Python入门

Python概述

- 弱类型编程语言
- 依赖于缩进，区分代码块的缩进至少一个空格，同一代码块内缩进数一致
- 注释用 `#`，无多行注释，但会忽略未用于赋值的字符串，可以用三个引号包围作为注释
- 单引号与双引号无区别，但字符串内需要使用引号时，需要使用另一种引号进行包围
- 创建变量直接 `var = content` 即可
- 在 **Shell 和 Jupyter** 中运行Python，一行的运算值或常量若未被赋值，也会被打印出来，即使未加 `print`
- Python 内的赋值均是引用赋值，除非建立临时变量再赋值，例如
- Python 内的 immutable 不可变数据类型 (int, float, str, tuple) 都是可哈希 (hashable) 的，例如 list, set, dict 则是不可哈希的，在**集合 set 中无法存储不可哈希的数据类型**

```
list1 = [1, 2, 3]
list2 = list1
list3 = list1.copy() # 可以用 list.copy() 或者 list[:]

list1.append(4)
print(list1) # [1, 2, 3, 4]
print(list2) # [1, 2, 3, 4]
print(list3) # [1, 2, 3]
```

- Python 的变量均是类的对象，并且具有一系列 **magic/special methods 魔法方法**，例如 `__len__()`，`__init__()`，用于实现类的功能，需要调用此类的功能，均有对应的函数或方法或运算符（类似**重载**），例如查找字符串的长度，可以直接用 `len(str)`，也可以用 `str.__len__()`，后者更多用在**自定义类**或**增加内置类的方法**上.
- 对于类和模块，都应该在开始时加入文档字符串做解释
- 类内方法之间空一行，模块内的类或函数之间空两行
- `import` 时先加入标准库模块，空一行后再加入第三方库模块，空一行后再加入自定义模块

数据类型

主要有数字类型（包含 Integer 整数和 Float 浮点数）、String 字符串、Boolean 布尔值、List 列表、Tuple 元组、Set 集合、Dictionary 字典

数字类型

有三种数字型：整数、浮点数、复数，其中复数的虚数单位为 `j`.

```
i = 2
d = 3.5
c = 1 + 2j

print(type(i)) # <class 'int'>
print(type(d)) # <class 'float'>
print(type(c)) # <class 'complex'>
```

String 字符串

Python 内存储类型有四种：

- `f"str"` 或 `F"str"`：formatted string 格式化字符串，可以在字符串中直接插入变量或表达式

  ```
  a = 10
  b = 20
  result = f"The sum of {a} and {b} is {a + b}."
  print(result) # The sum of 10 and 20 is 30.
  ```

- `r"str"` 或 `R"str"`：raw string 原始字符串，原封不动输出字符串而不会进行转义等行为

  ```
  normal_string = "I can't fly\nIt's a bird"
  raw_string = r"I can't fly\nIt's a bird"

  print(normal_string)
  """
  I can't fly
  It's a bird
  """

  print(raw_string)
  """
  I can't fly\nIt's a bird
  """
  ```

- `u"str"` 或 `U"str"`：unicode string，Python 3 中所有字符串默认都是 Unicode 字符串，可以省略前面的 `u`

- `b"str"` 或 `B"str"` : byte string, 用 1byte 存储字符串，用 ASCII 编码，不允许出现中文等非 ASCII 字符. 前三种在 Python 内均是 `str` 类型，此形式事实上是 `bytes` 类型. 由 `u"str"` 转为 `b"str"` 需要使用 `encode()` 函数，非 ASCII 编码字符会以**字节编码形式**存在，由 `b"str"` 转为 `u"str"` 需要使用 `decode()` 函数.

转义字符以下：

转义字符	含义
\n	换行
\t	制表符
\\	反斜杠
\'	单引号
\"	双引号
\a	响铃
\b	退格
\f	换页
\r	回车
\v	垂直制表符
\xhh	十六进制数，其中 hh 是两位十六进制数
\ooo	八进制数，其中 ooo 是三位八进制数
\N{name}	Unicode 字符名，其中 name 是 Unicode 字符名

```
print("12345\babc") # 5被退格/删除了 # 1234abc
print("12345\fabc") # 古老打印机上用是换纸，控制台无法显示 # 12345abc
print("12345\rabc") # 回车，光标回到首位，123被覆盖 # abc45
print("12345\vabc") # 垂直制表符，向下跳，控制台无法显示 # 12345?abc
```

Bool 布尔值

布尔值只有 `True` 和 `False` 两种

**注意要区分大小写， `true` 和 `false` 不是布尔值*

数组类型

Python 内有四种集合数据类型：

- List 列表：有序可更改，允许重复，用 `[value1, value2, ...]` 表示.
- Tuple 元组：有序不可更改，允许重复，用 `(value1, value2, ...)` 表示. 小括号也可以省略，直接用 `value1, value2, ...` 也是构成元组类型数据
- Set 集合：无序和无索引，不可重复，用 `{value1, value2, ...}` 表示.
- Dictionary 字典：无序，可变和有索引，不可重复，用 `{key1: value1, key2: value2, ...}` 表示.

有序数组（List, Tuple）的索引从 `0` 开始，Python 也提供了负数索引，从末尾开始， `-1` 表示最后一个元素， `-2` 表示倒数第二个元素，以此类推. String 类型也可通过索引查找，与 List 类型的索引一致. 无序数组中 Set 的索引无意义，因为其内部元素没有顺序，但可以用 `in` 关键字判断元素是否在集合中. Dictionary 有关键词 key 索引，可以通过 `key` 来获取对应的值.

```
# List
list = [1, 2, 3, 4, 5]
dict = {"key1": "value1", "key2": "value2"}
print(list[1]) # 2
print(list[-1]) # 5
print(dict["key1"]) # value1
```

有序数据集的索引可以有三部分： `[start: end: step = 1]` ，其中 `start` 表示起始索引， `end` 表示结束索引，**默认是整个字符串的范围**， `step` 表示步长（不能为 `0` ），步长为负数表示倒序.

只有一个参数时，表示索引，返回对应索引的值；有两个参数时，表示切片，顺序返回对应索引范围 `[start, end)` 内的所有值，范围顺序与 `start` 和 `end` 的值大小无关，是**索引对应元素的位置有关**；有三个参数时，表示步长切片，返回对应索引范围内的值.（所谓参数可以是 `None` 或就不写，但需要有：）（要注意**步长正负要与区间左右索引位置匹配**，否则返回空）

```
# List
list = [1, 2, 3, 4, 5]
print(list[1]) # 2
print(list[1:3]) # [2, 3]
print(list[1:4:2]) # [2, 4]
print(list[2:]) # [3, 4, 5]
print(list[::-1]) # [5, 4, 3, 2, 1]
print(list[1:4:-1]) # []
print(list[-4:-1]) # [2, 3, 4] # -4 是 2 的索引，-1 是 5 的索引，是由对应元素的实际位置决定的，而非
print(list[-1:-4]) # [] # 此处虽 -1 < -4 但对应元素位置是倒着的，所以输出空
```

字典的 key 和 value 可以任意，例如 {1: "one", "two": 2, "three": "0b11", 4: 100}

Range 类型

Python 内置了一种类型 range，用于生成一个**不可变**的数字序列，并非是**列表数据类型**，可以用于 for 循环中或将其转化为列表等数组类型.

```
class range(stop: SupportsIndex): ...

class range(
    start: SupportsIndex,
    stop: SupportsIndex,
    step: SupportsIndex = ...
): ...

# range(stop) == range(0, stop, 1)
# 在 [start, stop) 范围内生成数字序列，步长为 step
```

Type Casting 类型转换

- **数字类型**（可以**转换非复数数字、布尔、字符串类型**）
  - int()：
    对 **Float 小数处理是截断**，例如 int(-3.5) = -3，int(3.999) = 3
    对**只显性包含数字 (可以由空白包围) 的 String**，可以转换，例如 int(" 123 \t") = 123，无法实现 int("0b11") = 3 . **字符串**转化为数字可以用 base 参数**指定进制** (范围是 0，2-36)，例如 int("11", base=2) = 3 . 当 base=0 时表示**进制在字符串内**表示，如 int('0b10', base=0) = 2
    对**Bool 布尔值**，True 转化为 1，False 转化为 0
  - float()：
    同上，但可以转化小数，且无进制
  - complex()：
    复数无法转化为其它数字类型!!
    可以由两个**数字类型作参数**转化，如 complex(1+2j, 1) = 1+3j，计算原理是 real + imag*j
    由 **String** 作参数时，只能有一个参数，如 complex("1+2j") = 1+2j
    对布尔值同上数字类型的转化
- **布尔类型**（**所有对象**均能转换）
  - bool()：
    只有 bool(False)，bool(None)，bool(0)，bool("")，bool(())，bool([])，bool({})，bool(range(0)) 等空数据为 False，其余都为 True
- **字符串类型**（**所有对象**均能转换）

- `str()`：
  直接将其它数据类型存储后的结果转化为字符串，例如 `str([1, 2, '1']) = "[1, 2, '1']"`，`str(True) = "True"`，`str(range(30)) = "range(0, 30)"`．也可以用 `repr()` 函数，效果一样
  对于**字节字符串**的转换，可以有第二个参数和第三个参数指定编码格式 `encoding` 和错误 `errors`，如 `str(b'123', "utf-8") = "123"`．也可以直接用 `bytes.decode(encoding="utf-8", errors="strict")` 方法转化
- `bytes()`：
  事实上这并非数据转化，对于数值或数值列表，得到的是**字节编码**，**整数**用于创建**多个空字节**，例如 `bytes(3) = b'\x00\x00\x00'`，数值数组用于创建**指定的ASCII编码**，例如 `bytes([1, 2, 65, 97]) = b'\x01\x02Aa'`．
  对于**字符串 String**，则是将字符串转化为字节编码，如 `bytes('Python中', encoding='utf-8') = b'Python\xe4\xb8\xad'`．也可以直接用 `str.encode(encoding="utf-8", errors="strict")` 方法转化

- **数组类型**（能转换**可迭代对象**，如**字符串、数组、range类型**）
  - `list()`
  - `tuple()`
  - `set()`
  - `dict()`：
    前三种数组类型均能相互转换，且能转化 Dictionary，但所得到的数组是 `dict.keys()`
    若要将前数组类型转化为字典，需要用包含 `key-value` 的数组，例如 `dict([["a", 1], ["b", 2]]) = {"a": 1, "b": 2}`

基本语法

if elif else

```
if condition:
  # do something
elif condition: # 也可以用 else if
  # do something
else:
  # do something
```

for

```
for item in iterable:
  # do something
```

值得注意的是 item 是通过传值的方式得到的，所以若 item 是 immutable 类型 (如 `int`，`float` 等)，则 item 的改变只在循环内部生效，对于 iterable 中的值是不影响的

```
t = (1, 2, 3)
l = [1, 2, 3]
for item in t:
  item = item + 1
  print(item) # 2, 3, 4
print(t) # (1, 2, 3)
for item in l:
  item = item + 1
print(l) # [1, 2, 3]

# in different case
tl = ([1, -1], [2, -2])
for item in tl:
  item[0] = item[0] + item[1]
  print(item) # [0, -1], [0, -2]
print(tl) # ([0, -1], [0, -2])
```

while

```
while condition:
  # do something
```

break, continue, pass

`break` 和 `continue` 用于控制循环， `break` 用于跳出循环， `continue` 用于跳过当前循环，继续下一次循环.

`pass` 表示什么都不做，如同 C++ 的 `return null` 或 `;`. 由于 Python 的块语言必须要有内容，所以在占位或无内容时可以通过 `pass` 可以实现一个空的块.

try except else finally

```
try:
  do something
except:
  do something_else when an error occurs
else:
  do something_when_no_error
finally:
  do something_no_matter_what
```

```
try:
  userInput1 = int(input("Please enter a number: "))
  userInput2 = int(input("Please enter another number: "))
  answer =userInput1/userInput2
  print ("The answer is ", answer)
  myFile = open("missing.txt", 'r')
except ValueError:
  print ("Error: You did not enter a number")
except ZeroDivisionError:
  print ("Error: Cannot divide by zero")
except Exception as e:
  print ("Unknown error: ", e)
```

具体的错误类型可以查看 Python 官方文档 Exceptions

with

```
with open('file.txt') as f:
  # do something
```

`with` 语句用于管理资源，例如文件，在 `with` 语句块内打开文件，在 `with` 语句块结束后会自动关闭文件，无需手动调用 `f.close()`.

function 函数

函数同 C++，只不过用 `def` 申明.

函数参数

1. **位置实参**

函数调用时需要按照函数定义的参数顺序传入实参. 同 C++

2. **关键字实参**
   函数调用时可以按照参数名传入实参，可以不按照顺序，也可以不传入所有参数，但必须保证传入的参数名在函数定义中存在.
   若要同时使用位置实参和关键字实参，则**位置实参必须在关键字实参之前**.

```
def fun(x, y):
  return x + y * 2

print(fun(1, 2)) # 5
print(fun(y=2, x=2)) # 6
print(fun(x=2, 2)) # SyntaxError: positional argument follows keyword argument
print(fun(2, x=2)) # fun() got multiple values for argument 'x'
print(fun(2, y=2)) # 6
```

3. **默认值**
   同 C++，先定义无默认值的参数，再定义有默认值的参数
4. **形参的赋值**
   在 Python 内，普通数据类型通过函数调用时，是**值传递**，即函数内部对参数的修改**不会影响**函数外部的变量. 但对于**列表、字典、集合**这类可变 (不可哈希) 类型，函数内部对参数的修改会影响到函数外部的变量.

```
def fun1(x):
  x = x + 1
def fun2(x):
  x.append(1)
a = 1
b = [1]
fun1(a)
fun2(b)
print(a) # 1
print(b) # [1, 1]
```

   若想输入的参数在函数内部修改后不影响函数外部的变量，可以使用副本拷贝，即使用 `list()` 、 `dict()` 、 `set()` 或 `copy()` 等方法将参数转化为副本.
5. **任意数量实参**

用 `*args` 表示接受任意数量的实参 (arguments)，以**元组形式**存储，用 `**kwargs` 表示任意数量的**关键字实参** (keyword arguments)，以**字典形式**存储， `*` 和 `**` 来自编译原理内的闭包和解包.

将 `*` 作为函数的第一个位置形参时，这将代表之后的位置实参**只能通过关键字实参**的方式输入

module 模块

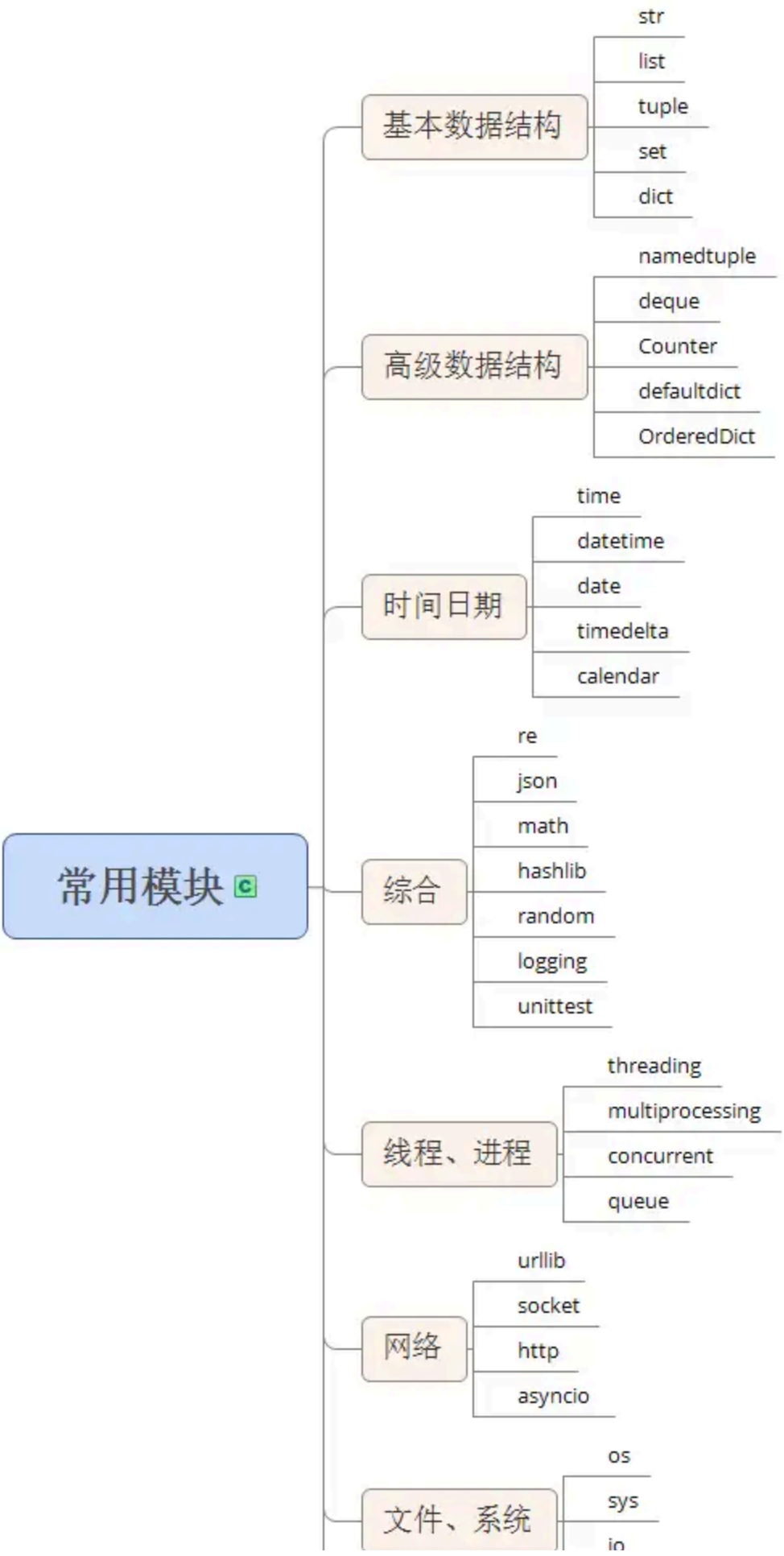
即**封装函数和类的包**，通过 `import` 调用 (本质是复制进主程序文件)

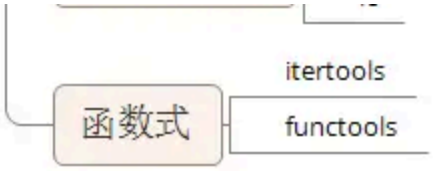
```
import module_name # 调用整个包
module_name.function_name() # 通过该方式调用模块内的函数，或是类和变量

from module_name import function_name # 调用模块内的某个函数，或是类和变量，之后可以直接使用该函数
function_name()

import module_name as mn # 给模块起别名
mn.function_name()

from module_name import function_name as fn # 给函数起别名
fn()

from module_name import * # 调用模块内的所有函数，类和变量，之后可以直接使用这些函数和变量
```

以上只适用于模块在**程序同目录**下或在**系统路径**内，若要调用不同文件夹下自设计的模块，需要将该模块路径加入系统路径

```
import sys
sys.path.append('path_to_module') # 此方法只是暂时加入，在程序运行结束后就会失效，若要永久加在系统路

import module_name # 此时可以调用不同文件夹下的模块
```

下面给出标准库的模块，第三方的模块一般具有较多功能，在此不具体叙述.


常用模块
基本数据结构
str
list
tuple
set
dict
高级数据结构
namedtuple
deque
Counter
defaultdict
OrderedDict
时间日期
time
datetime
date
timedelta
calendar
综合
re
json
math
hashlib
random
logging
unittest
线程、进程
threading
multiprocessing
concurrent
queue
网络
urllib
socket
http
asyncio
文件、系统
os
sys
io



Python_官方文档_标准库

其中较为常用的有时间模块、 `re` ， `json` , `random` ， `math` ， `cmath` (可以进行复数运算), `os` ， `sys`