

# 第一讲：基本类型-数据变量

## 今日内容：

1. 基本输入输出+注释的作用
2. 变量的作用
3. Debug工具
4. 认识数据类型
5. 格式化输出
6. 总结

## 1.开始你的Python开发之旅-基本输入输出

### 输出-第一个Python程序-HelloWorld

Python的第一个程序也从hello world开始吧

在pycharm里面输出字符串

字符串就是多个字符的集合，由双引号“”，或者单引号‘’包围列如：

字符串可以包含英文、数字、中文及各种符号

`print`输出字符串的格式如下：

```
print("字符串内容")
print('字符串内容')
```

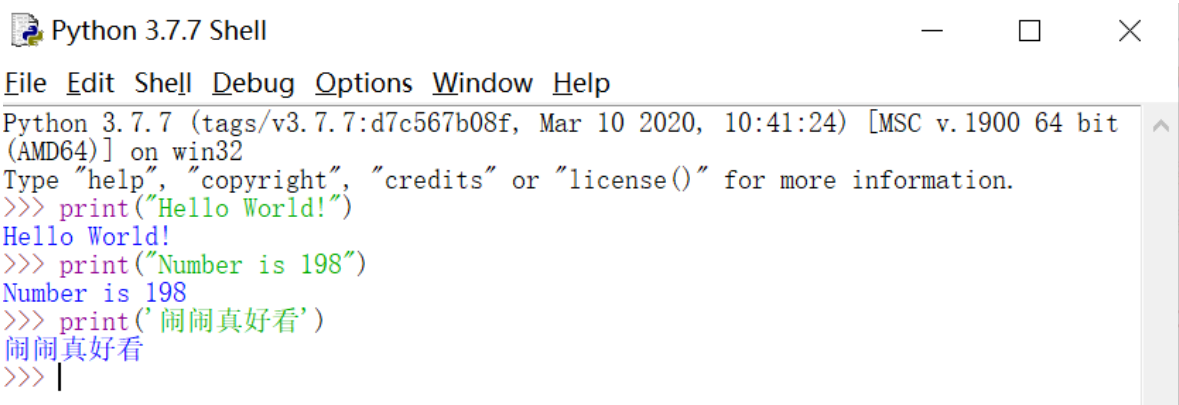
执行结果：`"C:\Program Files\Python35\python.exe"` `D:/python/Day1/test/HelloWorld.py`  
Hello world!

Process finished with exit code 0

### print 用法举例

```
print("Hello world!") #输出英文
print("Number is 198") #输出数字
print('闹闹真好看')#输出中文
```

在 IDLE 下的演示效果：

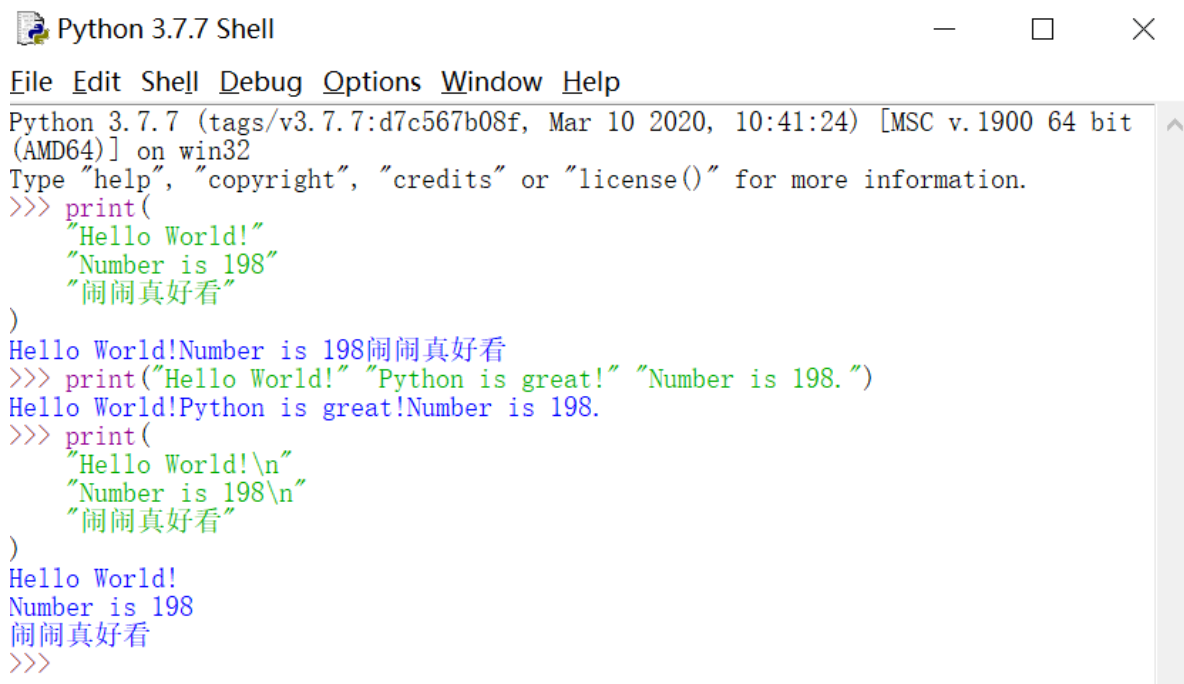


```
Python 3.7.7 Shell
File Edit Shell Debug Options Window Help
Python 3.7.7 (tags/v3.7.7:d7c567b08f, Mar 10 2020, 10:41:24) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> print("Hello World!")
Hello World!
>>> print("Number is 198")
Number is 198
>>> print('闹闹真好看')
闹闹真好看
>>> |
```

也可以将多段文本放在一个print函数中：

```
print(
    "Hello world!"
    "Number is 198"
    "闹闹真好看"
)
print("Hello world!" "Python is great!" "Number is 198.")
print(
    "Hello world!\n"
    "Number is 198\n"
    "闹闹真好看"
)
```

注意：同一个print函数的字符串之间不会自动换行，加上\n才能看到换行效果



```
Python 3.7.7 Shell
File Edit Shell Debug Options Window Help
Python 3.7.7 (tags/v3.7.7:d7c567b08f, Mar 10 2020, 10:41:24) [MSC v.1900 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> print(
    "Hello World!"
    "Number is 198"
    "闹闹真好看"
)
Hello World!Number is 198闹闹真好看
>>> print("Hello World!" "Python is great!" "Number is 198.")
Hello World!Python is great!Number is 198.
>>> print(
    "Hello World!\n"
    "Number is 198\n"
    "闹闹真好看"
)
Hello World!
Number is 198
闹闹真好看
>>>
```

## 在屏幕上输出数字

print除了能输出字符串，还能输出数字，将数字或者数学表达式直接放在print中就可以输出，如下所示：

```
print( 100 )
print( 65 )
print( 100 + 12 )
print( 8 * (4 + 6) )
```

注意：输出数字时不能用引号包围，否则就变成了字符串，下面的写法就是一个反面教材，数学表达式会原样输出：

```
print("100+12")
运行结果是 100+12，而不是112
```

另外，和输出字符串不同，不能讲多个数字放在一个print函数中，例如，下面的写法就是错误的：

```
print(100 12 95)

print(
    80
    26
    205
)
```

## 总结:

---

Python程序的写法比较简单，直接书写功能代码即可，不用给他套上“外壳”，下面我们分别使用C语言，Java和Python输出hello world，让大家对比感受一下

使用C语言

```
#include <stdio.h>
int main()
{
    puts("hello world");
    return 0;
}
```

使用java:

```
public class HelloJava {
    public static void main(String[] args) {
        System.out.println("hello world");
    }
}
```

使用Python:

```
print("hello world")
```

## 输入:

在Python中，程序接收用户输入的数据的功能既是输入



## 输入的语法：

```
input('提示信息')
```

## 输入的特点

- 当程序执行到 input ，等待用户输入，输入完成之后才继续向下执行。
- 在Python中， input 接收用户输入后，一般存储到变量，方便使用。
- 在Python中， input 会把接收到的任意用户输入的数据都当做字符串处理

```
password = input('请输入您的密码：')  
  
print(f'您输入的密码是{password}')
```

# <class 'str'>

```
print(type(password))
```

## 控制台输出结果如下：

```
请输入你的密码1  
你输入的密码是1  
<class 'str'>
```

## 总结

输入功能-`input('提示文字')`

输入的特点:

- 一般讲- `input`接收的任何数据默认都是字符串数据类型

输入的规则:

1. 在python环境里面输入 `input()` 回车之后, python的环境就会"接受"你在键盘里面输入的东西,就相当于于是写字. 输入完毕,按下回车. python环境就会识别你输入完了,把你刚刚敲得东西都输出一遍
2. 大部分时候我们输入东西,不是随便输入的,而是要输入一个有意义的内容. 第二个规则,就是提醒你,你要输入啥?

## 1.2注释的作用

没有注释的代码

```
2048.py
83 def is_win(self):
84     return any(any(i >= self.win_value for i in row) for row in self.field)
85
86 def is_gameover(self):
87     return not any(self.move_is_possible(move) for move in actions)
88
89 def draw(self, screen):
90     help_string1 = '(W)Up (S)Down (A)Left (D)Right'
91     help_string2 = '      (R)Restart (Q)Exit'
92     gameover_string = '                GAME OVER'
93     win_string = '                YOU WIN!'
94     def cast(string):
95         screen.addstr(string + '\n')
96
97     def draw_hor_separator():
98         line = '+' + ('+-----' * self.width + '+')[1:]
99         separator = defaultdict(lambda: line)
100         if not hasattr(draw_hor_separator, "counter"):
101             draw_hor_separator.counter = 0
102         cast(separator[draw_hor_separator.counter])
103         draw_hor_separator.counter += 1
104
105     def draw_row(row):
106         cast(''.join('|{: ^5} '.format(num) if num > 0 else '|' for num in row
107
108     screen.clear()
109     cast('SCORE: ' + str(self.score))
110     if 0 != self.highscore:
111         cast('HGHSCORE: ' + str(self.highscore))
112     for row in self.field:
113         draw_hor_separator()
```

添加注释的代码 :

```

158 def main(stdscr):
159     def init():
160         #重置游戏棋盘
161         game_field.reset()
162         return 'Game'
163
164     def not_game(state):
165         #画出GameOver或者Win的界面
166         game_field.draw(stdscr)
167         #读取用户输入得到action, 判断是重启游戏还是结束游戏
168         action = get_user_action(stdscr)
169         responses = defaultdict(lambda: state) #默认是当前状态, 没有行为就会一直在当前界面循环
170         responses['Restart'], responses['Exit'] = 'Init', 'Exit' #对应不同的行为转换到不同的状态
171         return responses[action]
172
173     def game():
174         #画出当前棋盘状态
175         game_field.draw(stdscr)
176         #读取用户输入得到action
177         action = get_user_action(stdscr)
178
179         if action == 'Restart':
180             return 'Init'
181         if action == 'Exit':
182             return 'Exit'
183         if game_field.move(action): # move successful
184             if game_field.is_win():
185                 return 'Win'
186             if game_field.is_gameover():
187                 return 'Gameover'
188         return 'Game'

```

以#开头的就是  
注释

- 通过用自己熟悉的语言，在程序中对某些代码进行标注说明，这就是注释的作用，能够大大增强程序的可读性

## 注释的分类及语法

注释分为两类：单行注释 和 多行注释

- 单行注释

只能注释一行内容，语法如下：

```
# 注释内容
```

- 多行注释

可以注释多行内容，一般用在注释一段代码的情况，语法如下：

```

"""
    第一行注释
    第二行注释
    第三行注释
"""
'''
    注释1
    注释2
    注释3
'''

```

☒ 快捷键：ctrl + /

## 快速体验

- 单行注释

```
# 输出hello world
print('hello world')
print('hello Python') # 输出(简单的说明可以放到一行代码的后面，一般习惯代码后面添加两个空格再书写注释文字)
```

- 多行注释

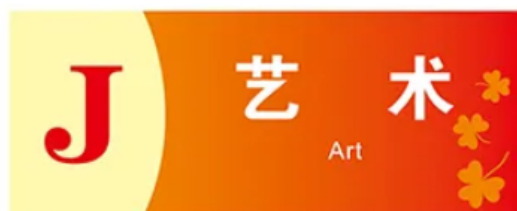
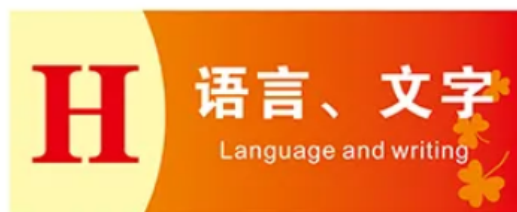
```
"""
    下面三行都是输出的作用，输出内容分别是：
    hello Python
    hello world
    hello itdalao
"""
print('hello Python')
print('hello world')
print('hello itdalao')
'''
    下面三行都是输出的作用，输出内容分别是：
    hello Python
    hello itcast
    hello itdalao
'''
print('hello Python')
print('hello world')
print('hello itdalao')
```

注意：解释器不执行任何的注释内容。

## 总结

- 注释的作用：用人类熟悉的语言对代码进行解释说明，方便后期维护
- 注释的分类：
  - 单行：# 注释内容，快捷键ctrl+/
    - 多行：""" 注释内容 """ 或 ''' 注释内容 '''
- 解释器不执行注释内容

## 一. 变量的作用



举例体验：我们去图书馆读书，怎么样快速找到自己想要的书籍呢？是不是管理员提前将书放到固定位置，并把这个位置进行了编号，我们只需要在图书馆中按照这个编号查找指定的位置就能找到想要的书籍。

这个编号其实就是把书籍存放的书架位置起了一个名字，方便后期查找和使用。程序中，数据都是临时存储在内存中，为了更快速的查找或使用这个数据，通常我们把这个数据在内存中存储之后定义一个名称，这个名称就是变量。

|      |      |      |    |    |      |    |      |    |
|------|------|------|----|----|------|----|------|----|
| num1 | num2 | num3 |    |    | num4 |    | num5 |    |
| ↓    | ↓    | ↓    |    |    | ↓    |    | ↓    |    |
| 10   | 20   | 30   | 10 | 40 | 20   | 50 | 40   | 10 |
|      |      |      |    |    |      |    |      |    |
|      |      |      |    |    |      |    |      |    |

变量就是一个存储数据的时候当前数据所在的内存地址的名字而已。



## 二. 定义变量

```
变量名 = 值
```

变量名自定义，要满足**标识符**命名规则。

2.1 标识符 标识符命名规则是Python中定义各种名字的时候的统一规范，具体如下：

- 由数字、字母、下划线组成
- 不能数字开头
- 不能使用内置关键字
- 严格区分大小写

| and      | exec    | not    |
|----------|---------|--------|
| assert   | finally | or     |
| break    | for     | pass   |
| class    | from    | print  |
| continue | global  | raise  |
| def      | if      | return |
| del      | import  | try    |
| elif     | in      | while  |
| else     | is      | with   |
| except   | lambda  | yield  |

## 2.2 命名习惯

- 见名知义。
- 大驼峰：即每个单词首字母都大写，例如：MyName。
- 小驼峰：第二个（含）以后的单词首字母大写，例如：myName。
- 下划线：例如：my\_name。

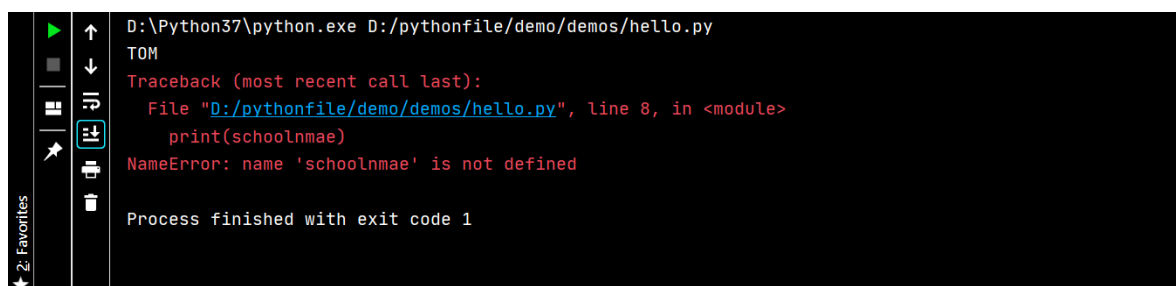
## 2.3 使用变量

```
my_name = '闹闹'
print(my_name)

schoolName = '中职通程序员'
print(schoolName)
```

## 2.4 认识bug

所谓bug，就是程序中的错误。如果程序有错误，需要程序员排查问题，纠正错误。



```
D:\Python37\python.exe D:/pythonfile/demo/demos/hello.py
TOM
Traceback (most recent call last):
  File "D:/pythonfile/demo/demos/hello.py", line 8, in <module>
    print(schoolmae)
NameError: name 'schoolmae' is not defined

Process finished with exit code 1
```

### 三. Debug工具

Debug工具是PyCharm IDE中集成的用来调试程序的工具，在这里程序员可以查看程序的执行细节和流程或者调解bug。

Debug工具使用步骤：

1. 打断点
2. Debug调试

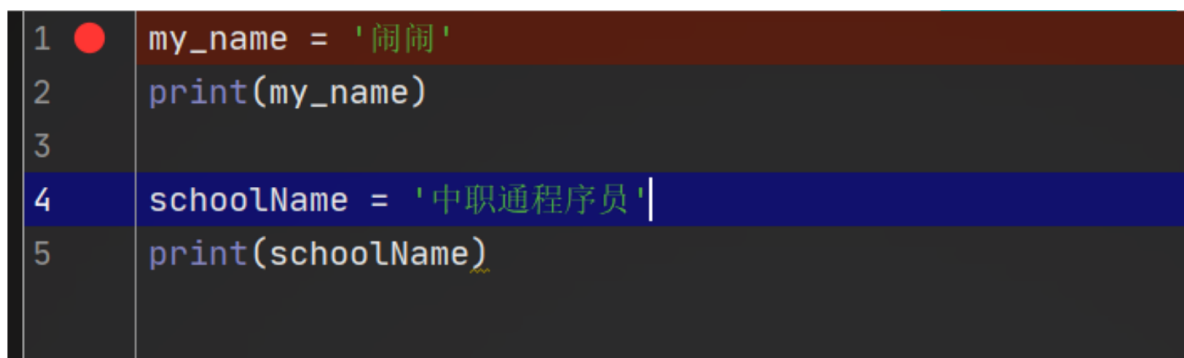
#### 3.1 打断点

- 断点位置

目标要调试的代码块的第一行代码即可，即一个断点即可。

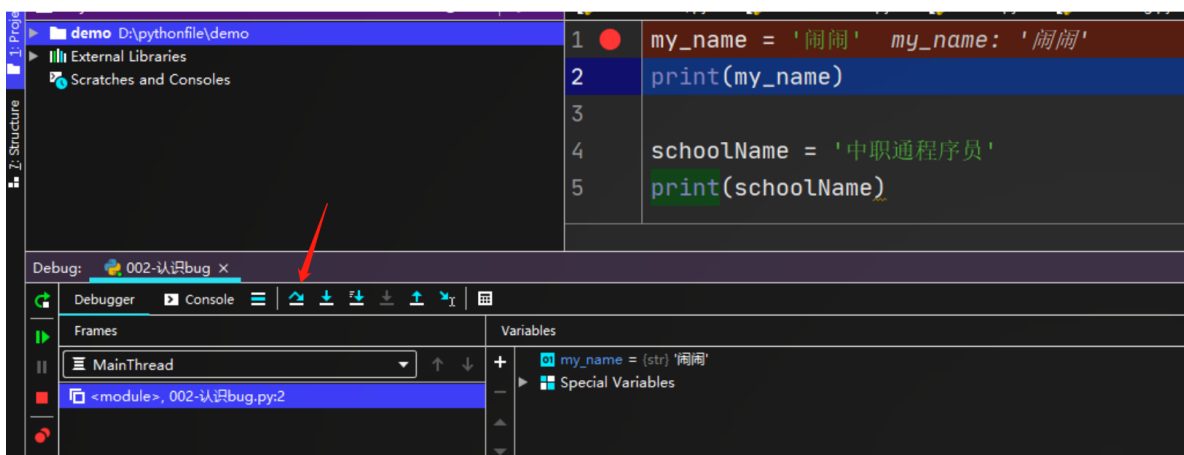
- 打断点的方法

单击目标代码的行号右侧空白位置。



#### 3.2 Debug调试

打成功断点后，在文件内部任意位置 — 右键 -- Debug'文件名' — 即可调出Debug工具面板 -- 单击Step Over/F8，即可按步执行代码

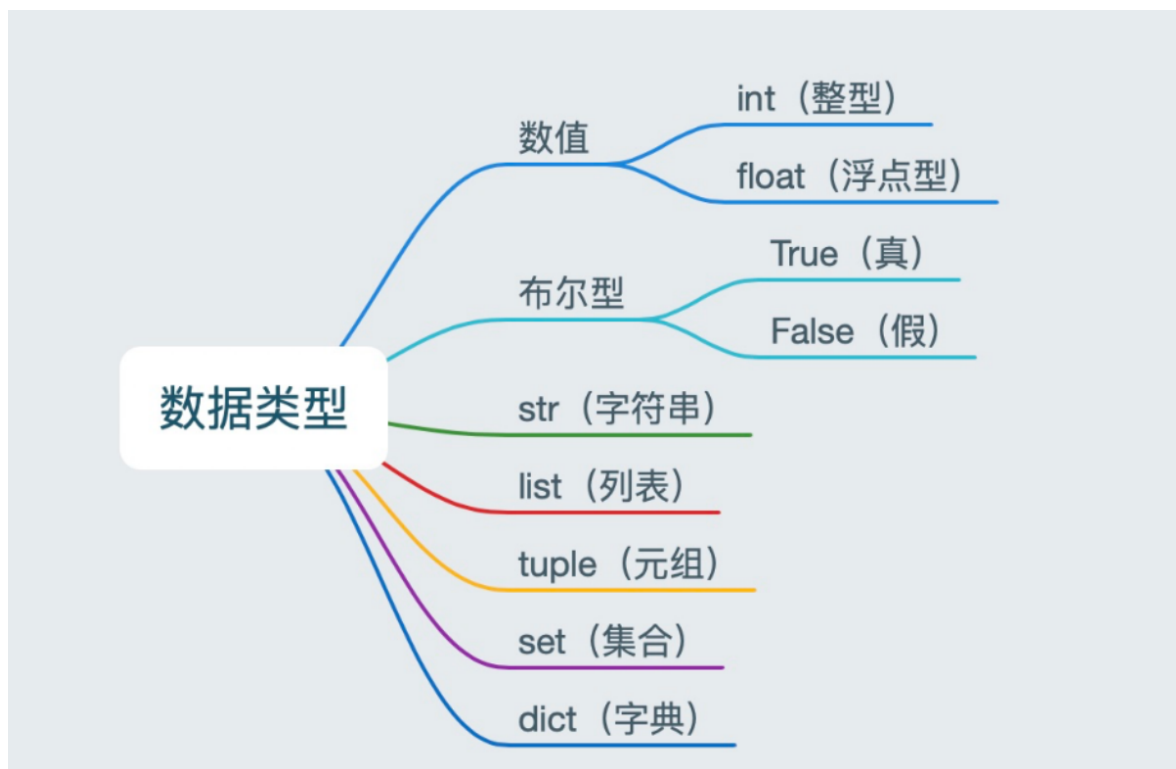


##### 3.2.1 Debug输出面板分类

- Debugger
  - 显示变量和变量的细节
- Console
  - 输出内容

### 四. 认识数据类型

在 Python 里为了应对不同的业务需求，也把数据分为不同的类型。



检测数据类型的方法: `type()`

```
a = 1
print(type(a)) # <class 'int'> -- 整型

b = 1.1
print(type(b)) # <class 'float'> -- 浮点型

c = True
print(type(c)) # <class 'bool'> -- 布尔型

d = '12345'
print(type(d)) # <class 'str'> -- 字符串

e = [10, 20, 30]
print(type(e)) # <class 'list'> -- 列表

f = (10, 20, 30)
print(type(f)) # <class 'tuple'> -- 元组

h = {10, 20, 30}
print(type(h)) # <class 'set'> -- 集合

g = {'name': 'TOM', 'age': 20}
print(type(g)) # <class 'dict'> -- 字典
```

## 总结

- 定义变量的语法

变量名 = 值

- 标识符
  - ☐ 由数字、字母、下划线组成

- ☐ 不能数字开头
- ☐ 不能使用内置关键字
- ☐ 严格区分大小写
- ☐ 数据类型

- 整型: `int`
- 浮点型: `float`
- 字符串: `str`
- 布尔型: `bool`
- 元组: `tuple`
- 集合: `set`
- 字典: `dict`

## 学习目标

---

- 格式化输出
  - 格式化符号
  - f-字符串
- `print`的结束符

## 6.输出

作用：程序输出内容给用户

```
print('hello Python')

age = 18
print(age)

# 需求：输出“今年我的年龄是18岁”
```

### 6.1格式化输出

所谓的格式化输出即按照一定的格式输出内容

#### 1.1格式化符号

| 格式符号   | 转换           |
|--------|--------------|
| ==%s== | 字符串          |
| ==%d== | 有符号的十进制整数    |
| ==%f== | 浮点数          |
| %c     | 字符           |
| %u     | 无符号十进制整数     |
| %o     | 八进制整数        |
| %x     | 十六进制整数（小写ox） |
| %X     | 十六进制整数（大写OX） |
| %e     | 科学计数法（小写'e'） |
| %E     | 科学计数法（大写'E'） |
| %g     | %f和%e的简写     |
| %G     | %f和%E的简写     |

### 技巧

- %06d，表示输出的整数显示位数，不足以0补全，超出当前位数则原样输出
- %.2f，表示小数点后显示的小数位数。

## 1.2 体验

格式化字符串除了%s，还可以写为 f'{表达式}'

```
age = 18
name = 'TOM'
weight = 75.5
student_id = 1

# 我的名字是TOM
print('我的名字是%s' % name)

# 我的学号是0001
print('我的学号是%4d' % student_id)

# 我的体重是75.50公斤
print('我的体重是%.2f公斤' % weight)

# 我的名字是TOM，今年18岁了
print('我的名字是%s，今年%d岁了' % (name, age))

# 我的名字是TOM，明年19岁了
print('我的名字是%s，明年%d岁了' % (name, age + 1))

# 我的名字是TOM，明年19岁了
print(f'我的名字是{name}，明年{age + 1}岁了')
```

f-格式化字符串是Python3.6中新增的格式化方法，该方法更简单易读。

### 1.3转义字符

- `\n`：换行。
- `\t`：制表符，一个tab键（4个空格）的距离

### 1.4结束符

想一想，为什么两个print会换行输出？

```
print('输出的内容', end="\n")
```

在Python中，`print()`，默认自带 `end="\n"` 这个换行结束符，所以导致每两个 `print` 直接会换行展示，用户可以按需求更改结束符。

### 1.5输出总结

- 格式化符号
  - `%s`：格式化输出字符串
  - `%d`：格式化输出整数
  - `%f`：格式化输出浮点数
- f-字符串
  - `f'{表达式}'`
- 转义字符
  - `\n`：换行
  - `\t`：制表符
- print结束符

```
print('内容', end="")
```