nsd1912-py01-day01

python官方手册：https://docs.python.org/zh-cn/3/library/index.html

python语法结构

- python靠缩进表达代码逻辑，推荐缩进4个空格
- 如果某一代码块只有一行语句，可以直接写在条件冒号后面，但是不推荐。
- 注释和续行与shell完全一样。在pycharm中，注释可以选中多行后，按ctrl + /
- 同一行可以输入多条语句，语句之间用分号分隔。不推荐。

函数调用

- 在python中函数调用，使用的是()

```
# 比如下面是一个函数的定义
>>> def func1(a=None, b=None, c=None):
...   pass
# 调用函数
>>> func1()  # 不传参，函数的参数使用默认值
>>> func1(10, 20, 30)  # 传参将会按顺序为参数赋值
>>> func1(10, c=100)   # 10赋值给a，b使用默认值，c的值为100
>>> func1(x=100)   # Error，函数没有名为x的参数
```

print函数

```
# 打印一个字符串，字符串使用单双引号没有区别
print('Hello World')
# 打印一个字符串和一个数字
print('Hao', 123)
# 打印3项数据，它们之间使用***进行分隔
print('hao', 123, 'abc', sep='***')
```

input函数

- input函数用于获取用户的键盘输入

- 注意：input获取的数据一定是字符类型

```
# 屏幕提示语为username: ，用户输入内容保存到变量 user中
user = input("username: ")
print(user)

>>> n = input('number: ')
number: 10
>>> n + 5   # 报错，字符和数字不能直接运算
>>> type(n)  # n的数据类型是字符
<class 'str'>
>>> int(n) + 5  # int将字符串转成整数
15
>>> n + str(5)  # str将数字5转成字符串'5'
'105'
```

变量

- 变量：能变化的量。
- 字面量：字面本身含义的量。如'hello'字符串永远是这5个字符，数字100永远是100。
- 为什么要用变量：
  - 方便
  - 变量有意义
- 在写程序时，尽量多用变量，而不是直接使用字面量。直接使用字面量，被称作“硬编码”。
- 变量只是使用起来方便，但是它最终仍然是代表某一项数据。大家就应该训练自己拥有一双可以“透视”的眼睛，当你看到变量时，应该想到这个变量的值是什么。
- 变量定义的约定：
  - 首字符只能是字母或下划线
  - 其他字符可以是字母、数字、下划线
  - 区分大小写
- 变量赋值时，使用=，=两边可以有空格，也可以没有。推荐作用是有空格。
- 推荐的命名方法：
  - 变量全部采用小写字母。如pythonstring
  - 简短、有意义。pystr
  - 多个单词间用下划线分隔。如py_str
  - 变量名采用名词，函数名采用谓词（动词＋名词）。如变量起名为phone，函数起名为update_phone
  - 类名采用驼峰的形式。如MyClass

变量赋值

- 变量在使用之前，必须赋值初始化。
- 变量的类型，由它的值决定。
- 变量赋值是一个自右向左的运算。将=右边的表达式计算出结果，再赋值给左侧的变量。

```
>>> a = 5 + 5
>>> print(a)
10
# 在python的交互解释器中，没有 print，直接写变量，也可以把它的值显示在屏幕上
>>> a
10
```

- 变量也可以支持自增、自减等运算

```
>>> a = a + 1
>>> a
11
>>> a += 1  # a = a + 1的简化写法
>>> a
12
>>> a *= 2  # 举一反三
>>> a
24
>>> a -= 10
>>> a
14
```

- 在python中，不支持a++或a--这种写法。因为理解起来比较绕

```
>>> a = 10
>>> ++a  # 正正为正
10
>>> a
10
>>> --a  # 负负为正
10
>>> a
10
```

- python是工程，不是艺术。一定有某些方法是最好的方法。

```
# 导入python之禅
>>> import this
The Zen of Python, by Tim Peters

Beautiful is better than ugly.     # 美胜丑
Explicit is better than implicit.  # 明胜暗
Simple is better than complex.     # 简胜繁
... ...
```

运算符

- 算术运算符

```
>>> 5 / 3   # 真正的除法
1.6666666666666667
>>> 5 // 3  # 保留商
1
>>> 5 % 3   # 模运算，只保留余数
2
>>> 2 ** 3  # 幂运算，2的3次方
8
>>> divmod(5, 3)
(1, 2)
>>> a, b = divmod(5, 3)  # 商和余数分别赋值给 a和b
>>> a
1
>>> b
2
```

- 比较运算符：返回结果是True或False

```
>>> 5 > 3
True
>>> 10 < 8
False
>>> 10 == 10
True
>>> 10 = 10   # 赋值语句，语法错误
  File "<stdin>", line 1
SyntaxError: can't assign to literal
>>> 10 != 8
True
# python支持连续比较
>>> 20 > 15 > 10
True
>>> 20 > 10 < 15
True
# 上面的比较等价于
>>> 20 > 10 and 10 < 15
True
```

- 逻辑运算符。计算的结果为True或False

```
# and两边的表达式结果全为真，最终为真。有一边为假，最终是假
>>> 10 > 5 and 10 < 20
True
# or两边的表达式结果全为假，最终为假。有一边为真，最终是真
>>> 10 > 50 or 10 < 100
True
# not是单目运算符，它将假变真，真变假
>>> not 10 > 5
False
>>> not 10 > 50
True
# 写代码时，适当的加括号有助于提升可读性
>>> not True or False
False
# 写为以下形式，结果不变，但是可读性会更好
>>> (not True) or False
False
```

python数据类型

数字

- 数字
  - 没有小数的整数
  - 有小数的浮点数
  - 布尔值True的值是1，False的值是0

```
>>> True + 5
6
>>> False * 5
0
```

- 整数数字的表示方法
  - 没有任何前缀的数字是10进制数
  - 2进制以0b或0B开头
  - 8进制以0o或0O开头
  - 16进制以0x或0X开头

```
>>> 0x11
17
>>> 0o11
9
>>> 0b11
3
# 进制转换。2小时3分5秒等于多少秒？
# 2 * 60 * 60 + 3 * 60 + 5
>>> 0o235  # 2 * 8 * 8 + 3 * 8 + 5
157
>>> 0x235  # 2 * 16 * 16 + 3 * 16 + 5
565
>>> 0o82   # Error
```

字符串

- 在引号内的字符集合
- 单双号和双引号完全一样
- python支持3引号（三个连续的单引号或双引号），它可以将用户的输入样式保存下来。

```
>>> users = '''张三
... 李四
... 王从
... 赵小六
... '''
>>> print(users)
张三
李四
王从
赵小六
>>> users
'张三\n李四\n王从\n赵小六\n'
>>> names = 'tom\njerry\nbob\nalice'
>>> print(names)
tom
jerry
bob
alice
```

- 字符串常用操作

```
>>> s1 = 'python'
>>> len(s1)  # 求长度
6
>>> s1[0]    # 下标从0开始
'p'
>>> s1[6]    # 报错，下标超出范围
>>> s1[-1]   # 下标为负，表示从右向左
'n'
>>> s1[-6]
'p'
>>> 'python'[2]
't'

>>> s1[2:4]   # 切片操作，起始下标包含，结束下标不包含
'th'
>>> s1[2:6]   # 切片时，下标越界不报错
'thon'
>>> s1[2:6000]
'thon'
>>> s1[2:]    # 结束下标不写，表示取到结尾
'thon'
>>> s1[:2]    # 起始下标不写，表示从开头取
'py'
>>> s1[:]     # 从开头取到结尾
'python'

>>> s1[::2]   # 第2个冒号后面的值是步长值，默认是 1
'pto'
>>> s1[1::2]
'yhn'
>>> s1[::-1]  # 步长为负，表示从右向左取
'nohtyp'

# 字符串拼接使用+，重复使用*
>>> s1 + ' is cool'
'python is cool'
>>> '#' * 50
'##################################################'
>>> s1 * 3
'pythonpythonpython'

>>> 't' in s1       # t在s1中吗？
True
>>> 'th' in s1      # th在s1中吗？
True
>>> 'to' in s1      # to在s1中吗？
False
>>> 'to' not in s1  # to不在s1中吗？
True
```

列表和元组

- 列表和元组类似于其他语言的数组
- 列表可变，元组相当于是不可变的列表

- 列表和元组所支持的操作方法，基本上与字符串一样

```
# 列表使用[]表示
>>> l1 = ['tom', 'jerry', 100, 200]
>>> l1[-1] = 1000   # 将列表最后一项重新赋值
>>> l1
['tom', 'jerry', 100, 1000]
>>> l1.append(50)   # 向列表尾部追加一项
>>> l1
['tom', 'jerry', 100, 1000, 50]
>>> l1 + [100]
['tom', 'jerry', 100, 1000, 50, 100]


# 元组使用()表示
>>> t1 = ('tom', 'jerry', 100, 1000, 50)
>>> t1[0] = 'zhangsan'  # 报错，元组不可变
```

字典

- 字典用{}表示
- 字典是key / value对形式的数据类型
- 它没有顺序
- 通过key找到value

```
>>> d1 = {'name': 'bob', 'age': 20}
>>> len(d1)   # 求长度
2
>>> 'bob' in d1   # bob是字典的key吗？
False
>>> 'name' in d1
True
>>> d1['name']    # 通过key取value
'bob'
```