

Day 1：Python 两大特性和四大基本语法

1. Python 语言两大特性

1.1 什么是动态语言？

1.2 什么是强类型语言？

2. 四大基本语法

2.1 命名规则

2.2 缩进原则

2.3 特殊关键字

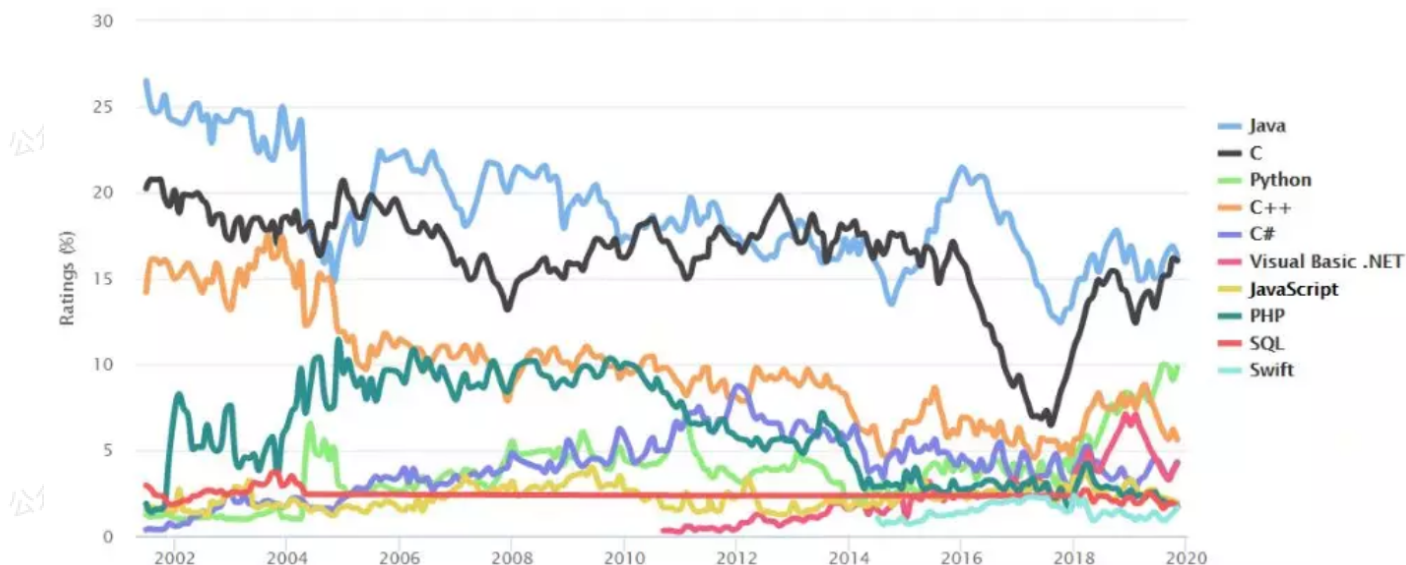
2.4 特殊运算符

小结

你好，我是悦创。

Python 语言使用率越来越高，使用 Python 不仅可以做 GUI 开发、Web 开发，还能进行数据预处理、数据探索性分析（EDA），更是进行数据挖掘、机器学习、深度学习等的首选语言。

基于 Python 的包更是枝繁叶茂，遍地开花，“Tiobe 编程语言排行榜”最新统计显示 Python 是增长最快的语言。



这得益于 Python 语言语法简单，开发效率高，集成系统非常方便。

Python 相关的就业机会也是非常多，待遇也很优厚。

因此，不管从易用性，还是工作机会和待遇来说，Python 都是 IT 从业者需要掌握的语言。

接下来，与大家，一起开始我们的 60 天 Python 探索之旅吧。

开始前，先了解下这个专栏的基本使用说明，主要包括如下几点：

- 使用的是 Python3
- 被讨论到的每个知识点都配备有小案例，辅助大家快速理解知识点，同时加深印象。
- 为了学习方便，对于小的代码块，尽量使用 `Ipython` 或 `jupyter notebook` 交互工具做演示。
- 对于文章涉及到的实战项目，相应的代码会按照软件工程和设计模式的思想，去拆分和组织。
- 书写的语言尽量做到通俗易懂，不搞华丽辞藻。

所有的这些考虑，都是为了让大家在短时间内掌握 Python 技术栈，多一个生存的本领。拿到理想的 offer 后，早日过上自己想要的生活。

让我们开始吧。

首先问大家一个问题，你知道 Python 是一门什么样的语言吗？

1. Python 语言两大特性

Python 是一门动态的、强类型语言。

1.1 什么是动态语言？

要了解什么是动态语言，要首先了解 **类型检查**。类型检查是验证类型约束的过程，编译器或解释器通常在编译阶段或运行阶段做类型检查。

类型检查就是查看 **变量** 和它们的 **类型**，然后判断表达式是否合理。例如，不能拿一个 **string 类型** 变量除以 **浮点数** 变量。否则就会出现报错。

```
1 In [1]: str1 = "6"
2
3 In [2]: float_num = 2.0
4
5 In [3]: str1 / float_num
6 -----
7 TypeError                                Traceback (most recent
   call last)
8 <ipython-input-3-54e30778909d> in <module>
9 ----> 1 str1 / float_num
10
11 TypeError: unsupported operand type(s) for /: 'str' and 'float'
```

如果类型检查发生在程序 **运行阶段** (run time)，那么它便是 **动态类型语言** (dynamically typed languages)。常见的动态语言包括：

- Python
- Javascript
- PHP

类型检查发生在 `编译阶段` (compile time)的是 `静态类型语言` (statically typed languages)。常见的静态类型语言包括：

- `C`
- `C++`
- `Java`
- `C#`
- `Scala`

1.2 什么是强类型语言？

强类型语言是指：不管是在编译阶段还是运行阶段，一旦某种类型绑定到变量后，此变量便会持有此类型，并且不能同其他类型在计算表达式时，混合使用。

例如，在交互式工具 `IPython` 中输入如下两行代码：

```
1 In [1]: a = 5
2 In [2]: a = a + 's'
```

程序会抛出 `TypeError` 异常：

```
1 unsupported operand type(s) for +: 'int' and 'str'
```

意思是不支持 `int` 变量和 `str` 变量相加。

常见的强类型语言有：

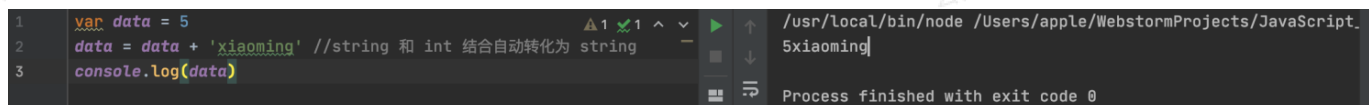
- `Python`
- `Java`
- `C#`
- `Scala`.

与之对应的是弱类型语言，弱类型语言容易与其他类型混合计算。弱类型语言代表

Javascript。

支持如下操作：

```
1 var data = 5
2 data = data + 'xiaoming' //string 和 int 结合自动转化为 string
```



```
1 var data = 5
2 data = data + 'xiaoming' //string 和 int 结合自动转化为 string
3 console.log(data)
```

/usr/local/bin/node /Users/apple/WebstormProjects/JavaScript_5xiaoming

Process finished with exit code 0

常见的弱类型语言有：

- C
- C++
- PHP
- Javascript.

如下，按照是否为静态/动态语言，弱类型/强类型两个维度，总结常用的语言分类。



2. 四大基本语法

分别从变量命名规则、缩进原则、特殊关键字和特殊运算符四个方面，总结 Python 的基本语法。

2.1 命名规则

Python 的变量命名规则主要包括两条：

- 允许包括英文、数字以及下划线(`_`)，不能以数字开头
- 名称区分大小写

特别说明以 `_` 开头的变量是有特殊意义的：

- 类变量若以单下划线(`_`)开头，代表不能被直接访问，类似于 C# 的受保护型变量(`protected`)，表示不能通过 `import module_name` 而导入。
- 类变量若以双下划(`__`)开头，表示为类的私有成员，不能被导入和其他类变量访问。
- 以双下划开头和双下划线结尾的变量是 Python 里的专用标识，有特殊的身份。

如 Python 自定义类中都包括 `__init__` 和 `__add__` 方法，如果不重写 `__add__` 去执行两个类加法操作，程序会抛 `TypeError` 异常。只有重写后，程序才能正常执行加法操作。

Python 变量命名习惯一般遵守蛇形命名法(`snake case`):

- 一般变量命名, `book_id`, `book_store_count`;
- 类名首字符为 大写, 如 Python 内置模块 `collections.abc` 中的 `Iterable` 类, 我们自定义的 `Book` 类等;
- 类方法名: `get_store_count()`;
- 其他特殊变量, 会全部大写, `M_PI`, `MAX_VEHICLE_SPEED`

这与 `Java` 命名方法不同, `Java` 最典型的命名方法: 驼峰命名法(`camel case`).

2.2 缩进原则

Python 最具特色的地方就是用缩进代替 `Java`, `C++` 中的 `{}`, 缩进的层级结构表示代码的逻辑层次。

比如, 自定义一个 `Book` 类, 重写 `__add__` 方法计算两类书的库存量和。

Python 的缩进方法, 一般为 4 个字符。

- 代码行 `class Book(object)` 与代码行 `# 定义类的参数` 的缩进, 此处为 4 个字符;
- 代码行 `def __add__(self,book):` 与 `return` 所在行缩进也是 4 个字符;

通过这种层级结构展现出代码的逻辑层次。

接下来我们先来了解一下: `__add__` 的使用样例。

```
1 # def __add__(self, *args, **kwargs): # real signature unknown
2 #     """ Return self+value. """
```

```

3 # pass
4
5 # def __str__(self, *args, **kwargs): # real signature unknown
6 #     """ Return str(self). """
7
8 # 示例一
9 class Vector(object):
10     def __init__(self, a, b):
11         self.a = a
12         self.b = b
13
14     def __str__(self):
15         # return 'Vector (%d, %d)' % (self.a, self.b)
16         return 'Vector ({}, {})'.format(self.a, self.b)
17
18     def __add__(self, other):
19         return Vector(self.a + other.a, self.b + other.b)
20
21
22 if __name__ == '__main__':
23     v1 = Vector(2, 10)
24     v2 = Vector(5, -2)
25     v3 = Vector(10, 10)
26     print(v1 + v2 + v3) # 输出: Vector (17, 18)
27     print(v1) # 输出: Vector (2, 10)

```

```

1 # -*- coding: utf-8 -*-
2 # @Time      : 2021/4/21 9:12 上午
3 # @Author    : AI悦创
4 # @FileName  : demo02.py
5 # @Software  : PyCharm
6 # @Blog      : http://www.aiyc.top
7 # @公众号    : AI悦创
8
9 # 示例二
10 class A(object):
11     def __add__(self, other):
12         # print("A __add__")

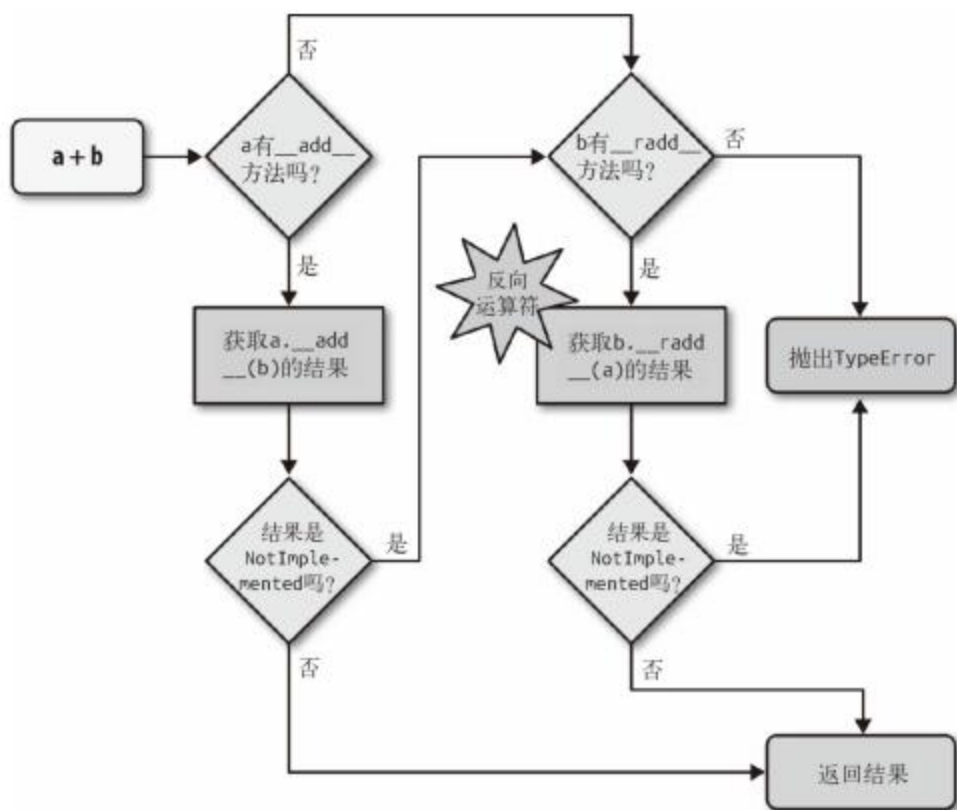
```



```

13         return "A __add__"
14
15     def __radd__(self, other):
16         # print("A __radd__")
17         return "A __radd__"
18
19
20 class B:
21     pass
22
23
24 if __name__ == '__main__':
25     a = A()
26     b = B()
27     print(a + b) # 当执行类的加法时候, 自动调用 add 方法 # 输出结果: A _
    __add__
28     print(b + a) # 输出: A __radd__
29     c = B()
30     print(b + c)
31     # 输出
32 # Traceback (most recent call last):
33 #   File "/Users/apple/Desktop/GitHub/PyCharm_Coder/Python 全栈 60
    天精通之路/Day1/class_add_parse/demo02.py", line 30, in <module>
34 #       print(b + c)
35 # TypeError: unsupported operand type(s) for +: 'B' and 'B'

```



下面代码，创建一个 Book 类：「示例一与二没啥区别」

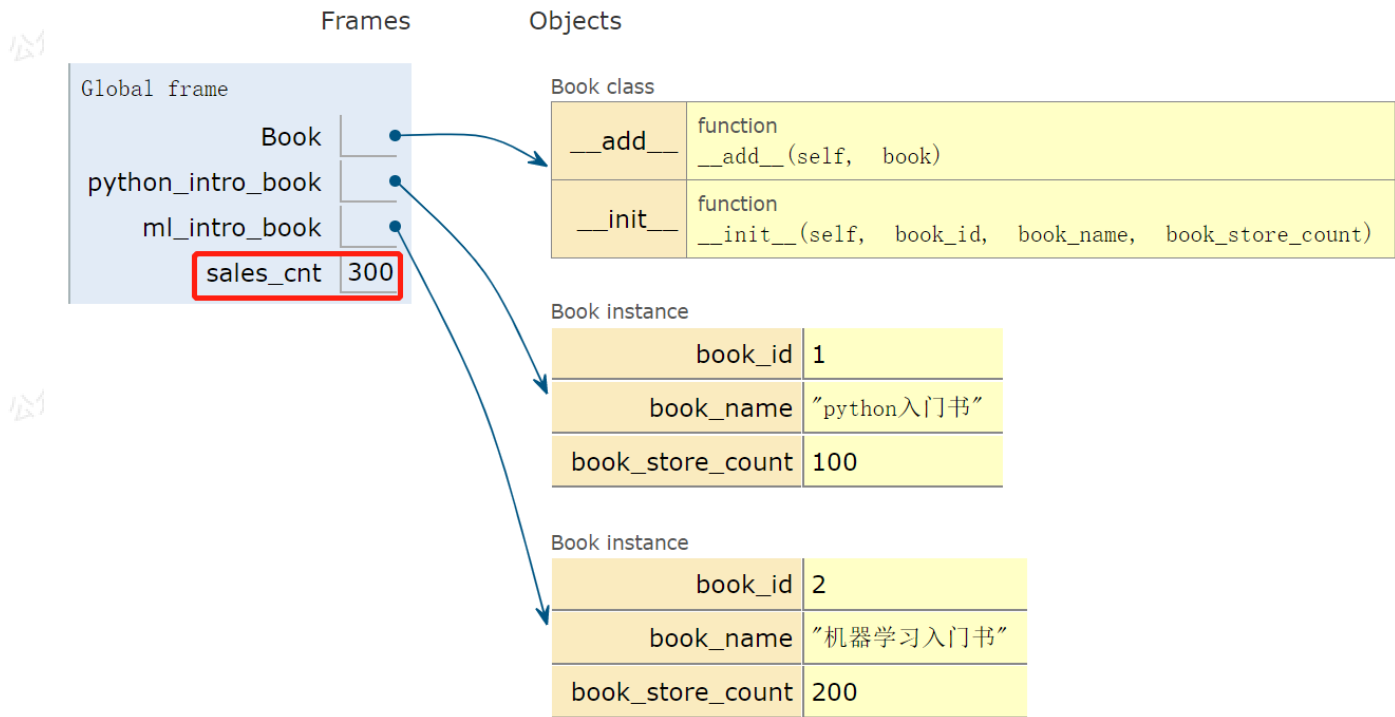
```

1 # 示例一
2 class Book(object):
3     # 定义类的参数
4     def __init__(self, book_id, book_name, book_store_count):
5         self.book_id = book_id
6         self.book_name = book_name
7         self.book_store_count = book_store_count
8
9     # 重写加法操作
10    def __add__(self, book):
11        return self.book_store_count + book.book_store_count
12
13
14 store_count = Book(1, 'python入门书', 100) + Book(2, '机器学习入门书', 200)
15 print(store_count) # 300
16

```

```
17
18 # 示例二
19 class Book(object):
20     # 定义类的参数
21     def __init__(self, book_id, book_name, book_store_count):
22         self.book_id = book_id
23         self.book_name = book_name
24         self.book_store_count = book_store_count
25
26     # 重写加法操作
27     def __add__(self, book):
28         return self.book_store_count + book.book_store_count
29
30
31 # 创建两个 Book 类的实例:
32 python_intro_book = Book(1, 'python入门书', 100)
33 ml_intro_book = Book(2, '机器学习入门书', 200)
34 # 求两本书的总销量
35 sales_cnt = python_intro_book + ml_intro_book
36 print(sales_cnt) # 300
```

如下是代码执行结果的演示图，打印总销量 300。结合图形，辅助大家快速理解代码。



为了帮助新手更容易理解代码整个执行过程，专栏会配备相应的演示动画：



Frames **Objects**

缩进格式，行间空行数，变量和等号空格等 Python [编码规范](#) 参考 [PEP8](#) 。

`autopep8` 包遵循 [PEP8](#) 的所有规范，安装此包，做好相关配置，便可自动实现 [PEP8](#) 制定的编码规范。

2.3 特殊关键字

Python 有 35 个关键字：

```
1 False      await      else        import      pass
2 None        break      except      in          raise
3 True        class      finally     is          return
4 and         continue  for         lambda      try
5 as          def        from        nonlocal    while
6 assert      del        global      not         with
7 async       elif       if          or          yield
```

自定义变量名不能与它们重复。

常用且不同于其他常用语言 c++ 和 Java 的关键字，如：

- True 和 False 用于表示值的真假，在 Java 中是 true 和 false；
- 逻辑反操作 Python 使用 not，Java 是 `!`；
- None 表示空值，Java 使用 null；
- Python 两个条件同时满足使用 and，Java 是 `&&`；
- 两者满足其一，Python 使用 or，Java 使用 `||`；
- Python 使用 elif，Java 是 `else if`；

其他比较特殊的关键字，如：

- del 用于删除可迭代对象中某个元素；
- def 用于定义函数；
- 带 yield 用于定义生成器(generator)函数；
- global 和 nonlocal 一种应用是 Python 函数式编程的闭包场景；
- pass 一种应用是定义接口，也是 Python 语言特有的一个关键字。

这些关键字的用法，会在后续文章，更为详细的介绍。在此，先构建一个整体上的认识，即可。

2.4 特殊运算符

Python 的运算符包括：

1	+	-	*	**	/	//	%	@
2	<<	>>	&		^	~	:=	
3	<	>	<=	>=	==	!=		

大部分运算符应该被熟知，重点介绍 3 个比较特殊的：`//`，`**`，`:=`。

- `//` 用于两个数值相除且向下取整，与 Python 的 `math` 模块中 `floor` 功能相似：

```
1 In [1]: 5//2
2 Out[1]: 2
3 In [2]: 5//4.5
4 Out[2]: 1.0
```

- `**` 用于幂运算：

```
1 In [1]: 2**3
2 Out[1]: 8
```

- `:=` 是在 2019 年，Python3.8 版本里，刚刚才被支持的运算符，被形象的称为 `海象运算符`。

```
1 n = len(a)
2 if n > 10:
3     print(f"{n}大于10")
```

如果使用 `海象运算符`，写法上更为精简：

```
1 if (n := len(a)) > 10:
2     print(f"{n}大于10")
```

- Python 比较运算符还支持链式比较，应用起来更加方便，比如：

```
1 i = 3
2 print(1 < i < 3) # False
3 print(1 < i <= 3) # True
```

- 另外，运算符 `@` 用于装饰器功能，本专栏会深入解释它的本质，同时配备的几个相关案例，一定会帮助你学会使用装饰器。

小结

Python 学习第一天，首先认识 Python 两大特征：

- 动态语言：动态指代码运行时才被编译器一行一行翻译执行；
- 强类型：强类型指被绑定一个类型后便不能修改，不能与其他类型混用。

四大基本语法，总结了 Python 的命名规则，缩进原则，特殊关键字，特殊运算符，为后面的学习打下基础。