Python 基础语法学习笔记

はなとみず

2025 年 8 月 7 日

目录

1 注释

编程语言中，注释是给人看的，不会被电脑阅读. 在 Python 中，注释使用 # 或者 ''' ''' 来表示，后者可以用于多行注释.

```
# 这是单行注释
'''
这是多行注释
'''

print(123) # #后面的部分不会被电脑阅读
# print(456) 这行代码不会被执行
```

执行结果:

```
>>> # 这是单行注释
... '''
... 这是多行注释
... '''
...
... print(123) # #后面的部分不会被电脑阅读
... # print(456) 这行程序不会被执行
123
```

可以看到，只有在注释外面的 print(123) 被执行.

2 输入输出

输入输出需要实践，不然很难理解

2.1 输入

你可以使用 input() 来输入一行字符串. 注意，你一次性读入的是一行字符串. 如果你想要一行读入两个整型，你可以使用 a, b = map(int, input().split()). 解释一下，input() 是你读入的字符串，字符串中有个 split() 方法，可以按照空格把字符串切成多串字符串，然后 map() 来把字符串转化为整形，依次赋值给 a, b.

如果你要读入一行列表，你可以直接 a = list(map(int, input().split())).

可以运行以下代码感受一下:

```
buf = input()
print(type(buf))
tmp = buf.split()
print(tmp)
print(type(tmp))
a = list(map(int, tmp))
print(a)
```

2.2 输出

直接使用 `print()` 来输出.

`print()` 可以使用如下操作:

1. `print("awa", end="qwq")` 这里的 `end="qwq"` 是把结束标志换为 `"qwq"`
2. `print("awa", 123)` 输出多个值，每个值用逗号分开
3. `print("%.2f" % (0.01))` 格式化输出，使用 `%f` 等关键字占位，然后在引号外面跟上 `%` 来补充，百分号后面是填充的值

建议多实践

3 变量与数据类型

3.1 变量

Python 中等号为赋值语句,可以把一个值赋予给一个变量,如下,如果没有声明 a 的值,`print()` 语句直接报错；声明 a 的值后就可以正常运行，并且 a 的值可以被多次修改，程序是顺序执行的,变量的值即为当前状态下最新赋的值. 如下：当 `a = 2` 被执行前，a 的值为 1，而执行 `a = 2` 后，a 的值变为 2.

```
>>> print(a)
Traceback (most recent call last):
File "<python-input-0>", line 1, in <module>
print(a)
      ^
NameError: name 'a' is not defined
>>> a = 1
>>> print(a)
1
>>> a = 2
>>> print(a)
2
```

3.2 数据类型

变量的类型可以是多样的,从简单的整型、浮点型、字符串、布尔值等，再到高级的列表、class 等. 目前，我们只需要认识最简单的四种类型，如下:

```
# 基本数据类型
integer = 42            # 整型
floatnum = 3.14         # 浮点型
string = "Hello"        # 字符串
boolean = True          # 布尔值 (True/False)

# 类型转换
strnum = str(123)       # "123"
intnum = int("456")     # 456
floatnum = float("7.8") # 7.8
```

需要注意的是，"123" 的类型是字符串. 只需要记住，无论是什么内容，只要是被引号包裹的，都是字符串类型. Python 语言中，单引号和双引号作用相同，只需要选择你喜欢的即可.

但是请注意，以下的写法会导致错误:

```
s = 'I'd like a fish'
print(s)
```

因为计算机会认为字符串为 'I'，而后面的 d like a fish 被视为语法错误，导致运行错误. 此时你有两种解决方法:

```
s = "I'd like a fish"  # 直接使用双引号来规避
s = 'I\'d like a fish' # 或者可以使用反斜杠来转义字符串内的引号，来告诉计算机，这个引号是字符串的一部分.
```

3.3 字符串

字符串是很有意思的一个数据结构，在学习 C/C++ 后会有更加底层的理解，我们现在不需要.

规定一个字符串 s = "Hello Python!"，我们有如下操作:

1. len(s): 求出字符串的长度
2. "string" in s: 查询前面的字符串 (这里的 "string") 是否在 s 内出现过，是则为 True，否则为 False
3. s[1:5]: 切片，取出下标为 $[1,5)$ 的字符

运行一下看看:

```
>>> s = "Hello Python!"
>>> print(len(s))
13
>>> print("Hell" in s)
True
>>> print("awa" in s)
False

>>> print(s[1:5])
ello
>>> print(s[::-1])
!nohtyP olleH
>>>
```

这里比较重要的是切片的操作，再详细说一下切片.

s[a:b:c]，表示从 a 下标开始，切到最后一个小于 b 下标的位置，每 c 个元素取 1 个. 如果 $c < 0$ 那么就是从后往前取，此时必须满足 $a \geq b$，否则取出来就是空字符串. 特别的，如果 $c = 0$，程序直接报错. 没的商量.

如果不填（如 s[::]）则默认情况下 $a = 0, b = len(s), c = 1$.

4 运算符

```python
# 算术运算符
print(10 + 3)   # 13
print(10 - 3)   # 7
print(10 * 3)   # 30
print(10 / 3)   # 3.333...
print(10 // 3)  # 3 (整除)
print(10 % 3)   # 1 (取余)
print(2 ** 3)   # 8 (幂运算)

# 比较运算符
print(3 > 2)    # True
print(3 == 2)   # False
print(3 != 2)   # True

# 逻辑运算符
print(True and False)  # False, and 表示当且仅当左右两边都为 True 时才是 True
print(True or False)   # True, or 表示只要有一边为 True 就是 True
print(not True)        # False, 取反
```

5 数据结构

5.1 列表

列表的基础操作

```python
# 列表的基础操作
fruits = ["apple", "banana", "cherry"]
fruits.append("orange")   # 添加元素
fruits[1] = "mango"       # 修改元素
print(fruits[0:2])        # ['apple', 'mango']

# 其他的常见操作:

a = [0] * 100 # 声明一个长度为 100 的列表, 其中所有的值都为 0
a = [0 for _ in range(100)] # 结果同上, 这个写法题目出过, 看得懂就可以

# 列表的遍历:
a = [1, 2, 3, 4]
for i in a:
    print(i, end=", ")

for i in range(len(a)):
    print(a[i], end=", ")
# 以上两种写法输出结果都是 1, 2, 3, 4
# 区别在于, 第一种写法直接遍历列表中的元素, 而第二种写法遍历的是列表的下标
# 体会一下
```

6 流程控制

6.1 顺序结构程序

一般都程序都是直接顺序结构运行，就是从上到下，一行一行跑，如下：

```
>>> a = 1
>>> print(a)
1
>>> b = a
>>> print(b)
1
>>> a = 2
>>> print(a)
2
```

6.2 分支结构程序

Python 使用形如 if ... else ... 的语句来控制分支结构，格式如下：

```
if condition:
    code here
else:
    code here

if condition:
    code here
elif condition2:
    code here
else:
    code here
```

```
# 细节处理：条件判断按顺序执行，当某个条件分支被触发后，后续分支将不再检查
if 条件1:
    print("awa") # 条件 1 满足时运行这个语句
elif 条件2:
    print("qwq") # 当条件 1 满足时，上一个 print("awa") 被执行，然后直接跳出了这个 if ... elif ... else 的框架，这样，这行代码就算是条件 2 满足了，也不会被执行. 所以这里当且仅当条件 1 不满足且条件 2 满足时执行
else:
    print("uwu") # 如果上面两个 print 都没有被执行，即条件 1 条件 2 都不满足时，才执行 else

# 区分一下，上述条件为在同一个 if ... elif ... else 中，下面的写法执行的结果截然不同
if 条件1:
    print("awa") # 条件 1 满足时执行
else:
    print("qwq") # 条件 1 不满足时就执行
if 条件2:
```

```
    print("uwu") # 条件 2 满足时就执行
```

6.3 循环结构程序

Python 中有两种循环结构，分别是 for 循环和 while 循环

```
for variable in list:
    process

while condition:
    process
```

for 循环中 variable 为循环变量，一般我们命名为 i，每次循环从 list 中取下一个值作为循环变量，一般 list 我们使用 range(n)，她可以生成一个 $[0, n)$ 的整数列表.

while 循环中只要 condition 成立，循环就会一直下去，可以自己运行一下下面的代码理解.

```
# for循环
for i in range(3):        # 0, 1, 2
    print("Count: %d" % i)

# while循环
count = 0
while count < 3:
    print(count)
    count += 1
```

另外，我们还有 break 和 continue 两个关键字来控制循环. 前者为直接结束整个循环，后者则为结束本次循环，进入下次循环.

给个形象的例子:

```
for i in range(1, 10):
    if i == 5:
        continue
    print(i)
for i in range(1, 10):
    if i == 5:
        break
    print(i)
```

运行后发现，上面的输出了 $1, 2, 3, 4, 6, 7, 8, 9$，而下面的输出了 $1, 2, 3, 4$，体会一下两者的区别.

7 函数

函数使用 def 关键字来声明，形式如下:

```
def func(var1, var2):
    return;
```

函数的组成为函数名、参数、返回值类型、函数体. 解释一下:

- 函数名: 函数的名字，上述示例中的 func

- 参数：写在括号内的东西，可以没有，上述的 var1，var2 就是参数. 参数只有在函数内才是有效的，离开了函数参数会被立刻销毁. 传入的可以是变量，可以是任何类型. 但是如果声明的时候写了参数，调用的时候必须传入需要的参数（比如声明了 5 个参数，你调用必须传 5 个进去）. 高考题中需要根据函数体的内容来推断参数的数据类型.
- 返回值类型：决定了这个函数会返回什么东西，通常不用写
- 函数体：函数里面的代码，要执行什么

函数在声明的时候函数体不会被执行，给个例子理解一下

```
def f():
    print("珂朵莉是世界上最幸福的女孩")

print("全体目光向我看齐")
f()
```

运行后可以发现，函数体只有在被调用时才会被执行. 需要注意的是，函数体内声明的变量在函数结束后会立刻被销毁，你在函数外面无法访问.

return 是独属于函数的一个关键字，它可以返回一个值，并且立刻结束整个函数. 在没有返回值的函数中可以不写

再给几个示例理解一下：

```
def fib(a):
    if a <= 0:
        return 0
    if a == 1 or a == 2:
        return 1 # 当这里被执行后，函数结束，下一行就不会执行
    return fib(a-1) + fib(a-2); # 函数可以自己调用自己，这里的原理是栈，暂时不要求了解
print(fib(20))

def add(a, b):
    return a + b
res = add(1, 2)
print(res)

def testa():
    print("begin function")
    return
    print("uwu") # 上一行 return 了，这行不会执行
testa()

def testb():
    print("begin")
    # 没有返回值的函数可以不写 return
testb()
```