Python 入门 – 数据类型与运算符

数据类型和运算符

- 数据类型：整型、浮点型、布尔型、字符串、列表、元组、集合、字典
- 运算符：算术、赋值、比较、逻辑、成员、恒等运算符
- 内置函数、复合数据结构、类型转换
- 空格和样式指南

data type

```
Print()  /显示结果
** /取幂函数
% /取模运算  显示余数
//  /整除，向下取整
```

新建变量

右侧值赋值给左侧函数

命名规则：

1、只能在变量名称中使用常规字母、数字和下划线。不能包含空格，并且需要以字母或下划线开头。

2.、不能使用保留字或内置标识符：如下表

3、在 python 中，变量名称的命名方式是全部使用小写字母，并用下划线区分单词。

Keywords in Python programming language

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

```
x = 2
y = 3
z = 5
x, y, z = 2, 3, 5  #以上一致
mv = 700
mv = mv + 700 - 20 # mv += 700 - 20 左右一致 += 为递增赋值符号
```

目前水库的蓄水量（单位：立方米） reservoir_volume = 4.445e8 一场暴雨的降雨量（单位：立方米） rainfall = 5e6 考虑到流失的水量，将降雨量这个变量降低10%，将降雨量与蓄水量相加，暴雨后的一段时间内，雨水会不断流入水库，考虑这部分水量，需要将水库的蓄水量变量增加 5%，考虑到雨水蒸发情况，将蓄水量变量减小5%，水库里的水会用来浇灌干涸地区，所以需要将蓄水量减少 2.5e5 立方米，打印蓄水量变量的最新值

答案：447627500.0

```
# The current volume of a water reservoir (in cubic metres)
reservoir_volume = 4.445e8
# The amount of rainfall from a storm (in cubic metres)
rainfall = 5e6
# decrease the rainfall variable by 10% to account for runoff
rainfall *= (1 - 0.1)
# add the rainfall variable to the reservoir_volume variable
reservoir_volume += rainfall
# increase reservoir_volume by 5% to account for stormwater that flows
```

```
# into the reservoir in the days following the storm
reservoir_volume *= (1 + 0.05)
# decrease reservoir_volume by 5% to account for evaporation
reservoir_volume *= (1 - 0.05)
# subtract 2.5e5 cubic metres from reservoir_volume to account for water
# that's piped to arid regions.
reservoir_volume -= 2.5e5
# print the new value of the reservoir_volume variable
print(reservoir_volume)
```

整数(int)与浮点(float)

浮点转整数，直接切掉小数点后，没有四舍五入 49.7 – 49

你可以使用函数 type 检查数据类型：

```
>>> print(type(x))
int
>>> print(type(y))
float
```

代码可读性

```
print(x)
print( x - y )  #多数字，需要空格
```

运算符

布尔型运算符

布尔数据类型存储的是值 True 或 False，通常分别表示为 1 或 0。

比较运算符

符号使用情况	布尔型	运算符
5 < 3	False	小于
5 > 3	True	大于
3 <= 3	True	小于或等于
3 >= 5	False	大于或等于
3 == 5	False	等于
3 != 5	True	不等于

逻辑运算符

逻辑使用情况	布尔型	运算符
5 < 3 and 5 == 5	False	and - 检查提供的所有语句是否都为 True
5 < 3 or 5 == 5	True	or - 检查是否至少有一个语句为 True
not 5 < 3	True	not - 翻转布尔值

字符串

一般用‘‘，也可以””

如果‘’、“‘都用了，那就用 \’

空格用“ ”表示

len函数告诉我们字符数，可以返回一个具体值 len（‘xxxx’）就是数字符数有多少

```
house = 13 /int
street = 'TC' /str
town = 'BL' /str
//转换 int  - str
print(type(house))
house = str(house)
print(house)
//str - float
grams = '35.0'
grams = float(grams) // 直接上函数
```

转换

```
mon_sales = "121"
mon_sales = float(mon_sales)
tues_sales = "105"
tues_sales = float(tues_sales)
wed_sales = "110"
wed_sales = float(wed_sales)
thurs_sales = "98"
thurs_sales = float(thurs_sales)
fri_sales = "95"
fri_sales = float(fri_sales)
x = mon_sales + tues_sales + wed_sales + thurs_sales + fri_sales
x = str(x)
print("This week\'s total sales: " + x )
```

字符串方法

title（） – 将字符串的单词首字母大写

islower（） – 判断字符串是否小写

count（）– 判断一段字符串里重复了多少次

通过.来调用的一种函数

例如，lower()是一个字符串方法，对于一个叫 "sample string" 的字符串，它可以这样使用：sample_string.lower()

```
>>> my_string.islower()
True
>>> my_string.count('a')
2
>>> my_string.find('a')
3
```

list 列表 – 索引规则

顺序性查看 – 所有有序容器（例如列表）的起始索引都是 0

倒查从 –1 开始

使用负数从列表的末尾开始编制索引，其中 -1 表示最后一个元素，-2 表示倒数第二个元素，等等。

列表切片

可以使用切片功能从列表中提取多个值。在使用切片功能时，起始索引包含在内，终止索引排除在外

冒号表示从冒号左侧的起始值开始，到右侧的元素（不含）结束。从列表的开头开始，也可以省略起始值。返回到列表结尾的所有值，可以忽略最后一个元素。

```
lst_of_random_things[:2]  /从列表的开头开始，省略起始值
[1, 3.4]
lst_of_random_things[1:]  /列表结尾的所有值，忽略最后一个元素
[3.4, 'a string', True]
 >>> use list indexing to determine the number of days in month
month = 8
days_in_month = [31,28,31,30,31,30,31,31,30,31,30,31]
num_days = days_in_month[month - 1]
print(num_days)
```

in / not in 返回布尔值

表示某个元素是否存在于列表中，或者某个字符串是否为另一个字符串的子字符串。

```
>>> 'isa' in 'this is a string'
False
>>> 5 not in [1, 2, 3, 4, 6]
True
```

可变性和顺序
可变性是指对象创建完毕后，我们是否可以更改该对象，字符串是不可变的
字符串和列表都是有序的

实用的列表函数

- len() 返回列表中的元素数量。
- max() 返回列表中的最大元素。数字列表中的最大元素是最大的数字。字符串列表中的最大元素是按照字母顺序排序时排在最后一位的元素。如果列表包含不同的无法比较类型的元素，则max() 的结果是 undefined。
- min() 返回列表中的最小元素。它是 max() 函数的对立面，返回列表中的最小元素。
- sorted() 返回一个从最小到最大排序的列表副本，并使原始列表保持不变。
- join （） Join 是一个字符串方法，将字符串列表作为参数，并返回一个由列表元素组成并由分隔符字符串分隔的字符串。

```
new_str = "\n".join(["fore", "aft", "starboard", "port"])
print(new_str)
//输出 ， "\n" 是分隔符，"-"是连字符
fore
aft
starboard
port
```

append() 会将元素添加到列表末尾

```
letters = ['a', 'b', 'c', 'd']
letters.append('z')
print(letters)
//输出
['a', 'b', 'c', 'd', 'z']
```

元组：顺序 – 不可变
元组解包
将元组中的信息赋值给多个变量

```
dimensions = 52, 40, 100
length, width, height = dimensions
print("The dimensions are {} x {} x {}".format(length, width, height))
//元组解包，上下其实是一样的
length, width, height = 52, 40, 100
print("The dimensions are {} x {} x {}".format(length, width, height))
```

集合：无序 – 可变， 没有最后一个元素， set就是把重复的组合起来
集合的一个用途是快速删除列表中的重复项。
集合和列表一样支持 in 运算，支持add() 和pop() 前者添加，后者随机删除

```
numbers = [1, 2, 6, 3, 1, 1, 6]
unique_nums = set(numbers)
print(unique_nums)
//输出
{1, 2, 3, 6}
```

字典和恒等运算

字典 – 数据类型可变，存储的是唯一键到值的映射，字典用 { }
也可以使用 in 检查 值 是否在 字典中
get 也对应查询，get到就给出值，没有就给None
如果你预计查询有时候会失败，get 可能比普通的方括号查询更合适，因为错误可能会使程序崩溃。
恒等运算符

关键字	运算符
is	检查两边是否恒等
is not	检查两边是否不恒等

```
elements = {"hydrogen": 1, "helium": 2, "carbon": 6}
print(elements["helium"])  # 直接输出字典中某个值
elements["lithium"] = 3  # 在字典中插入一个新值
print("carbon" in elements)  # 检查carbon在不在字典中 —— true
print(elements.get("dilithium"))  # 字典里能不能get到这个 —— None
-----
n = elements.get("dilithium")
print(n is None)  # 使用运算符 is 检查某个键是否返回了 None —— True
print(n is not None) # 或者使用 is not 检查是否没有返回 None。 —— False
```

复合数据结构
嵌套结构

```
elements = {"hydrogen": {"number": 1,
                         "weight": 1.00794,
                         "symbol": "H"},
              "helium": {"number": 2,
                         "weight": 4.002602,
                         "symbol": "He"}}
helium = elements["helium"]  # get the helium dictionary，获取helium字典
hydrogen_weight = elements["hydrogen"]["weight"]  # get hydrogen's weight
定义hydrogen_weight的函数，获取h的w

#嵌套字典。向 elements 字典中的每个字典添加另一个条目 'is_noble_gas
elements = {'hydrogen': {'number': 1, 'weight': 1.00794, 'symbol': 'H'},
            'helium': {'number': 2, 'weight': 4.002602, 'symbol': 'He'}}
```

```
方法一
noble_dict = {'hydrogen': False, 'helium': True}  # 创建一个noble字典
for noble in noble_dict:
    elements[noble]['is_noble_gas'] = noble_dict[noble] # 循环查找
方法二
elements['hydrogen']['is_noble_gas'] = False
elements['helium']['is_noble_gas'] = True
print(elements['hydrogen']['is_noble_gas'])
print(elements['helium']['is_noble_gas'])
```