# 阅读 6



## Python基础

- 今天我们将学习一种叫做 Python 的新编程语言。 一种比 C 更新的语言，它具有额外的功能和简单性，因此很受欢迎。
- Python 中的源代码看起来比 C 简单得多。实际上，要打印“hello, world”，我们只需要编写以下代码：

```language
print("hello, world")
```

  - 请注意，与 C 不同，我们不需要在 `print` 函数或使用分号结束我们的行。
  - 要编写和运行这个程序，我们将使用 CS50 IDE，将新文件另存为 `hello.py` 仅使用上面的行，然后运行命令 `python hello.py`.
- 我们可以从用户那里获取字符串：

```language
answer = get_string("What's your name? ")
print("hello, " + answer)
```

- 我们还需要导入Python版本的CS50库， `cs50`，对于函数 `get_string`，所以我们的代码将如下所示：

```language
from cs50 import get_string

answer = get_string("What's your name? ")
print("hello, " + answer)
```

- 我们创建一个名为 `answer` 在不指定类型的情况下，我们可以将两个字符串与 `+` 运算符在我们将其传递给之前 `print`.

- 我们可以使用 **格式字符串 ** ， `f"..."`，插入变量。 例如，我们可以写 `print(f"hello, {answer}")` 插入值 `answer` 通过用花括号将其包围到我们的字符串中。

- 我们可以创建变量 `counter = 0`. 通过赋值 `0`，我们隐式地将类型设置为整数，因此我们不需要指定类型。 要增加一个变量，我们可以使用 `counter = counter + 1` 或者 `counter += 1`.

- 条件如下：

```language
if x < y:
    print("x is less than y")
elif x > y:
    print("x is greater than y")
else:
    print("x is equal to y")
```

- 与 C 语言中花括号用于指示代码块不同，每行的确切缩进决定了 Python 中的嵌套级别。

- 而不是 `else if`，我们只是说 `elif`.

- 布尔表达式也略有不同：

```language
while True:
    print("hello, world")
```

- 两个都 `True` 和 `False` 在 Python 中大写。

- 我们可以写一个带变量的循环：

```language
i = 0
while i < 3:
    print("hello, world")
    i += 1
```

- 我们也可以使用一个 `for` 循环，我们可以为列表中的每个值做一些事情：

```language
for i in [0, 1, 2]:
    print("cough")
```

  - Python中的列表， `[0, 1, 2]`，就像 C 中的数组。
  - 这 `for` 循环将设置变量 `i` 到第一个元素， `0`，运行，然后到第二个元素， `1`、运行等。
  - 我们可以使用一个特殊的功能， `range`，得到一些值，如 `for i in range(3):`. `range(3)` 会给我们一个最多但不包括 3 个的列表，其中包含值 `0`， `1`， 和 `2`，然后我们可以使用。 `range()` 也接受其他选项，因此我们可以拥有以不同值开始并且值之间具有不同增量的列表。 通过查看 文档，我们可以使用 `range(0, 101, 2)` 得到一个范围 `0` 到 `100` （因为第二个值是独占的），递增 `2` 一次。
  - 打印出来 `i`，我们也可以写 `print(i)`.
  - 由于在 Python 中编写相同的代码通常有多种方法，因此最常用和接受的方法称为 **Pythonic ** 。

- 在 Python 中，有许多内置的数据类型：
  - `bool`， `True` 或者 `False`
  - `float`，实数
  - `int`，整数
  - `str`，字符串

- 虽然 C 是一种 **强类型 **语言，我们需要在其中指定类型，但 Python 是 **松散类型 **的，其中类型由值隐含。

- Python 中的其他类型包括：
  - `range`，数字序列
  - `list`， 可变值序列，或我们可以改变的值

    - 列表，即使它们就像 C 中的数组，也可以在 Python 中自动增长和收缩
  - `tuple`，有序值的集合，如 x 和 y 坐标，或经度和纬度
  - `dict`，字典，键/值对的集合，如哈希表
  - `set`，唯一值的集合，或没有重复的值
- Python 的 CS50 库包括：
  - `get_float`
  - `get_int`
  - `get_string`
- 我们可以一次导入一个函数，或者一起导入：

language

```
from cs50 import get_float
from cs50 import get_int
from cs50 import get_string
```

language

```
import cs50
```

language

```
from cs50 import get_float, get_int, get_string
```

## 例子

- 由于 Python 包含许多功能以及其他人编写的代码库，我们可以在更高的抽象层次上解决问题，而不是自己实现所有细节。
- 我们可以使用以下方法模糊图像：

language

```
from PIL import Image, ImageFilter

before = Image.open("bridge.bmp")
after = before.filter(ImageFilter.BoxBlur(1))
after.save("out.bmp")
```

  - 在 Python 中，我们包含其他库 `import`，在这里我们将 `import` 这 `Image` 和 `ImageFilter` 来自的名字 `PIL` 图书馆。 （其他人编写了这个库等等，并提供给我们所有人下载和使用。）

    - `Image`是一个结构，不仅有数据，还有我们可以访问的函数`.`语法，例如 with `Image.open`.
    - 我们打开一个名为`bridge.bmp`，调用模糊过滤器函数，并将其保存到一个名为`out.bmp`.
    - 我们可以运行它`python blur.py`保存到名为的文件后`blur.py`.
- 我们可以实现一个字典：

```language
words = set()

def load(dictionary):
  file = open(dictionary, "r")
  for line in file:
      words.add(line.rstrip())
  file.close()
  return True

def check(word):
    if word.lower() in words:
        return True
    else:
        return False

def size():
    return len(words)

def unload():
    return True
```

    - 首先，我们创建一个名为`words`.
    - 请注意，我们不需要`main`功能。 我们的 Python 程序将从上到下运行。 在这里，我们要定义一个函数，所以我们使用`def load()`. `load`将带一个参数，`dictionary`，并且它的返回值是隐含的。 我们打开文件`open`，并迭代文件中的行`for line in file:`. 然后，我们删除末尾的换行符`line`，并将其添加到我们的集合中`words`. 请注意`line`是一个字符串，但有一个`.rstrip`我们可以调用的函数。
    - 那么，对于`check`，我们可以问`if word.lower() in words`. 为了`size`， 我们可以用`len`计算我们集合中元素的数量，最后，对于`unload`，我们不需要做任何事情，因为 Python 会为我们管理内存。

- 事实证明，尽管在 Python 中实现程序对我们来说更简单，但我们在 Python 中的程序的运行时间比我们在 C 中的程序要慢，因为该语言必须通过通用解决方案为我们做更多的工作，比如内存管理。
- 另外，Python 也是一个叫做 **解释器 ** ，它读入我们的源代码，逐行翻译成我们的 CPU 可以理解的代码。
- 例如，如果我们从第 0 周开始的伪代码是西班牙语，而我们不懂西班牙语，我们将不得不慢慢地将其逐行翻译成英语，然后才能在电话簿中搜索姓名：

```language
1  Recoge guía telefónica
2  Abre a la mitad de guía telefónica
3  Ve la página
4  Si la persona está en la página
5      Llama a la persona
6  Si no, si la persona está antes de mitad de guía telefónica
7      Abre a la mitad de la mitad izquierda de la guía telefónica
8      Regresa a la línea 3
9  Si no, si la persona está después de mitad de guía telefónica
10     Abre a la mitad de la mitad derecha de la guía telefónica
11     Regresa a la línea 3
12 De lo contrario
13     Abandona
```

- 因此，根据我们的目标，我们还必须考虑编写更高效的程序的人力时间与程序的运行时间之间的权衡。

## 输入、条件

- 我们可以通过 `input` 功能：

```language
answer = input("What's your name? ")
print(f"hello, {answer}")
```

- 我们可以向用户询问两个整数并将它们相加：

```language
from cs50 import get_int

# Prompt user for x
x = get_int("x: ")

# Prompt user for y
```

```
y = get_int("y: ")

# Perform addition
print(x + y)
```

- 评论开头 `#` 代替 `//`.

- 如果我们自己调用 input，我们会得到我们的值的字符串：

```language
# Prompt user for x
x = input("x: ")

# Prompt user for y
y = input("y: ")

# Perform addition
print(x + y)
```

- 需要 **从 **我们 `input` 成一个 `int` 在我们存储它之前：

```language
# Prompt user for x
x = int(input("x: "))

# Prompt user for y
y = int(input("y: "))

# Perform addition
print(x + y)
```

- 但是如果用户没有输入数字，我们就需要做更多的错误检查，否则我们的程序就会崩溃。 所以我们通常会想使用一个常用的库来解决这样的问题。

- 我们将值划分：

```language
# Prompt user for x
x = int(input("x: "))

# Prompt user for y
y = int(input("y: "))

# Perform division
print(x / y)
```

- 请注意，即使我们将两个整数相除，我们也会得到浮点、十进制值。

- 我们可以证明条件：

```language
from cs50 import get_int

x = get_int("x: ")
y = get_int("y: ")

if x < y:
    print("x is less than y")
elif x > y:
    print("x is greater than y")
else:
    print("x is equal to y")
```

- 我们可以导入整个库，并在其中使用函数，就好像它们是结构一样：

```language
import cs50

x = cs50.get_int("x: ")
y = cs50.get_int("y: ")
```

  - 如果我们的程序需要导入两个不同的库，每个库都有一个 `get_int` 函数，例如，我们需要使用这个方法来 **命名 **函数，将它们的名称保持在不同的空间中以防止它们发生冲突。

- 要比较字符串，我们可以说：

```language
from cs50 import get_string

s = get_string("Do you agree? ")

if s == "Y" or s == "y":
    print("Agreed.")
elif s == "N" or s == "n":
    print("Not agreed.")
```

  - Python没有字符，所以我们检查 `Y` 和其他字母作为字符串。 我们也可以直接比较字符串 `==`. 最后，在我们的布尔表达式中，我们使用 `or` 和 `and` 而不是符号。
  - 我们也可以说 `if s.lower() in ["y", "yes"]:` 检查我们的字符串是否在列表中，首先将其转换为小写。

## 喵

- 我们可以改进版本 `meow`， 也：

```language
print("meow")
print("meow")
print("meow")
```

  - 我们不需要声明一个 `main` 函数，所以我们只需将同一行代码写三遍。

- 我们可以定义一个可以重用的函数：

```language
for i in range(3):
    meow()

def meow():
    print("meow")
```

  - 但是，当我们尝试运行它时，这会导致错误： `NameError: name 'meow' is not defined`. 事实证明，我们需要在使用它之前定义我们的函数，所以我们可以移动我们的定义 `meow` 到顶部，或者先定义一个主函数：

```language
def main():
    for i in range(3):
        meow()

def meow():
    print("meow")

main()
```

  - 现在，当我们真正调用我们的 `main` 函数 `meow` 函数将已经被定义。

- 我们的函数也可以接受输入：

```language
def main():
    meow(3)

def meow(n):
    for i in range(n):
        print("meow")

```

```
8
main()
```

- 我们的 `meow` 函数接受一个参数， `n` ，并将其传递给 `range` .

## get_positive_int

- 我们可以定义一个函数来获取一个正整数：

language

```
from cs50 import get_int

def main():
    i = get_positive_int()
    print(i)

def get_positive_int():
    while True:
        n = get_int("Positive Integer: ")
        if n > 0:
            break
    return n

main()
```

- 由于 Python 中没有像 C 中那样的 do-while 循环，因此我们有一个 `while` 将无限进行的循环，并使用 `break` 尽快结束循环 `n > 0` . 最后，我们的函数将 `return n` ，在我们原来的缩进级别，在 `while` 环形。
- 请注意，Python 中的变量 **作用域为函数 ** ，这意味着 `n` 可以在循环中初始化，但在函数的后面仍然可以访问。

## 马里奥

- 我们可以在屏幕上打印出一行问号：

language

```
for i in range(4):
    print("?", end="")
print()
```

- 当我们打印每个块时，我们不希望自动换行，因此我们可以将 **命名参数 ** （也称为关键字参数）传递给 `print` 函数，它指定特定参数的值。 到目前为止，我们只看到了 **位置参数 ** ，其中参数是根据它们在函数调用中的位置来设置的。

    - 在这里，我们说 `end=""` 指定不应在我们的字符串末尾打印任何内容。 `end` 也是一个 **可选参数 ** ，我们不需要传入，默认值为 `\n` ， 这就是为什么 `print` 通常会为我们添加一个新行。
    - 最后，在我们用循环打印我们的行之后，我们可以调用 `print` 没有其他参数来换行。
- 我们还可以“乘”一个字符串并直接打印： `print("?" * 4)` .
- 我们可以实现嵌套循环：

```language
for i in range(3):
    for j in range(3):
        print("#", end="")
    print()
```

## 溢出，不精确

- 在 Python 中，试图引起整数溢出实际上是行不通的：

```language
i = 1
while True:
    print(i)
    i *= 2
```

    - 我们看到打印的数字越来越大，因为 Python 自动使用越来越多的内存来为我们存储数字，这与 C 中的整数固定为特定字节数不同。
- 浮点不精确性也仍然存在，但可以通过可以用所需位数表示十进制数的库来防止。

## 列表、字符串

- 我们可以列一个清单：

```language
scores = [72, 73, 33]

print("Average: " + str(sum(scores) / len(scores)))
```

    - 我们可以用 `sum` ，一个内置于 Python 的函数，用于将列表中的值相加，然后除以分数的数量，使用 `len` 获取列表长度的函数。 然后，我们将浮点数转换为字符串，然后才能连接并打印它。

- 我们甚至可以将整个表达式添加到格式化字符串中以获得相同的效果：

```language
print(f"Average: {sum(scores) / len(scores)}")
```

- 我们可以通过以下方式将项目添加到列表中：

```language
from cs50 import get_int

scores = []
for i in range(3):
    scores.append(get_int("Score: "))
...
```

- 我们可以遍历字符串中的每个字符：

```language
from cs50 import get_string

s = get_string("Before:  ")
print("After: ", end="")
for c in s:
    print(c.upper(), end="")
print()
```

  - Python将为我们遍历字符串中的每个字符 `for c in s`.

- 要使字符串大写，我们也可以调用 `s.upper()`，而不必自己迭代每个字符。

## 命令行参数，退出代码

- 我们可以使用命令行参数：

```language
from sys import argv

if len(argv) == 2:
    print(f"hello, {argv[1]}")
else:
    print("hello, world")
```

  - 我们进口 `argv` 从 `sys`，或系统模块，内置于 Python 中。

    - 自从 `argv` 是一个列表，我们可以得到第二个项目 `argv[1]`，所以用命令添加一个参数 `python argv.py David` 将导致 `hello, David` 打印。
    - 就像在 C 中一样， `argv[0]` 将是我们程序的名称，例如 `argv.py`.
- 我们还可以让 Python 为我们遍历列表：

```language
from sys import argv

for arg in argv:
    print(arg)
```

- 当我们的程序退出时，我们也可以返回退出代码：

```language
import sys

if len(sys.argv) != 2:
    print("missing command-line argument")
    sys.exit(1)
print(f"hello, {sys.argv[1]}")
sys.exit(0)
```

    - 我们导入整个 `sys` 现在是模块，因为我们正在使用它的多个组件。 现在我们可以使用 `sys.argv` 和 `sys.exit()` 使用特定代码退出我们的程序。

## 算法

- 我们可以通过检查列表中的每个元素来实现线性搜索：

```language
import sys

numbers = [4, 6, 8, 2, 7, 5, 0]

if 0 in numbers:
    print("Found")
    sys.exit(0)

print("Not found")
sys.exit(1)
```

    - 和 `if 0 in numbers:`，我们要求 Python 为我们检查列表。
- 字符串列表也可以通过以下方式搜索：

language

```
names = ["Bill", "Charlie", "Fred", "George", "Ginny", "Percy", "Ron
if "Ron" in names:
    print("Found")
else:
    print("Not found")
```

- 如果我们有一个字典，一组键值对，我们还可以检查一个特定的键，并查看为它存储的值：

language

```
from cs50 import get_string

people = {
    "Brian": "+1-617-495-1000",
    "David": "+1-949-468-2750"
}

name = get_string("Name: ")
if name in people:
    print(f"Number: {people[name]}")
```

  - 我们首先声明一个字典，`people`，其中键是我们要存储的每个名称的字符串，我们要与每个键关联的值是对应电话号码的字符串。
  - 然后，我们使用`if name in people:`搜索我们字典的键`name`. 如果键存在，那么我们可以用括号表示法获取值，`people[name]`，很像用 C 对数组进行索引，除了这里我们使用字符串而不是整数。
  - 字典和集合通常在 Python 中实现，具有像哈希表这样的数据结构，因此我们可以进行接近恒定时间的查找。 同样，我们需要权衡的是对底层发生的事情的控制更少，比如能够选择一个哈希函数，而这样做的好处是我们自己做的工作更少。

- 交换两个变量也可以简单地通过同时分配两个值来完成：

language

```
x = 1
y = 2

print(f"x is {x}, y is {y}")
x, y = y, x
print(f"x is {x}, y is {y}")
```

  - 在 Python 中，我们无法访问指针，这可以防止我们在内存方面犯错。

# 文件

- 让我们打开一个 CSV 文件：

```language
import csv

from cs50 import get_string

file = open("phonebook.csv", "a")

name = get_string("Name: ")
number = get_string("Number: ")

writer = csv.writer(file)
writer.writerow([name, number])

file.close()
```

  - 原来Python也有一个 `csv` 帮助我们处理 CSV 文件的库，所以在我们打开文件进行追加后，我们可以调用 `csv.writer` 创建一个 `writer` 来自文件，它提供了额外的功能，比如 `writer.writerow` 将列表写成一行。

- 我们可以使用 `with` 关键字，它会在我们完成后为我们关闭文件：

```language
...
with open("phonebook.csv", "a") as file:
    writer = csv.writer(file)
    writer.writerow((name, number))
```

- 我们可以打开另一个 CSV 文件，计算一个值出现的次数：

```language
import csv

houses = {
    "Gryffindor": 0,
    "Hufflepuff": 0,
    "Ravenclaw": 0,
    "Slytherin": 0
}

with open("Sorting Hat (Responses) - Form Responses 1.csv", "r") as
    reader = csv.reader(file)
    next(reader)
```

```
    next(reader)
    for row in reader:
        house = row[1]
        houses[house] += 1

for house in houses:
    print(f"{house}: {houses[house]}")
```

- 我们使用 `reader` 从函数 `csv` 库，跳过标题行 `next(reader)`，然后遍历其余的每一行。
- 每行的第二个项目，`row[1]`，是房子的字符串，所以我们可以使用它来访问存储在 `houses` 为该键，并添加一个。
- 最后，我们将打印出每所房子的计数。

## 更多库

- 在我们自己的 Mac 或 PC 上，我们可以在安装 Python 后打开一个终端，并使用另一个库将文本转换为语音：

language

```
import pyttsx3

engine = pyttsx3.init()
engine.say("hello, world")
engine.runAndWait()
```

- 通过阅读文档，我们可以弄清楚如何初始化库，并说一个字符串。
- 我们甚至可以传入一个格式字符串 `engine.say(f"hello, {name}")` 说一些输入。

- 我们可以使用另一个库，`face_recognition`，在图像中查找人脸：

language

```
# Find faces in picture
# https://github.com/ageitgey/face_recognition/blob/master/examples/

from PIL import Image
import face_recognition

# Load the jpg file into a numpy array
image = face_recognition.load_image_file("office.jpg")

# Find all the faces in the image using the default HOG-based model.
# This method is fairly accurate, but not as accurate as the CNN mod
# See also: find_faces_in_picture_cnn.py
face_locations = face_recognition.face_locations(image)
```

```
face_locations = face_recognition.face_locations(image)

for face_location in face_locations:

    # Print the location of each face in this image
    top, right, bottom, left = face_location

    # You can access the actual face itself like this:
    face_image = image[top:bottom, left:right]
    pil_image = Image.fromarray(face_image)
    pil_image.show()
```

- 使用 recognize.py，我们可以编写一个程序来查找特定人脸的匹配项。
- 我们可以使用另一个库创建 QR 码或二维条码：

language

```
import os
import qrcode

img = qrcode.make("https://youtu.be/oHg5SJYRHA0")
img.save("qr.png", "PNG")
os.system("open qr.png")
```

- 我们可以识别来自麦克风的音频输入：

language

```
import speech_recognition

# Obtain audio from the microphone
recognizer = speech_recognition.Recognizer()
with speech_recognition.Microphone() as source:
    print("Say something:")
    audio = recognizer.listen(source)

# Recognize speech using Google Speech Recognition
print("You said:")
print(recognizer.recognize_google(audio))
```

  - 我们正在按照图书馆的文档来收听我们的麦克风并将其转换为文本。

- 我们甚至可以为基本响应添加额外的逻辑：

language

```
...
words = recognizer.recognize_google(audio)

```

```
# Respond to speech
if "hello" in words:
    print("Hello to you too!")
elif "how are you" in words:
    print("I am well, thanks!")
elif "goodbye" in words:
    print("Goodbye to you too!")
else:
    print("Huh?")
```

- 最后，我们使用另一个更复杂的程序来生成深度伪造，或者是看起来很逼真但由计算机生成的各种个性的视频。
- 通过利用在线免费提供的所有这些库，我们可以轻松地将高级功能添加到我们自己的应用程序中。