CTENETのPython笔记II

Python基础语法

数字

- 整数`int`
    - `Python3`开始不再区分 long 、int ，long 被重命名为 int，所以只有 int 类型了
    - 进制表示：
        - 十进制10
        - 十六进制0x10
        - 八进制0o10
        - 二进制0b10
    - bool 类型，有2个值 True、False
- 浮点数 float
    - 1.2、3.1415、-0.12、1.46e9等价于科学计数法1.46*109
- 复数 complex
    - 1+2j 或 +2J

字符串

- 使用‘ ”单双引号引用的字符的序列
- '''和""" 单双三引号，可以跨行、可以在其中自由的使用单双引号
- r 前缀：在字符串前面加上 r 或者 R 前缀，表示该字符串不做特殊的处理
- f 前缀：3.6版本开始，新增 f 前缀，格式化字符串

转义序列

- `\\` `\t` `\r` `\n` `\'` `\"`
- 上面每一个转义字符只代表一个字符，例如`\t`显示时占了4个字符位置，但是它是一个字符

- 前缀 r，把里面的所有字符当普通字符对待，则转义字符就不转义了

转义：让字符不再是它当前的意义，例如 \t，t 就不是当前意义字符 t 了，而是被 \ 转成了 tab 键

续行

- 在行尾使用 \，注意 \ 之后除了紧跟着换行之外不能有其他字符
- 如果使用各种括号，认为括号内是一个整体，其内部跨行不用 \

标识符

标识符

1. 一个名字，用来指代一个值
2. 只能是字母、下划线和数字
3. 只能以字母或下划线开头
4. 不能是 python 的关键字，例如 def、class 就不能作为标识符
5. Python 是大小写敏感的

python3.9 中关键字open in new window

```
False      await      else       import     pass
None       break      except     in         raise
True       class      finally    is         return
and        continue   for        lambda     try
as         def        from       nonlocal   while
assert     del        global     not        with
async      elif       if         or         yield
```

标识符约定：

- 不允许使用中文，也不建议使用拼音
- 不要使用歧义单词，例如class_
- 在python中不要随便使用下划线开头的标识符

常量

- 一旦赋值不能改变值的标识符
- python中无法定义常量

字面常量

- 一个单独的不可变量，例如 12、"abc" 、'2341356514.03e-9'

变量

- 赋值后，可以改变值的标识符

标识符本质

每一个标识符对应一个具有数据结构的值，但是这个值不方便直接访问，程序员就可以通过其对应的标 识符来访问数据，标识符就是一个指代。一句话，标识符是给程序员编程使用的。

False 等价

对象/常量	值
""	假
“string”	真
0	假
>=1	真
<=-1	真
()空元组	假
[]空列表	假
{}空字典	假
None	假

False 等价布尔值，相当于`bool(value)`

- 空容器
    - 空集合 set
    - 空字典 dict
    - 空列表 list
    - 空元组 tuple

- 空字符串
- None
- 0

运算符 Operator

算数运算符

+、-、*、/、//向下取整整除、%取模、**幂

注：在`Python2`中/和//都是整除。

位运算符

&位与、|位或、^异或、<<左移、>>右移

~按位取反，包括符号位

比较运算符

==、!=、>、>=、<、<=

链式比较： 4 > 3 > 2

逻辑运算符

与and、或or、非not

逻辑运算符也是短路运算符

- and 如果前面的表达式等价为False，后面就没有必要计算了，这个逻辑表达式最终一定等价为 False 1 and '2' and 0 0 and 'abc' and 1
- or 如果前面的表达式等价为True，后面没有必要计算了，这个逻辑表达式最终一定等价为True 1 or False or None
- 特别注意，返回值。返回值不一定是 bool 型
- 把最频繁使用的，做最少计算就可以知道结果的条件放到前面，如果它能短路，将大大减少计算量

赋值运算符

a = min(3, 5)

+=、 -= 、*=、/=、%=、//= 等

x = y = z = 10

成员运算符

in、not in

身份运算符

is、is not

运算符优先级

- 单目运算符 > 双目运算符
- 算数运算符 > 位运算符 > 比较运算符 > 逻辑运算符
    - -3 + 2 > 5 and 'a' > 'b'

搞不清楚就使用括号。长表达式，多用括号，易懂、易读。

表达式

由数字、符号、括号、变量等的组合。有算数表达式、逻辑表达式、赋值表达式、lambda 表达式等 等。

Python 中，赋值即定义。Python 是动态语言，只有赋值才会创建一个变量，并决定了变量的类型和 值。

如果一个变量已经定义，赋值相当于重新定义。

内建函数

内建函数	函数签名	说明
print	print(value, ..., sep=' ', end='\n')	将多个数据输出到控制台，默认使用空格分隔、\n 换行
input	input([prompt])	在控制台和用户交互，接收用户输入，并返回字符串
int	int(value)	将给定的值，转换成整数。int 本质是类
str	str(value)	将给定的值，转换成字符串。str 本质是元类
type	type(value)	返回对象的类型。本质是元类
isinstance	isinstance(obj, class_or_tuple)	比较对象的类型，类型可以是 obj 的基类

```
print(1,2,3,sep='\n', end='***')

type(1) # 返回的是类型，不是字符串
type('abc') # 返回的是类型，不是字符串
type(int) # 返回type，意思是这个int类型由type构造出来
type(str) # 返回type，也是类型
type(type) # 也是type

print(isinstance(1, int))
print(isinstance(False, int)) # True
```

Python程序控制

顺序

- 按照先后顺序一条条执行 例如，先洗手，再吃饭，再洗碗

分支

- 根据不同的情况判断，条件满足执行某条件下的语句 例如，先洗手，如果饭没有做好，玩游戏；如果饭做好了，就吃饭；如果饭都

没有做，叫外卖

循环

- 条件满足就反复执行，不满足就不执行或不再执行 例如，先洗手，看饭好了没有，没有好，一会来看一次是否好了，一会儿来看一次，直到饭 好了，才可是吃饭。这里循环的条件是饭没有好，饭没有好，就循环的来看饭好了没有

单分支

```
if condition:
    代码块

if 1<2: # if True
    print('1 less than 2') # 代码块
```

- condition必须是一个bool类型，这个地方有一个隐式转换bool(conditon)，相当于False等价
- if语句这行最后，会有一个冒号，冒号之后如果有多条语句的代码块，需要另起一行，并缩进
    - if、for、def、class等关键字后面都可以跟代码块
    - 这些关键后面，如果有一条语句，也可以跟在这一行后面。例如 `if 1>2: pass`

多分支

```
if condition1:
    代码块1
elif condition2:
    代码块2
elif condition3:
    代码块3
......
else:
    代码块


```

```
12
13 a = 5
14 if a<0:
15     print('negative')
16 elif a==0: # 相当于 a >= 0
17     print('zero')
18 else: # 相当于 a > 0
19     print('positive')
```

- 多分支结构，只要有一个分支被执行，其他分支都不会被执行
- 前一个条件被测试过，下一个条件相当于隐含着这个条件

```
1 # 嵌套
2 a = 5
3 if a == 0:
4     print('zero')
5 else:
6     if a < 0:
7     print('negative')
8 else:
9     print('positive')
```

while 循环

while循环多用于死循环，或者不明确知道循环次数的场景

```
1 while cond:
2     block
3
4
5 while True: # 死循环
6     pass
7
8 a = 10
9 while a: # 条件满足则进入循环
10    print(a)
11    a -= 1
```

for 语句

```
for element in iterable:
    block

for i in range(0, 10):
    print(i)
```

内建函数	函数签名	说明
range	range(stop) range(start, stop, [step])	返回惰性的对象 可以生成一个序列，遍历它就可以得到这个序列的一个个元素 前包后不包

continue

跳过当前循环的当次循环，继续下一次循环

```
for i in range(0, 10):
    if i % 2 != 0: continue
    print(i)
```

break

结束当前循环

```
# 计算1000以内的被7整除的前20个正整数
count = 0
for i in range(7, 1000, 7):
    print(i)
    count += 1
    if count >= 20:
        print(count)
        break
```

总结

- continue 和 break 是循环的控制语句，只影响当前循环，包括 while、for 循环
- 如果循环嵌套， continue 和 break 也只影响语句所在的那一层循环
- continue 和 break 只影响循环，所以`if cond: break`不是跳出 if，而是终止 if 外的 break 所在的循环
- 分支和循环结构可以嵌套使用，可以嵌套多层

else 字句

如果循环正常结束，else子句会被执行，即使是可迭代对象没有什么元素可迭代

```
for i in range(0): # 可迭代对象没有迭代
    pass
else:
    print('ok')

for i in range(0, 10):
    break
else:
    print('ok')

for i in range(0, 10):
    continue
else:
    print('ok')
```

有上例可知，一般情况下，循环正常执行，只要当前循环不是被break打断的，就可以执行else子句。 哪怕是range(0)也可以执行else子句。

三元表达式

在Python中，也有类似C语言的三目运算符构成的表达式，但python中的三元表达式不支持复杂的语句

```
真值表达式 if 条件表达式 else 假值表达式
```

三元表达式比较适合简化非常简单的if-else语句

```python
# 判断用户的输入的值，如果为空，输出"empty"，否则输出该值

value = input('>>>')
if value:
    print(value)
else:
    print('empty')

value = input('>>>')
print(value if value else 'empty')
```

Python数据类型

内建常用数据类型

- 数值型
    - int、float、complex、bool
- 序列 sequence
    - 字符串 str、字节序列 bytes、bytearray
    - 列表 list、元组 tuple
- 键值对
    - 集合 set、字典 dict

类型转换

- int、float、complex、bool 也可以当做内建函数对数据进行类型转换
- int(x) 返回一个整数
- float(x) 返回一个浮点数
- complex(x)、complex(x,y) 返回一个复数
- bool(x) 返回布尔值，前面讲过False等价的对象

封装和解构

基本概念

```
t1 = 1, 2
print(type(t1)) # 什么类型, tuple

t2 = (1, 2)
print(type(t2))
```

Python等式右侧出现逗号分隔的多值的时候，就会将这几个值封装到元组中。这种操作称为封装packing。

```
x, y = (1, 2)
print(x) # 1
print(y) # 2
```

Python 中等式右侧是一个容器类型，左侧是逗号分隔的多个标识符，将右侧容器中数据的一个个和左侧 标识符一一对应。这种操作称为解构unpacking。

从 Python3 开始，对解构做了很大的改进，现在用起来已经非常的方便快捷。

封装和解构是非常方便的提取数据的方法，在 Python、JavaScript 等语言中应用极广。

```
# 交换数据
x = 4
y = 5
t = x
x = y
y = t

# 封装和解构，交换
x = 10
y = 11
x, y = y, x
```

简单解构

```
# 左右个数相同  ,必须相等
a,b = 1,2
a,b = (1,2)
a,b = [1,2]
a,b = [10,20]
a,b = {10,20} # 非线性结构
a,b = {'a':10,'b':20} # 非线性结构也可以解构
[a,b] = (1,2)
[a,b] = 10,20
(a,b) = {30,40}
```

剩余变量解构

在 Python3.0 中增加了剩余变量解构（rest）。

```
a, *rest, b = [1, 2, 3, 4, 5]
print(a, b)
print(type(rest), rest) # <class 'list'> [2, 3, 4]
```

标识符 rest 将尽可能收集剩余的数据组成一个列表。

```
a, *_, b = [1, 2, 3, 4, 5]
print(_) # 在IPython中实验，_是最后一个输出值，这里将把它覆盖
_, *b, _ = [1, 2, 3]
print(_) # 第一个_是什么
print(b) # 是什么
print(_) # 第二个_是什么
```

_ 是合法的标识符，这里它没有什么可读性，它在这里的作用就是表示不关心这个变量的值，我不想要。有人把它称作 丢弃(Throwaway)变量。

线性数据结构

线性表

- 线性表（简称表），是一种抽象的数学概念，是一组元素的序列的抽象，它由有穷个元素组成（0 个或任意个）

- 顺序表：使用一大块连续的内存顺序存储表中的元素，这样实现的表称为顺序表，或称连续表
    - 在顺序表中，元素的关系使用顺序表的存储顺序自然地表示
- 链接表：在存储空间中将分散存储的元素链接起来，这种实现称为链接表，简称链表

列表如同地铁站排好的队伍，有序，可以插队、离队，可以索引。

链表如同操场上手拉手的小朋友，有序但排列随意。或者可以想象成一串带线的珠子，随意盘放在桌上。也可以离队、插队，也可以索引。