**脚本语言介绍**

**——以Python和Perl为例**

|  |  |
| --- | --- |
| 作者 | 刘润瞻 |
| 版本 | 0.1 |
| 日期 | 2020年09月21日 |

目录

[1 编程语言概述 3](#_Toc51608741)

[2 脚本语言概述 3](#_Toc51608742)

[3 程序安装和运行 3](#_Toc51608743)

[3.1 Windows版本安装和运行 3](#_Toc51608744)

[3.2 Linux安装和运行 4](#_Toc51608745)

[3.3 Shebang 4](#_Toc51608746)

[3.4 语法检查 4](#_Toc51608747)

[3.5 代码风格 4](#_Toc51608748)

[3.6 第三方模块安装 5](#_Toc51608749)

[3.7 模块使用 5](#_Toc51608750)

[4 数据类型 5](#_Toc51608751)

[4.1 整型变量 5](#_Toc51608752)

[4.2 浮点型变量 6](#_Toc51608753)

[4.3 字符串变量 6](#_Toc51608754)

[5 数据结构 6](#_Toc51608755)

[5.1 列表 6](#_Toc51608756)

[5.2 字典/hash表 6](#_Toc51608757)

[6 常用语法 6](#_Toc51608758)

[6.1 函数定义 6](#_Toc51608759)

[7 文件和文件夹处理 7](#_Toc51608760)

[7.1 写文件 7](#_Toc51608761)

[7.2 读文件 7](#_Toc51608762)

[7.3 遍历文件和文件夹 8](#_Toc51608763)

[8 系统调用 8](#_Toc51608764)

[9 正则表达式 8](#_Toc51608765)

[10 Makefile编写 9](#_Toc51608766)

## 编程语言概述

常用的计算机编程语言一般分为两类，编译型和解释型语言。编译型语言一般由编译器对源代码进行编译，直接生成对应平台的二进制文件，如C、C++、Fortan等；解释型语言则并不直接生成二进制文件，通常是交由解释器进行边解释边运行，如Python、Perl、Shell、Makefile等。但这两种语言的界限并不明确，如python3.0以上的版本为提高程序运行速度，也会在运行前预先编译出一些中间代码，但本质上还是属于解释型语言。

## 脚本语言概述

脚本语言是一类用于封装、扩展的胶水语言，常见的脚本语言都是解释型语言，如Python等（Python封装了一个易于调用C和C++程序的接口，使得Python调用C/C++程序变得非常容易，这也是Python在人工智能领域大热的原因，人工智能领域常见的成熟库都是用C/C++编写的）。这些语言提供的数据类型和结构相同，语法和功能相近，一通百通。

由于Python2.7已于今年春季结束生命周期，不再维护；Python3与Python2之间的语法和实现层面差异较大，最好作为两个独立版本进行学习；而Perl自5.0以后的所有版本均能实现向下兼容，不同子版本间差异较小，学习成本低。故下面以Python3.0和Perl5.0为例，对脚本语言进行概述。

工作中，可使用集成开发环境IDE进行脚本的开发，如Pycharm、Spyder、Eclipse/MyEclipse、VS Code等。这些工具有着适合GUI界面的DEBUG/RUN模式，更适合基于Windows平台的脚本开发；而在Linux平台中，最好使用Makefile进行工程的创建、运行和维护，这也利于理解不同脚本语言的运行方式和配置参数，故在结尾增加对Makefile的编写/使用说明。

## 程序安装和运行

### Windows版本安装和运行

Python3.8版本的下载地址为：

<https://www.python.org/ftp/python/3.8.5/python-3.8.5-amd64.exe>

Perl5的下载地址为：

<http://strawberryperl.com/download/5.30.2.1/strawberry-perl-5.30.2.1-64bit.msi>

除此之外，可安装MSYS2——windows平台下类linux的shell命令行开发环境，可通过pacman安装最新版Python和Perl（此外，该软件还可安装开源波形查看软件gtkwave——名称mingw-w64-x86\_64-gtkwave和Verilog编译器iverilog/verilator——网址<https://github.com/steveicarus/iverilog />软件 mingw-w64-x86\_64-verilator，用于小模块的功能仿真，可参考网络教程）。按常规软件操作流程安装完成后，

1. 打卡msys2软件，执行pacman –Syyuu更新软件（如有确认，输入y）；
2. 更新完成后关闭软件并重新打开，执行pacman –S base-devel base perl python python-pip msys2-devel mingw-w64-x86\_64-toolchain mingw-w64-x86\_64-perl mingw-w64-x86\_64-python mingw-w64-x86\_64-python-pip（如有确认，输入y）；
3. 更新完成后，可根据windows的命令行cmd操作，在MSYS2中确认Perl和Python的成功安装

MSYS2软件的下载地址为：

<https://repo.msys2.org/distrib/x86_64/msys2-x86_64-20200720.exe>

推荐直接安装，如有添加环境变量的，可勾选。

安装成功的标志：

Win7打开cmd命令行，Win10打开Powershell，运行python –version和perl –version，给出软件的版本信息。

Python可以通过运行python程序进入交互式命令行界面，直接运行代码，如下所示：

\*\*\*\*\*python3.8\*\*\*\*\*

>>> a = 0

0

Python也可以通过编写脚本运行程序，但Perl一般只支持通过脚本运行程序。这里推荐使用脚本执行程序，便于保存和调试代码。Windows平台推荐使用VS Code作为脚本编辑器。

### Linux安装和运行

所有Linux的发行版均默认安装Python3和Perl，只是安装的版本不同。可自行安装不同版本，但初期不建议，无需浪费时间于此。可通过打开新的终端terminal运行Perl和Python程序。

### Shebang

Shebang是脚本语言的一个特色，放于程序首句，用于指定执行当前脚本的编译器和版本。Windows和Linux下均可使用，便于在环境变量配置无误时，直接运行脚本，无需再次指定解释器版本（经测试发现，Windows平台的文件后缀名必须为指定后缀时才可使用Shebang，如Python的“.py”、Perl的“.pl”；Linux平台的文件后缀名无特殊要求）。

Python的Shebang是：#!/usr/bin/python3

Perl的Shebang是：#!/usr/bin/perl

最好在脚本前统一加上Shebang，不是语法要求，是代码风格的要求。

### 语法检查

目前Python的解释器尚无支持预先的语法检查功能，只能边执行边调试。而Perl中可以在Shebang后添加如下两句，用于对语法和未声明变量的提示：

use strict;

use warnings;

这两句同样不是必须的，但最好加上，是代码风格的要求。如觉得麻烦，可指定解释器支持的Perl版本，自动添加这些语法检查功能：

use 5.30.2;

### 代码风格

无论使用何种编程语言，无论是否从事软件行业，一个好的代码风格，于别人、于自己而言，都是百利无一害的。

对于脚本语言而言，请务必遵守如下代码风格：

1）变量和常量的命名规范：

名称上区分变量和常量，如大小写等；

变量意义简洁无歧义，除循环变量外不得使用i,j,k等做变量名称；

常量增加常量名，而不直接使用常量值（称为“魔法数”）。

2）务必注意变量和常量的生命周期，禁止在子函数中直接使用外部变量/常量和全局变量/常量，除非这些变量/常量的值和生命周期绝对可控；

3）代码中不要用tab实现代码缩进的效果，尤其是Python脚本，用2/4个空格做替换；

4）一行一个语句，单个语句过长时，分行写；

5）Perl语句以“;”结尾；Python语句无需结尾标志，多个语句放在一行时，利用“;”区分不同语句（不推荐，最好一个语句独占一行）。

### 第三方模块安装

现代编程语言的一个特点就是模块化，即不同的程序按功能被划分为不同的模块。当需要某个功能时，仅需要导入相应的模块名便可使用，实现了代码复用的同时，保证了代码的简洁性。

Python/Perl等语言的流行，就在于这些语言除了标准模块外，还维护了大量成熟可靠的第三方模块，便于在日常工作中的程序开发和维护。日常工作中，可能涉及到文件系统、文本处理、xlsx/xls文件管理等领域，需要知道在Python/Perl中如何查询和安装这些模块。

1）Python的第三方模块分发网址pypi：<https://pypi.org/>

通过自带工具pip进行安装，方法如下：

pip install module\_name

通过自带工具pip进行查找，方法如下：

pip search module\_name

2）Perl的第三方模块分发网址cpan：<https://metacpan.org/>

通过自带工具cpan进行安装，方法如下：

cpan install module\_name

或：

cpan module\_name

### 模块使用

1）Python的模块导入方式有两种：

导入模块的全部内容 import os

导入模块的指定内容 from sys import argv

2）Perl的模块导入方式有两种：

导入模块的全部内容 use File::Spec;

导入模块的指定内容 use File::Basename qw/basename/;

## 数据类型

这两门语言都是无类型的，也就是说，对同一个变量名可以例化为多个不同的变量类型，如Python中的：

a = 50

a = “hello world!”

和Perl中的：

$a = 50;

$a = “hello world!”;

这样写，解释器是不会报错的。每赋一次值，解释器就会根据等号右边的数据类型，推断出左边的变量类型，将相应的数据绑定在这个变量名a上。在C上是不允许这么操作的，编译器会报错并终止编译。

注意：

1）Python和Perl中的变量名区分大小写，且首位不能为数字；

2）Perl中的变量必须以“$”开头（部分系统变量不是，但用户自定义的变量最好遵循这一条，包括定义读写文件的句柄变量）。

### 整型变量

整型变量包含有符号和无符号两种。

### 浮点型变量

浮点数是一种小数点位置不定的实数数据类型。Python和Perl中的浮点数默认为双精度浮点型，占用64位，足以满足一般运算的精度需求。

### 字符串变量

字符串变量是一种用于存储多个字符的数据类型，字符串代表的含义与字符串的编码有关。常见的字符串编码方式有ASCII、GB2312、UTF-8、UTF-16等。ASCII码不支持中文，GB2312、UTF-8/16编码格式支持中文。

Python3.0中默认的字符串编码方式是UTF-8。Perl中默认的字符串编码方式是ASCII，通过命令“use utf8;”修改为支持UTF-8编码。

## 数据结构

### 列表

列表是一种顺序存储数据的数据结构，允许重复数据的存取。

Python中列表的常用方式如下：

|  |  |  |
| --- | --- | --- |
| 操作 | 方法 | 示例 |
| 定义一个空列表 | “[]”或“list()” | 1. vector = []  2. vector = list() |
| 向列表尾部添加一个元素 | 方法append() | 1. vector.append(123) |
| 将列表尾部的最后一个元素弹出 | 方法pop() | 1. vector.pop() |
| 向指定位置插入一个元素 | 方法insert() | 1. vector.insert(k, element) |
| 清空列表 | 关键词del | 1. del vector |
| 遍历列表 | for … in …: | 1. for element in vector:  print(element)  2. for i in range(len(vector)):  print(vector[i]) |
|  |  |  |
|  |  |  |

Perl中列表的常用方式如下：

|  |  |  |
| --- | --- | --- |
| 操作 | 方法 | 示例 |
| 定义一个空列表 | “@” | 1. my @vector = (); |
| 向列表尾部添加一个元素 | 函数push() | 1. push(@vector, 123); |
| 将列表尾部的最后一个元素弹出 | 函数pop() | 1. pop(@vector); |
| 向指定位置插入一个元素 |  |  |
| 清空列表 | () | 1. @vector = (); |
| 遍历列表 | foreach … (@ ) | 1. foreach my $element (@vector)  { } |
|  |  |  |
|  |  |  |

### 字典/hash表

字典/hash表是一种用于存储“键—值”对（Key-Value Pair）的数据结构，要求结构中的键唯一。

Python中字典的常用方式如下：

|  |  |  |
| --- | --- | --- |
| 操作 | 方法 | 示例 |
| 定义一个空字典 | “{}”或“dict()” | hash = {}  hash = dict() |
| 判断元素是否存在于字典中 | if … in … : | if key in hash:  … |
| 向字典中添加元素 | [] | hash[key] = value |
| 清空字典 | del … | del hash 或 hash = {} |
| 遍历字典 | for … in …:  … | for key in hash:  value = hash[key]  print(“%s %s” % (key, value)) |
|  |  |  |
|  |  |  |

Perl中hash表的常用方式如下：

|  |  |  |
| --- | --- | --- |
| 操作 | 方法 | 示例 |
| 定义一个空字典 | “%” | my %hash = (); |
| 判断元素是否存在于字典中 | exists() | if exists($hash{$element}) {    } |
| 向字典中添加元素 | “$”和“[]” | $hash[$key] = $value; |
| 清空字典 | () | %hash = (); |
| 遍历字典 | foreach … (… keys .. ) | 1. 键为字符串，按键由小至大的顺序进行遍历  foreach my $key (sort{$a cmp $b} keys %hash) {    }  2. 键为数字，按键由小至大的顺序进行遍历  foreach my $key (sort{$a <=> $b} keys %hash) {    }  3. 值为字符串，按值由小至大的顺序进行遍历  foreach my $key (sort{$hash{$a} cmp $hash{$b}} keys %hash) {    }  4. 值为数字，按值由小至大的顺序进行遍历  foreach my $key (sort{$hash{$a} <=> $hash{$b}} keys %hash) {    } |
|  |  |  |
|  |  |  |

## 常用语法

### 函数定义

建议在脚本中，通过定义函数，实现代码复用。

下面以加法函数为例，介绍函数的定义模板。

Python中定义函数的模板为：

def function\_name(a, b):

return a+b

其中：

1）def 为Python中定义函数的关键词；

2）a, b为函数的参数，按顺序写入和使用；

3）return后跟函数的返回值，如不写，则无返回值；

4）Python中以空格或制表符tab（\t）为函数作用域的标志，相同缩进表示代码块属于一个函数中。

**5）python支持lambda表达式，允许函数在定义时不定义函数名，lambda表达式的定义如下：**

**lambda a, b: a + b**

使用时配合map、filter等函数，使用示例如下：

filter(lambda x:x%2 == 1, list) # 对列表list进行过滤，获取满足条件的元素，生成新的列表

Perl中定义函数的模板为：

sub function\_name

{

my ($a, $b) = @\_;

return ($a+$b);

}

其中：

1）sub为Perl中定义函数的关键词：

2）Perl中的参数默认放在内置的列表变量@\_中，使用参数时需按顺序解析和使用；

3）return后跟函数的返回值，如不写，则默认返回函数中最后一句语句结果；

4）Perl中以“{}”为函数的作用域标志。

## 文件和文件夹处理

linux环境下的换行符为“\n”，windows环境下的换行符为”\r\n”，以下对不同环境的换行符不做区分。

无论Python还是Perl，对文件的操作均是通过句柄进行的，通过open函数申请文件句柄，通过close关闭文件句柄（除Python的with语句外，open和close函数必须配对，否则会导致资源泄露，影响向系统申请新的文件句柄，这是由于任意系统可申请的文件句柄数量是有限的）。

Python句柄的类型根据如下格式进行定义：

|  |  |  |  |
| --- | --- | --- | --- |
| 文件类型 | 操作 | 格式 | 追加 |
| 文本文件 | 读 | “r” | “r+” |
| 写 | “w” | “w+” |
| 二进制文件 | 读 | “rb” | “rb+” |
| 写 | “wb” | “wb+” |

Perl句柄的类型根据如下格式进行定义：

|  |  |  |  |
| --- | --- | --- | --- |
| 文件类型 | 操作 | 格式 | 追加 |
| 文本文件 | 读 | “<” | “<<” |
| 写 | “>” | “>>” |
| 二进制文件 | 读 | “<”  binmode(文件句柄) | “<<”  binmode(文件句柄) |
| 写 | “>”  binmode(文件句柄) | “>>”  binmode(文件句柄) |

### 写文件

Python进行写文件的模板代码如下：

with open(file, “w”) as fout:

print(“hello world!”, file=fout)

注意：Python的print函数默认添加换行符。

Perl进行写文件的模板代码如下：

open my $fout, “>”, $file or die “$file is written wrong!”;

print $fout “hello world!\n”;

close($fout);

注意：

1. Perl的print函数默认不添加换行符，需手动添加；
2. open和close必须配对，其中open为开文件句柄，close为关文件句柄。

### 读文件

Python进行读文件的模板代码如下：

with open(file, “r”) as fin:

lines = fin.readlines()

for line in lines:

print(line)

Perl进行读文件的模板代码如下：

open my $fin, “<”, $file or die “$file is read wrong!”;

while(<$fin>)

{

# 默认每行数据存放于参数$\_中

# Perl中的函数不输入参数时，默认对变量$\_进行处理。

chmop; # 去除换行符，可用 $\_ =~ s/\n//g; $\_ =~ s/\r//g; 替代，做跨平台处理

my $line = $\_;

}

close($fin);

注意：

1. open和close必须配对，其中open为开文件句柄，close为关文件句柄。

### 遍历文件和文件夹

Python通过os模块中的walk函数，遍历给定目录下的所有文件和文件夹，

import os

path = “.”

for root, dirs., files in os.walk(path):

for dir in dirs.:

path = os.path.join(root, dir)

……

for file in files:

path = os.path.join(root, file)

……

Perl通过opendir申请句柄，读取当前目录下的所有文件和文件，如下所示：

my $path = “.”;

opendir (my $din, $path) or die “$din cannot be read!”;

my @dirs = readdir($din);

closedir($din);

foreach my $dir (@dirs)

{

if((-d $dir) && ($dir !~ /^\./)) # -d 判断是否为目录，并且目录不以“.”开头，排除Linux环境中的当前目录“.”、上级目录“..”和“.svn”、“.vim”等目录

{ }

elsif((-f $dir) && ($dir !~ /^\./)) # -f 判断是否为文件，并且文件不以“.”开头，排除Linux环境中的“.cshrc”、“.vimrc”等配置文件

{ }

}

## 系统调用

Python调用外部程序os.system(cmd)

Perl调用外部程序 system(cmd)

## 正则表达式

Python的正则表达式模块是re，参考Perl的规则实现，两者表达式的含义相同，统一进行介绍。如单独编译正则表达式，可在字符串前加r，如下所示：

r”\*.c$”

Perl的正则表达式是内置功能，无需导入额外模块。

正则表达式较为复杂，无法在此详细展开描述。结合一般场景的需求，给出下列几种常用的表达式及其含义：

1. 匹配以字符串string开头的字符串

“^string”

2)匹配以文件后缀名.txt结尾的字符串

“\.txt$”

3)匹配空格

单个空格：“\s”

一个以上的空格“\s+”

没有或有一个或一个以上的空格：“\s\*”

3）匹配制表符

单个制表符：“\t”

一个以上的制表符：“\t+”

没有或有一个或一个以上的制表符：“\t\*”

1. 匹配数字0-9

单个数字：“[0-9]”

一个以上的数字：“[0-9]+”

没有或有一个或一个以上的数字：“[0-9]\*”

1. 匹配英文字母a-zA-Z

单个英文字母：“[a-zA-Z]”

一个以上的英文字母：“[a-zA-Z]+”

没有或有一个或一个以上的英文字母：“[a-zA-Z]\*”

## Makefile编写

1）调用外部shell命令$(shell echo “hello world!”)

2）通过export可将定义的变量传递至下一级Makefile中

3）make运行指定目录中的Makefile：make –C $(dir)

4）make运行指定的Makefile：make –f $(dir)/Makefile

以上可避免在Makefile中直接调用cd进入不同的目录，再执行make程序。此种做法不利于Makefile文件的移植，由此所有的项目可以以一套Makefile模板为基础进行修改和维护。