Python语言

Python发展

1989年圣诞节期间，为了打发无聊的时间，荷兰人Guido van Rossum（数学、计算机双硕士，2005年加入Google，2013年加入DropBox），决心开发一种新的解释性脚本语言。

1991年初发布了第一个公开发行版。由于他是英国BBC喜剧《Monty Python's Flying Circus》的忠实粉丝，因此为这门语言取名Python。

Python目前已经成为很多大学的编程课语言。甚至在国内一些考试已经引入了Python。科学计算方面、运维领域Python几乎已经成为最主要的编程语言，拥有非常方便快捷开发的库。

Python的哲学，可以使用`import this`查看Python之禅。

Python的版本

目前企业中使用的主要版本还是2.x和3.x。

2.x最后一个版本是2.7，很多企业为了兼容老项目依然在维护。Python2将在2019年底不在支持，官方还提供了一个倒计时网站https://pythonclock.org/。

3.x还在不断的扩充发展，当前主流版本是3.6。

2015年9月发布3.5，2016年12月发布3.6，2018年6月发布3.7，2019年10月发布3.8.0。

Python2和3的区别

- 语句函数化，例如print(1,2)打印出1 2，但是2.x中意思是print语句打印元组，3.x中意思是函数的2个参数
- 整除，例如1/2和1//2，3.x版本中/为自然除
- 3.x中raw_input重命名为input，不再使用raw_input

- round函数，在3.x中i.5的取整变为距离最近的偶数
- 3.x字符串统一使用Unicode
- 异常的捕获、抛出的语法改变

早几年，各主要国内外大公司都已经迁移到了Python3。很多重要的Python第三方库也陆续停止了对Python2的支持，所以，Python 3已经是必须学习的版本。2018年Python3的使用比例已经超过了85%。

在公司内，往往老项目维护维持2.x版本暂不升级，新项目使用3.x开发。

开发时，假如使用3.5.8，部署时应尽量保持一致，不要随意升级版本，更不要降低版本。

不要迷信版本，学会一个版本，好好学会一门语言，其他都不是问题。当然，也不要迷信语言。

在最合适的领域使用最合适的语言。

环境安装

官方网站下载不同平台。https://www.python.org/downloads/

Linux环境安装

如果是Ubuntu等桌面系统，都已经更新到了Python较新的版本。但多数生产环境使用的还是红帽系统。

CentOS7默认还是Python2.7，而开发环境如果是高版本Python就带来了问题。为了不破坏当前系统使用，甚至以后为了多个Python项目部署（这些项目使用不同Python解释器版本），建议使用多版本工具。

也可以考虑容器部署Python应用程序。

pyenv多版本

官网 https://github.com/pyenv/pyenv

快捷安装 https://github.com/pyenv/pyenv#the-automatic-installer

```
# yum install git curl

python编译依赖如下
# yum install gcc make patch gdbm-devel openssl-devel sqlite-devel readline-
devel zlib-devel bzip2-devel

创建普通用户
# useradd python
# su - python


在python用户下安装
$ curl -L https://github.com/pyenv/pyenv-installer/raw/master/bin/pyenv-
installer | bash

由于某些原因，浏览器可以访问，大家自行下载保存为shell脚本执行
https://raw.githubusercontent.com/pyenv/pyenv-installer/master/bin/pyenv-
installer
或者使用项目源码文件
```

```
https://github.com/pyenv/pyenv-installer/blob/master/bin/pyenv-installer

以后更新pyenv使用
$ pyenv update
```

```
安装完，按照提示处理，把下面的脚本放到当前用户的.bashrc文件末尾
# the following to ~/.bashrc:

export PATH="/home/python/.pyenv/bin:$PATH"
eval "$(pyenv init -)"
eval "$(pyenv virtualenv-init -)"
```

缓存文件

由于连接国外网站速度太慢，可以先下载好将要安装的Python版本的安装包，直接放置在~/.pyenv/cache目录下，再进行安装。此cache目录不存在，请自行创建。

安装python多版本

```
$ pyenv install -l
$ pyenv install 3.6.9 -vvv
$ pyenv install 3.7.7 -vvv

$ pyenv version   当前python版本，system表示当前操作系统使用的版本
$ pyenv versions 查看所有已经安装版本，*星号表示当前版本
```

pyenv是源代码编译安装Python解释器，所以一定要安装gcc等依赖。

创建虚拟环境

由于不同项目，或使用不同Python版本，或使用同版本Python但使用不同版本库，很难将它们部署在一起，所以，应该使用不同的虚拟环境隔离部署。

使用pyenv local 命令建立一个目录和一个Python版本或虚拟版本之间的关系，该目录的子孙目录默认也继承这个版本。

```
$ mkdir -p projects/test369
$ mkdir -p projects/test369
$ cd projects/test369/

[python@nodex test369]$ pyenv virtualenv 3.6.9 py369
[python@nodex test369]$ pyenv local py369
(py369) [python@nodex test369]$ python -V
Python 3.6.9
(py369) [python@nodex test369]$ pyenv version
py369 (set by /home/python/projects/test369/.python-version)
```

Windows环境安装

下载`Windows x86-64 executable installer`，按照提示安装即可。

勾选增加PATH路径，简单安装直接点击"Install Now"。

打开Windows命令行

```
$ python -V
Python 3.7.4

$ pip -V
pip 19.0.3 (python 3.7)
```

pip是Python包管理器，以后安装Python第三方包都需要它，它从3.x开始就集成在Python安装包里面了。

pip通用配置

windows配置文件：`~/pip/pip.ini`。windows家目录，在"运行"中键入`.`

Linux配置文件：`~/.pip/pip.conf`

内容，可参照 http://mirrors.aliyun.com的pypi帮助

```
[global]
index-url = https://mirrors.aliyun.com/pypi/simple/

[install]
trusted-host=mirrors.aliyun.com
```

`pip install pkgname` 命令，是安装python包的命令

安装ipython

ipython

是增强的交互式Python命令行工具

```
$ pip list

$ pip install ipython
$ ipython
```

Jupyter

是基于WEB的交互式笔记本，其中可以非常方便的使用Python。
安装Jupyter，也会依赖安装ipython的

```
$ pip install jupyter
$ jupyter notebook help
$ jupyter notebook --ip=0.0.0.0 --no-browser

$ ss -tanl
```

常用快捷键

- a之前插入代码块、b之后插入代码块
- L 增加行号
- 运行代码块 shift + enter，选择下面的代码块
- 运行当前代码块 ctrl + enter

导出包

项目中开发完毕，要部署前，可以先导出当前使用的各种包的版本。在目标机器上安装这些包。

```
(py369) [python@node web]$ pip freeze > requirement
(py369) [python@node web]$ mkdir -p ~/magedu/projects/pro1
(py369) [python@node web]$ cd ~/magedu/projects/pro1
[python@node pro1]$ pyenv install --list
[python@node pro1]$ pyenv virtualenv 3.6.9 m369
[python@node pro1]$ pyenv local m369
(m369) [python@node pro1]$ mv ../web/requirement ./
(m369) [python@node pro1]$ pip install -r requirement
```

Pycharm安装

官网下载Pycharm社区版，足够开发项目使用了。

按照软件安装向导提示安装即可。

查阅帮助

- 在线帮助，html
- 下载并打开官方文档，chm
    - 第一手好的资料应该是帮助文档

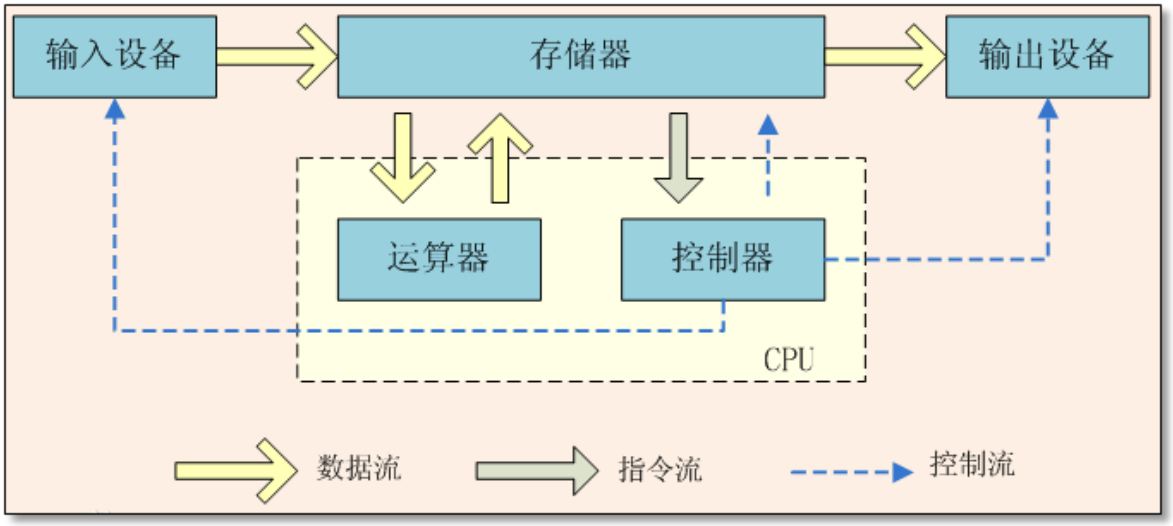
    - https://www.python.org/downloads/windows/
- IPython中
    - 使用help(keyword)，keyword可以是变量、对象、类、函数等
    - keyword?
    - keyword??

计算机基础知识

艾伦·麦席森·图灵（Alan Mathison Turing，1912年6月23日 - 1954年6月7日），英国数学家、逻辑学家，被称为计算机科学之父，人工智能之父。
图灵提出的著名的图灵机模型为现代计算机的逻辑工作方式奠定了基础。
图灵机已经有输入、输出和内部状态变化机制了。

冯·诺依曼著名匈牙利裔美籍犹太人数学家、计算机科学家、物理学家和化学家，数字计算机之父。他提出了以二进制作为数字计算机的数制基础，计算机应该按照程序顺序执行，计算机应该有五大部件组成。

冯诺依曼体系



五大核心部件

- 中央处理器CPU
  - 运算器：用于完成各种算术运算、逻辑运算和数据传送等数据加工处理。
  - 控制器：用于控制程序的执行，是计算机的大脑。运算器和控制器组成计算机的中央处理器（CPU）。控制器根据存放在存储器中的指令序列（程序）进行工作，并由一个程序计数器控制指令的执行。控制器具有判断能力，能根据计算结果选择不同的工作流程。
- 存储器：用于记忆程序和数据，例如：内存。程序和数据以二进制代码形式不加区别地存放在存储器中，存放位置由地址确定。内存是掉电易失的设备。
- 输入设备：用于将数据或程序输入到计算机中，例如：鼠标、键盘。
- 输出设备：将数据或程序的处理结果展示给用户，例如：显示器、打印机。

CPU中还有寄存器和多级缓存Cache。

- CPU并不直接从速度很慢的IO设备上直接读取数据，CPU可以从较慢的内存中读取数据到CPU的寄存器上运算
- CPU计算的结果也会写入到内存，而不是写入到IO设备上

计算机语言

语言是人与人沟通的表达方式。

计算机语言是人与计算机之间沟通交互的方式。

机器语言

- 一定位数的二进制的0和1组成的序列，也称为机器指令
- 机器指令的集合就是机器语言
- 与自然语言差异太大，难学、难懂、难写、难记、难查错

汇编语言

- 用一些助记符号替代机器指令，称为汇编语言。ADD A,B 指的是将寄存器A的数与寄存器B的数相加得到的数放到寄存器A中
- 汇编语言写好的程序需要汇编程序转换成机器指令

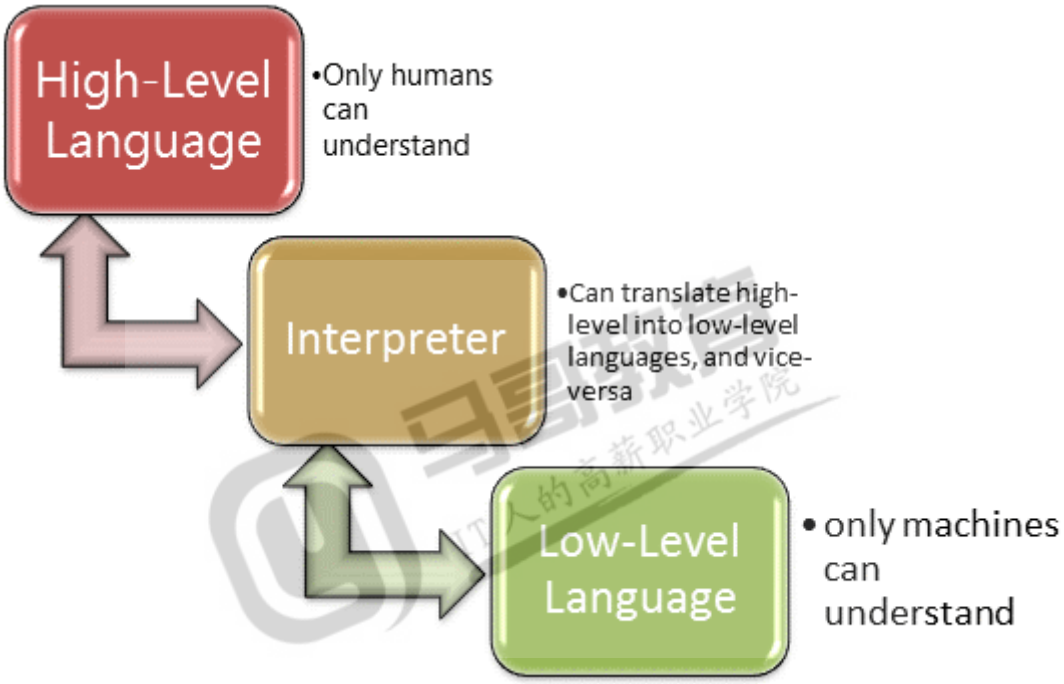
- 汇编语言只是稍微好记了些，可以认为就是机器指令对应的助记符。只是符号本身接近自然语言

低级语言

- 机器语言、汇编语言都是面向机器的语言，都是低级语言
- 不同机器是不能通用的，不同的机器需要不同的机器指令或者汇编程序

高级语言

- 接近自然语言和数学语言的计算机语言
- 高级语言首先要书写源程序，通过编译程序把源程序转换成机器指令的程序
- 1954年正式发布的Fortran语言是最早的高级语言，本意是公式翻译
- 人们只需要关心怎么书写源程序，针对不同机器的编译的事交给编译器关心处理



- 语言越高级，越接近人类的自然语言和数学语言
- 语言越低级，越能让机器理解
- 高级语言和低级语言之间需要一个转换的工具：编译器、解释器

编译语言

- 把源代码转换成目标机器的CPU指令
- C、C++等语言的源代码需要本地编译

解释语言

- 解释后转换成字节码，运行在虚拟机上，解释器执行中间代码
- Java、Python、C#的源代码需要被解释器编译成中间代码（Bytecode），在虚拟机上运行

高级语言的发展

- 非结构化语言

    - 编号或标签、GOTO，子程序可以有多个入口和出口
    - 有分支、循环
- 结构化语言
    - 任何基本结构只允许是唯一入口和唯一出口
    - 顺序、分支、循环，废弃GOTO
- 面向对象语言
    - 更加接近人类认知世界的方式，万事万物抽象成对象，对象间关系抽象成类和继承
    - 封装、继承、多态
- 函数式语言
    - 古老的编程范式，应用在数学计算、并行处理的场景。引入到了很多现代高级语言中
    - 函数是“一等公民”，高阶函数

程序Program

- `算法 + 数据结构 = 程序`
- 数据是一切程序的核心
- 数据结构是数据在计算机中的类型和组织方式
- 算法是处理数据的方式，算法有优劣之分

只有选对了合理的数据结构，并采用合适的操作该数据结构的算法，才能写出高性能的程序。

写程序的难点

- 理不清数据
- 搞不清处理方法
- 无法把数据设计转换成数据结构，无法把处理方法转换成算法
- 无法用设计范式来进行程序设计

Python基础

Python解释器

解释器	说明
CPython	官方，C语言开发，最广泛的Python解释器
IPython	一个交互式、功能增强的CPython
PyPy	Python语言写的Python解释器，JIT技术，动态编译Python代码
Jython	Python的源代码编译成Java的字节码，跑在JVM上
IronPython	与Jython类似，运行在.Net平台上的解释器，Python代码被编译成.Net的字节码
stackless	Python的增强版本解释器，不使用CPython的C的栈，采用微线程概念编程，并发编程

基础语法

注释

井号标注的文本

数字

- 整数int
  - Python3开始不再区分long、int，long被重命名为int，所以只有int类型了
  - 进制表示：
    - 十进制10
    - 十六进制0x10
    - 八进制0o10
    - 二进制0b10
  - bool类型，有2个值True、False
- 浮点数float
  - 1.2、3.1415、-0.12，1.46e9等价于科学计数法$1.46*10^9$
  - 本质上使用了C语言的double类型
- 复数complex
  - 1+2j或1+2J

字符串

- 使用 ' " 单双引号引用的字符的序列
- '''和""" 单双三引号，可以跨行、可以在其中自由的使用单双引号
- r前缀：在字符串前面加上r或者R前缀，表示该字符串不做特殊的处理
- f前缀：3.6版本开始，新增f前缀，格式化字符串

转义序列

- `\\` `\t` `\r` `\n` `\'` `\"`
- 上面每一个转义字符只代表**一个字符**，例如`\t`显示时占了4个字符位置，但是它是一个字符
- 前缀r，把里面的所有字符当普通字符对待，则转义字符就不转义了。

转义：让字符不再是它当前的意义，例如\t，t就不是当前意义字符t了，而是被\转成了tab键

缩进

- 未使用C等语言的花括号，而是采用缩进的方式表示层次关系
- 约定使用4个空格缩进

续行

- 在行尾使用 \，注意\之后除了紧跟着换行之外不能有其他字符
- 如果使用各种括号，认为括号内是一个整体，其内部跨行不用 \

标识符

标识符

1. 一个名字，用来指代一个值
2. 只能是字母、下划线和数字
3. 只能以字母或下划线开头
4. 不能是python的关键字，例如def、class就不能作为标识符
5. Python是大小写敏感的

标识符约定：

- 不允许使用中文，也不建议使用拼音
- 不要使用歧义单词，例如class_
- 在python中不要随便使用下划线开头的标识符

常量

- 一旦赋值就不能改变值的标识符
- python中**无法定义常量**

字面常量

- 一个单独的不可变量，例如 12、"abc" 、'2341356514.03e-9'

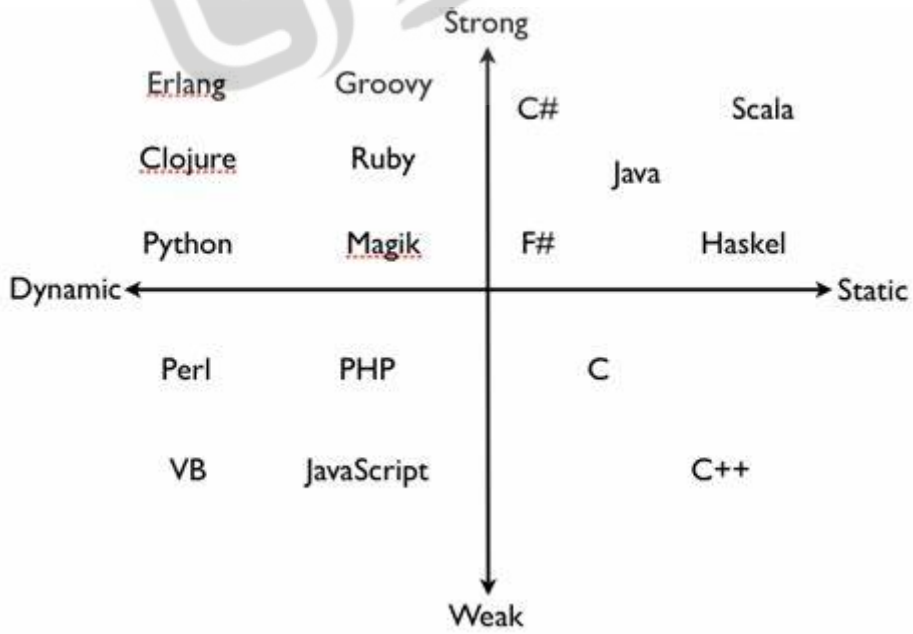
变量

- 赋值后，可以改变值的标识符

标识符本质

每一个标识符对应一个具有数据结构的值，但是这个值不方便直接访问，程序员就可以通过其对应的标识符来访问数据，标识符就是一个指代。一句话，标识符是给程序员编程使用的。

语言类型



Python是动态语言、强类型语言。

静态语言

- 事先声明变量类型，之后变量的值可以改变，但值的类型不能再改变
- 编译时检查

动态语言

- 不用事先声明类型，随时可以赋值为其他类型
- 编程时不知道是什么类型，很难推断

强类型语言

- 不同类型之间操作，必须先强制类型转换为同一类型。`print('a'+1)`

弱类型语言

- 不同类型间可以操作，自动隐式转换，JavaScript中`console.log(1+'a')`

但是要注意的是，强与弱只是一个相对概念，即使是强类型语言也支持隐式类型转换。

False等价

对象/常量	值
""	假
"string"	真
0	假
>=1	真
<=-1	真
()空元组	假
[]空列表	假
{}空字典	假
None	假

False等价布尔值，相当于bool(value)

- 空容器
  - 空集合set
  - 空字典dict
  - 空列表list
  - 空元组tuple
- 空字符串
- None
- 0

逻辑运算真值表

与逻辑

A	B	F
0	0	0
0	1	0
1	0	0
1	1	1

或逻辑

A	B	F
0	0	0
0	1	1
1	0	1
1	1	1

非逻辑

A	F
0	1
1	0

运算符Operator

算数运算符

+、-、*、/、//向下取整整除、%取模、**幂

注：在Python2中/和//都是整除。

位运算符

&位与、|位或、^异或、<<左移、>>右移

~按位取反，包括符号位

比较运算符

==、!=、>、>=、<、<=

链式比较： 4 > 3 > 2

比较运算符，返回一个bool值

思考：1 == '1' 吗？ 1 > '1' 吗?

逻辑运算符

与and、或or、非not

逻辑运算符也是短路运算符

- and 如果前面的表达式等价为False，后面就没有必要计算了，这个逻辑表达式最终一定等价为False
  1 and '2' and 0
  0 and 'abc' and 1
- or 如果前面的表达式等价为True，后面没有必要计算了，这个逻辑表达式最终一定等价为True
  1 or False or None
- 特别注意，返回值。**返回值不一定是bool型**
- 把最频繁使用的，做最少计算就可以知道结果的条件放到前面，如果它能短路，将大大减少计算量

赋值运算符

a = min(3, 5)
+=、 -= 、*=、/=、%=、//= 等
x = y = z = 10

成员运算符

in、not in

身份运算符

is 、is not

运算符优先级

运算符	描述
'expr'	字符串转换
{key:expr,...}	字典
[expr1,expr2...]	列表
(expr1,expr2,...)	元组
function(expr,...)	函数调用
x[index:index]	切片
x[index]	下标索引取值
x.attribute	属性引用
~x	按位取反
+x，-x	正，负
x**y	幂
x*y，x/y，x%y	乘，除，取模

运算符	描述
x+y，x-y	加，减
x<<y，x>>y	移位
x&y	按位与
x^y	按位异或
x|y	按位或
x<y，x<=y，x==y，x!=y，x>=y，x>y	比较
x is y，x is not y	等同测试
x in y，x not in y	成员判断
not x	逻辑否
x and y	逻辑与
x or y	逻辑或
lambda arg,...:expr	Lambda匿名函数

- 单目运算符 > 双目运算符
- 算数运算符 > 位运算符 > 比较运算符 > 逻辑运算符
    - -3 + 2 > 5 and 'a' > 'b'

搞不清楚就使用括号。长表达式，多用括号，易懂、易读。

表达式

由数字、符号、括号、变量等的组合。有算数表达式、逻辑表达式、赋值表达式、lambda表达式等等。

Python中，**赋值即定义**。Python是动态语言，只有赋值才会创建一个变量，并决定了变量的类型和值。

如果一个变量已经定义，赋值相当于重新定义。

内建函数

内建函数	函数签名	说明
print	print(value, ..., sep=' ', end='\n')	将多个数据输出到控制台，默认使用空格分隔、\n换行
input	input([prompt])	在控制台和用户交互，接收用户输入，并返回字符串
int	int(value)	将给定的值，转换成整数。int本质是类
str	str(value)	将给定的值，转换成字符串。str本质是类
type	type(value)	返回对象的类型。本质是元类
isinstance	isinstance(obj, class_or_tuple)	比较对象的类型，类型可以是obj的基类

```
print(1,2,3,sep='\n', end='***')

type(1) # 返回的是类型，不是字符串
type('abc') # 返回的是类型，不是字符串
type(int) # 返回type，意思是这个int类型由type构造出来
type(str) # 返回type，也是类型
type(type) # 也是type

print(isinstance(1, int))
print(isinstance(False, int)) # True
```


程序控制

- 顺序
  - 按照先后顺序一条条执行
  - 例如，先洗手，再吃饭，再洗碗
- 分支
  - 根据不同的情况判断，条件满足执行某条件下的语句
  - 例如，先洗手，如果饭没有做好，玩游戏；如果饭做好了，就吃饭；如果饭都没有做，叫外卖
- 循环
  - 条件满足就反复执行，不满足就不执行或不再执行
  - 例如，先洗手，看饭好了没有，没有好，一会来看一次是否好了，一会儿来看一次，直到饭好了，才可是吃饭。这里循环的条件是饭没有好，饭没有好，就循环的来看饭好了没有

单分支

```
if condition:
    代码块


if 1<2: # if True:
    print('1 less than 2') # 代码块
```

- condition必须是一个bool类型，这个地方有一个隐式转换bool(condition)，相当于False等价
- if 语句这行最后，会有一个冒号，冒号之后如果有多条语句的代码块，需要另起一行，并缩进
    - if、for、def、class等关键字后面都可以跟代码块
    - 这些关键后面，如果有一条语句，也可以跟在这一行后面。例如`if 1>2: pass`

多分支

```
if condition1:
    代码块1
elif condition2:
    代码块2
elif condition3:
    代码块3
......
else:
    代码块
```

```
a = 5
if a<0:
    print('negative')
elif a==0: # 相当于 a >= 0
    print('zero')
else: # 相当于 a > 0
    print('positive')
```

- 多分支结构，只要有一个分支被执行，其他分支都不会被执行
- **前一个条件被测试过，下一个条件相当于隐含着这个条件**

```
# 嵌套
a = 5
if a == 0:
    print('zero')
else:
    if a > 0:
        print('positive')
    else:
        print('negative')
```

while循环

while循环多用于死循环，或者不明确知道循环次数的场景

```
while cond:
    block

while True: # 死循环
    pass

a = 10
while a: # 条件满足则进入循环
    print(a)
    a -= 1
```

- 上例执行结果是什么?
- 为什么?
- 会不会打印出0?要注意边界的分析
- 如果a=-10可以吗?如何改?回忆一下，**False等价**是什么?

for语句

```
for element in iteratable: # 可迭代对象中有元素可以迭代，进入循环体
    block

for i in range(0, 10):
    print(i)
```

内建函数	函数签名	说明
range	range(stop) range(start, stop[, step])	返回**惰性**的对象 可以生成一个序列，遍历它就可以得到这个序列的一个个元素 **前包后不包**

```
# 计数器
for i in range(0):
    print(i)
print('---------')

for i in range(-2):
    print(i)
print('---------')

for i in range(1):
    print(i)
```

range(i)，i大于0，相当于计数器。

练习:

- 打印1到10

- 打印10以内的奇数
- 打印10以内的偶数
- 倒着打印10以内的奇数或偶数

```
# 打印偶数
for i in range(0, 10):
    if i % 2 == 0 :
        print(i)

for i in range(0, 10, 2):
    print(i)

# 打印奇数
for i in range(0, 10):
    if i % 2 != 0 :
        print(i)

for i in range(1, 10, 2):
    print(i)

# 倒着打印
for i in range(8, -1 , -2):
    print(i)
```

continue

跳过当前循环的当次循环，继续下一次循环

```
for i in range(0, 10):
    if i % 2 != 0 : continue
    print(i)

for i in range(10):
    if i % 2 != 0 :
        continue
    print(i)

for i in range(10):
    if i & 1: continue
    print(i)
```

break

结束当前循环

练习：计算1000以内的被7整除的前20个正整数（for循环）

```
# 计算1000以内的被7整除的前20个正整数

count = 0
for i in range(7, 1000, 7):
    print(i)
    count += 1
    if count >= 20:
        print(count)
        break
```

总结

- continue和break是循环的控制语句，只影响当前循环，包括while、for循环
- 如果循环嵌套， continue和break也只影响语句所在的那一层循环
- continue和break **只影响循环**，所以 if cond: break 不是跳出if，而是终止if外的break所在的循环
- 分支和循环结构可以**嵌套**使用，可以嵌套多层。

else子句

如果循环正常结束，else子句会被执行，即使是可迭代对象没有什么元素可迭代

```
for i in range(0): # 可迭代对象没有迭代
    pass
else:
    print('ok')

for i in range(0,10):
    break
else:
    print('ok')

for i in range(0,10):
    continue
else:
    print('ok')
```

有上例可知，一般情况下，循环正常执行，只要当前循环不是被break打断的，就可以执行else子句。哪怕是range(0)也可以执行else子句。

三元表达式

在Python中，也有类似C语言的三目运算符构成的表达式，但python中的三元表达式不支持复杂的语句

```
真值表达式 if 条件表达式 else 假值表达式
```

三元表达式比较适合简化非常简单的if-else语句。

```
# 判断用户的输入的值，如果为空，输出"empty"，否则输出该值

value = input('>>>')
if value:
    print(value)
else:
    print('empty')

value = input('>>>')
print(value if value else 'empty')
```

字符串拼接

```
str(1) + ',' + 'b' # 都转换成字符串拼接到一起
"{}-{}".format(1, 'a') # {}就是填的空，有2个，就使用2个值填充

# 在3.6后，可以使用插值
a = 100;
b = 'abc'
f'{a}-{b}' # 一定要使用f前缀，在大括号中使用变量名
```