加

2.需要占用大量的内存，内存浪费多

- 列表

1.查找和插入的时间随着元素的增加而增加

2.占用空间小，浪费内存很少

集合set

集合定义

1.由一系列不重复的不可变类型变量(元组/数/字符串)组成的可变散列容器。

2.相当于只有键没有值的字典(键则是集合的数据)。

集合基础操作

1.创建空集合:

```
集合名 = set()
集合名 = set(可迭代对象)
```

2.创建具有默认值集合:

```
集合名 = {1, 2, 3}
集合名 = set(可迭代对象)
```

3.添加元素:

```
集合名.add(元素)
```

4.删除元素:

```
集合名.discard(元素)
```

集合的运算

1.交集&：返回共同元素。

```
s1 = {1, 2, 3}
s2 = {2, 3, 4}
s3 = s1 & s2 # {2, 3}
```

2.并集：返回不重复元素

```
s1 = {1, 2, 3}
s2 = {2, 3, 4}
s3 = s1 | s2 # {1, 2, 3, 4}
```

3.差集-：返回只属于其中之一的元素

```
s1 = {1, 2, 3}
s2 = {2, 3, 4}
s1 - s2 # {1} 属于s1但不属于s2
```

补集^：返回不同的的元素

```
s1 = {1, 2, 3}
s2 = {2, 3, 4}
s3 = s1 ^ s2 # {1, 4} 等同于(s1-s2 | s2-s1)
```

4.子集<：判断一个集合的所有元素是否完全在另一个集合中

5.超集>：判断一个集合是否具有另一个集合的所有元素

```
s1 = {1, 2, 3}
s2 = {2, 3}
s2 < s1 # True
s1 > s2 # True
```

6.相同或不同== !=：判断集合中的所有元素是否和另一个集合相同。

```
s1 = {1, 2, 3}
s2 = {3, 2, 1}
s1 == s2 # True
s1 != s2 # False
```

子集或相同,超集或相同 <= >=

集合推导式

1.定义：

使用简易方法，将可迭代对象转换为集合。

2.语法:

```
{表达式 for 变量 in 可迭代对象}
{表达式 for 变量 in 可迭代对象 if 条件}
```

固定集合frozenset

固定集合定义

不可变的集合

固定集合的作用

固定集合可以作为字典的键，还可以作为集合的值。

固定集合基础操作

创建固定集合：

```
frozen(可迭代对象)
```

固定集合的运算

等同于set

函数function

pycharm相关设置

1."代码自动完成"时间延时设置

File-->Settings-->Editor-->General-->Code Editing

2.快捷键

Ctrl + P 参数信息（在方法中调用参数）
Ctrl + Q 快速查看文档
Ctrl + Alt + M 提取方法

函数定义

1.用于封装一个特定的功能，表示一个功能或者行为。

2.函数是可以重复执行的语句块, 可以重复调用。

函数的作用

提高代码的可重用性和可维护性（代码层次结构更清晰）。

定义函数

1.语法：

```
def 函数名(形式参数):
    函数体
```

2.说明：

- def关键字：全称是define，意为”定义”。
- 函数名：对函数体中语句的描述，规则与变量名相同。
- 形式参数：方法定义者要求调用者提供的信息。
- 函数体：完成该功能的语句。

3.函数的第一行语句建议使用文档字符串描述函数的功能与参数。

调用函数

1.语法：

```
函数名(实际参数)
```

2.说明：根据形参传递内容。

返回值

1.定义：

方法定义者告诉调用者的结果。

2.语法：

```
return 数据
```

3.说明：

return后没有语句，相当于返回None。
函数体没有return，相当于返回None。

可变/不可变类型在传参时的区别

1.不可变类型参数有：

- 数值型（整数，浮点数，复数）
- 布尔值bool
- None空值
- 字符串str
- 元组tuple
- 固定集合frozenset

2.可变类型参数有：

- 列表list
- 字典dict
- 集合set

3.传参说明：

不可变类型的数据传参时，函数内部不会改变原数据的值
可变类型的数据传参时，函数内部可以改变原数据。

函数参数

实参传递方式argument

位置传参

定义：实参与形参的位置依次对应。

序列传参

定义：实参用*将序列拆解后与形参的位置一次对应。

关键字传参

定义：实参根据形参的名字进行对应。

字典关键字传参

1.定义：实参用**将字典拆解后与形参的名字进行对应。

2.作用：配合形参的缺省参数，可以使调用者随意传参。

形参定义方式parameter

缺省参数

语法：

```
def 函数名(形参名1=默认实参1, 形参名2=默认实参2, ...):
    函数体
```

说明：

- 缺省参数必须自右至左依次存在，如果一个参数有缺省参数，则其右侧的所有参数都必须有缺省参数。
- 缺省参数可以有0个或多个，甚至全部都有缺省参数。

位置形参

语法：

```
def 函数名(形参名1, 形参名2, ...):
    函数体
```

星号元组形参

语法：

```
def 函数名(*元组形参名):
    函数体
```

作用：

收集多余的位置传参。

说明:

- 一般命名为'args'
- 形参列表中最多只能有一个

命名关键字形参

语法:

```
def 函数名(*, 命名关键字形参1, 命名关键字形参2, ...):
    函数体

def 函数名(*args, 命名关键字形参1, 命名关键字形参2, ...):
    函数体
```

作用:

强制实参使用关键字传参

双星号字典形参

语法:

```
def 函数名(**字典形参名):
    函数体
```

作用:

收集多余的关键字传参

说明:

- 一般命名为'kwargs'
- 形参列表中最多只能有一个

参数自左至右的顺序

位置形参 --> 星号元组形参 --> 命名关键字形参 --> 双星号字典形参

作用域LEGB

1.作用域:变量起作用的范围。

2.Local局部作用域:函数内部。

3.Enclosing外部嵌套作用域：函数嵌套。

4.Global全局作用域:模块(.py文件)内部。

5.Builtin内置模块作用域: builtins.py 文件。

变量名的查找规则

1.由内到外：L --> E --> G --> B

2.在访问变量是，先查找本地变量，然后时包裹此函数外部的函数内部的变量，然后是全局变量，最后是内置变量

局部变量

1定义在函数内部的变量(形参也是局部变量)

2.只能在函数内部使用

3.调用函数时才被创建，函数结束后自动销毁

全局变量

1.定义在函数外部，模块内部的变量。

2.在整个模块（.py文件）范围内访问（但函数内不能将其直接赋值）。

global语句

1.作用：

在函数内部修改全局变量。

在函数内部定义全局变量（全局声明）。

2.语法：

```
global 变量1, 变量2, ...
```

3.说明

在函数内直接为全局变量赋值，视为创建新的局部变量。
不能先声明局部的变量，再用global 声明为全局变量。

nonlocal语句

1.作用：

在内层函数修改外层嵌套函数内的变量

2.语法

```
nonlocal 变量名1, 变量名2, …
```

3.说明

在被嵌套的内函数中进行使用