

[笔记][“笨办法”学Python 3]

腾蛇起陆

Python

[笔记][“笨办法”学Python 3]

前言

习题0. 准备工作

习题1. 第一个程序

习题2. 注释和#号

习题3. 数字和数学计算

习题4. 变量和命名

习题5. 更多的变量和打印

习题6. 字符串和文本

习题7. 更多打印

习题8. 打印，打印

习题9. 打印，打印，打印

习题10. 那是什么

习题11. 提问

习题12. 提示别人

习题13. 参数、解包和变量

习题14. 提示和传递

习题15. 读取文件

习题16. 读写文件

习题17. 更多文件操作

习题18. 命名、变量、代码和函数

习题19. 函数和变量

习题20. 函数和文件

习题21. 函数可以返回某些东西

习题22. 到现在为止你学到了什么

习题23. 字符串、字节串和字符编码

习题24. 更多的练习

习题25. 更多更多的练习

习题26. 恭喜你，现在可以考试了！

习题27. 记住逻辑关系

习题28. 布尔表达式练习

习题29. if语句

习题30. else和if

习题31. 作出决定

习题32. 循环和列表

习题33. while循环

习题34. 访问列表的元素

习题35. 分支和函数

习题36. 设计和调试
习题37. 复习各种符号
习题38. 列表的操作
习题39. 字典，可爱的字典
习题40. 模块、类和对象
习题41. 学习面向对象术语
习题42. 对象、类及从属关系
习题43. 基本的面向对象分析和设计
习题44. 继承与组合
习题45. 你来制作一款游戏
习题46. 项目骨架
习题47. 自动化测试
习题48. 用户输入进阶
习题49. 创建句子
习题50. 你的第一个网站
习题51. 从浏览器获取输入
习题52. 创建 Web 游戏
接下来的路
老程序员的建议
附录：命令行快速入门
 简介：废话少说，命令行来也
 练习1. 准备工作
 练习2. 路径、文件夹和目录（ pwd ）
 练习3. 如果你迷失了
 练习4. 创建目录（ mkdir ）
 练习5. 更改目录（ cd ）
 练习6. 列出目录中的内容（ ls ）
 练习7. 删除目录（ rmdir ）
 练习8. 在多个目录中切换（ pushd 和 popd ）
 练习9. 创建空文件（ touch / New-Item ）
 练习10. 复制文件（ cp ）
 练习11. 移动文件（ mv ）
 练习12. 查看文件内容（ less / more ）
 练习13. 流文件内容显示（ cat ）
 练习14. 删除文件（ rm ）
 练习15. 退出终端（ exit ）

前言

“笨办法”指的是“指令式”教学方式。

`debug` 调试

习题0. 准备工作

Windows 的 PowerShell

macOS 的 Terminal

Linux 的 bash

Vim 用 `:q!` 或者 `ZZ` 退出

需要这些就够了

- 文本编辑器
- 命令行终端
- Python

习题1. 第一个程序

ex1.py

```
print("Hello World!")
print("Hello Again!")
print("I like typing this.")
print('Yay! Printing.')
print("I'd much rather you 'not'.")
print('I "said" do not touch this.')
```

```
PS C:\code\lpthw> python ex1.py
Hello World!
Hello Again!
I like typing this.
Yay! Printing.
I'd much rather you 'not'.
I "said" do not touch this.
PS C:\code\lpthw>
```

SyntaxError 语法错误

octothorpe , pound , hash , mesh

注释代码

习题2. 注释和 # 号

ex2.py

```
# A comment, this is so you can read your program later.  
# Anything after the # is ignored by python.  
  
print("I could have code like this.") # and the comment after is ignored  
  
# You can also use a comment to "disable" or comment out code:  
# print("This won't run.")  
  
print("This will run.")
```

```
PS C:\code\lpthw> python ex2.py  
I could have code like this.  
This will run.  
PS C:\code\lpthw> █
```

pound character

多行注释

每一行都放一个 #

倒着阅读代码是很好的查错技巧。

习题3. 数字和数学计算

ex3.py

```
print("I will now count my chickens:")  
  
print("Hens", 25 + 30 / 6)  
print("Roosters", 100 - 25 * 3 % 4)  
  
print("Now I will count the eggs:")  
  
print(3 + 2 + 1 - 5 + 4 % 2 - 1 / 4 + 6)  
  
print("Is it true that 3 + 2 < 5 - 7?")  
  
print(3 + 2 < 5 - 7)  
  
print("What is 3 + 2?", 3 + 2)  
print("What is 5 - 7?", 5 - 7)  
  
print("Oh, that's why it's False.")  
  
print("How about some more.")  
  
print("Is it greater?", 5 > -2)
```

```
print("Is it greater or equal?", 5 >= -2)
print("Is is less or equal?", 5 <= 2)
```

运算优先级 PEMDAS

括号 Parentheses

指数 Exponents

乘 Multiplication

除 Division

加 Addition

减 Substraction

习题4. 变量和命名

变量 `variable`

`_` 下划线 `underscore` 在变量里通常被用作假想的空格，用来隔开单词。

`ex4.py`

```
cars = 100
space_in_a_car = 4.0
drivers = 30
passengers = 90
cars_not_driven = cars - drivers
cars_driven = drivers
carpool_capacity = cars_driven * space_in_a_car
average_passengers_per_car = passengers / cars_driven

print("There are", cars, "cars available.")
print("There are only", drivers, "drivers available")
print("There will be", cars_not_driven, "empty cars today.")
print("We can transport", carpool_capacity, "people today.")
print("We have", passengers, "to carpool today.")
print("We need to put about", average_passengers_per_car, "in each car.")
```

`=` 的作用是为数据（数值、字符串等）取名

单等号 `=` 将右边的值赋给左边的变量名

双等号 `==` 检查左右两边的值是否相等

`x = 100` 比 `x=100` 好：操作符两边加上空格会让代码更容易阅读。

习题5. 更多的变量和打印

ex5.py

```
my_name = 'Zed A. Shaw'
my_age = 35 # not a lie
my_height = 74 # inches
my_weight = 180 # lbs
my_eyes = 'blue'
my_teeth = 'Whitie'
my_hair = 'Brown'

print(f"Let's talk about {my_name}.")
print(f"He's {my_height} inches tall.")
print(f"He's {my_weight} pounds heavy.")
print(f"Actually that's not too heavy.")
print(f"He's got {my_eyes} eyes and {my_hair} hair.")
print(f"His teeth are usually {my_teeth} depending on the coffee.")

# this line is tricky, try to get it exactly right
total = my_age + my_height + my_weight
print(f"If I add {my_age}, {my_height}, and {my_weight} I get {total}.")
```

格式化字符串 `format string`

双引号内的文本是字符串

创建嵌入变量的字符串

字符串用 `f` 开头，是 `format` 格式化的意思

```
f"Hello {somevar}"
```

`round(1.73)` 得到 `2`，四舍五入

另外还有两种格式化字符串的方法

- 用 `'字符串{0}{1}'.format(参数1, 参数2, ...)` 的形式
比如 `print('My name is {0}, I am {1} years old.'.format('jpch89', 30))`
- 用 `'字符串%d%s' % (数字, 字符串)`
比如 `print('My name is %s, I am %d years old.' % ('jpch89', 30))`

习题6. 字符串和文本

ex6.py

```
types_of_people = 10
x = f"There are {types_of_people} types of people."
```

```

binary = "binary"
do_not = "don't"
y = f"Those who know {binary} and those who {do_not}."

print(x)
print(y)

print(f"I said: {x}")
print(f"I also said: '{y}'")

hilarious = False
joke_evaluation = "Isn't that joke so funny?! {}"

print(joke_evaluation.format(hilarious))

w = "This is the left side of..."
e = "a string with a right side."

print(w + e)

```

Python 可以通过单引号 `'` 或者双引号 `"` 来识别字符串

f-string

```
f"some stuff here {avariable}"
```

```
f"some other stuff {anothervar}"
```

还有一种 `.format()` 语法的格式化方式

习题7. 更多打印

ex7.py

```

print("Mary had a little lamb.")
print("Its fleece was white as {}".format('snow'))
print("And everywhere that Mary went.")
print("." * 10) # what'd that do?

end1 = "C"
end2 = "h"
end3 = "e"
end4 = "e"
end5 = "s"
end6 = "e"
end7 = "B"
end8 = "u"
end9 = "r"

```

```
end10 = "g"
end11 = "e"
end12 = "r"

# watch that comma at the end. try removing it to see what happens
print(end1 + end2 + end3 + end4 + end5 + end6, end=' ')
print(end7 + end8 + end9 + end10 + end11 + end12)
```

单引号和双引号都可以创建字符串，单引号常用于创建简短的字符串。

习题8. 打印，打印

ex8.py

```
formatter = '{} {} {} {}'
print(formatter.format(1, 2, 3, 4))
print(formatter.format("one", "two", "three", "four"))
print(formatter.format(True, False, False, True))
print(formatter.format(formatter, formatter, formatter, formatter))
print(formatter.format(
    "Try your",
    "Own text here",
    "Maybe a poem",
    "Or a song about fear"
))
```

习题9. 打印，打印，打印

ex9.py

```
# Here's some new strange stuff, remember type is exactly.

days = "Mon Tue Wed Thu Fri Sat Sun"
months = "Jan\nFeb\nMar\nApr\nMay\nJun\nJul\nAug"

print("Here are the days: ", days)
print("Here are the months: ", months)

print("""
There's something going on here.
With the three double-quotes.
```



```
We'll be able to type as much as we like.  
Even 4 lines if we want, or 5, or 6.  
"""
```

用 `"""..."""` 来输出多行字符串，而不用 `\n` 换行

习题10. 那是什么

ex10.py

```
tabby_cat = "\tI'm tabbed in."  
persian_cat = "I'm split\non a line."  
backslash_cat = "I'm \\ a \\ cat."  
  
fat_cat = """  
I'll do a list:  
\t* Cat food  
\t* Fishies  
\t* Catnip\n\t* Grass  
"""  
  
print(tabby_cat)  
print(persian_cat)  
print(backslash_cat)  
print(fat_cat)
```

`\n` 代表 new line character

`\` 开头的转义序列 escape sequence

`\'` 转义单引号

`\"` 转义双引号

或者使用三引号 `"""`，可以在一对三引号里放入任意多行文本。

转义序列

转义字符	功能
<code>\\</code>	反斜杠 (<code>\</code>)
<code>\'</code>	单引号 (<code>'</code>)
<code>\"</code>	双引号 (<code>"</code>)
<code>\a</code>	ASCII 响铃符 (<code>BEL</code>)
<code>\b</code>	ASCII 退格符 (<code>BS</code>)

<code>\f</code>	ASCII 进纸符 (<code>FF</code>)
<code>\n</code>	ASCII 换行符 (<code>LF</code>)
<code>\N{name}</code>	Unicode 数据库中的字复名，其中 <code>name</code> 是它的名字，仅 Unicode 适用
<code>\r</code>	ASCII 回车符 (<code>CR</code>)
<code>\t</code>	ASCII 水平制表符 (<code>TAB</code>)
<code>\uxxxx</code>	值为 16 位十六进制值 <code>xxxx</code> 的字符
<code>\Uxxxxxxxx</code>	值为 32 位十六进制值 <code>xxxx</code> 的字符
<code>\v</code>	ASCII 垂直制表符 (<code>VT</code>)
<code>\ooo</code>	值为八进制 <code>ooo</code> 的字符
<code>\xhh</code>	值为十六进制值 <code>hh</code> 的字符

【?】什么时候用 `'''` 代替 `"""`

【!】目前可以先用 `'''`

习题11. 提问

ex11.py

```
print("How old are you?", end = ' ')
age = input()
print("How tall are you?", end = ' ')
height = input()
print("How much do you weigh?", end = ' ')
weight = input()

print(f"So, you're {age} old, {height} tall and {weight} heavy.")
```

一般软件做的主要事情就是下面几件：

1. 接收输入的内容
2. 改变输入的内容
3. 打印出改变了的内容

在 `print` 后面加入 `end = ' '`，不要用换行符结束这一行，而用空格

习题12. 提示别人

ex12.py

```
age = input("How old are you?")
height = input("How tall are you?")
weight = input("How much do you weigh?")

print(f"So, you're {age} old, {height} tall and {weight} heavy.")
```

pydoc input

或者在 Windows 下用 `python -m pydoc input`

q 键退出

习题13. 参数、解包和变量

ex13.py

写一个可以接收参数 `arguments` 的脚本

所谓脚本就是 `.py` 程序

```
from sys import argv
# read the WYSS section for how to run this
script, first, second, third = argv

print("The script is called:", script)
print("Your first variable is:", first)
print("Your second variable is:", second)
print("Your third variable is:", third)
```

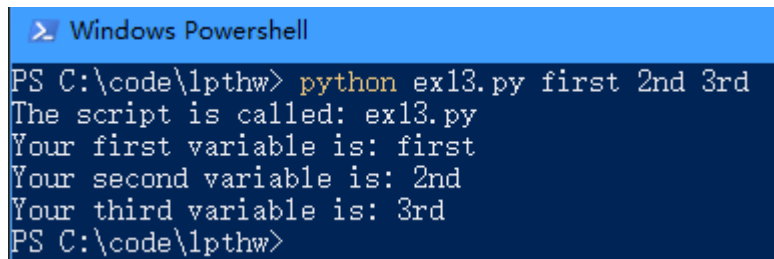
`argv` 是所谓的参数变量 (`argument variable`)

`sys.argv` 保存着你运行 Python 脚本时传递给 Python 脚本的参数

`script, first, second, third = argv` 将 `argv` 解包 (`unpack`)

即把 `argv` 中的所有东西取出，解包，将所有的参数依次赋值给左边的变量

通过 `import` 导入模块 `module`，有些人称之为库 `library`



```
> Windows Powershell
PS C:\code\lpthw> python ex13.py first 2nd 3rd
The script is called: ex13.py
Your first variable is: first
Your second variable is: 2nd
Your third variable is: 3rd
PS C:\code\lpthw>
```

argv 和 input() 有什么不同？

不同点在于用户输入的时机。

如果参数是在用户执行命令时就要输入，那就用 argv

如果实在脚本运行过程中需要用户输入，那就用 input()

习题14. 提示和传递

ex14.py

Zork 游戏和 Adventure 游戏

```
from sys import argv

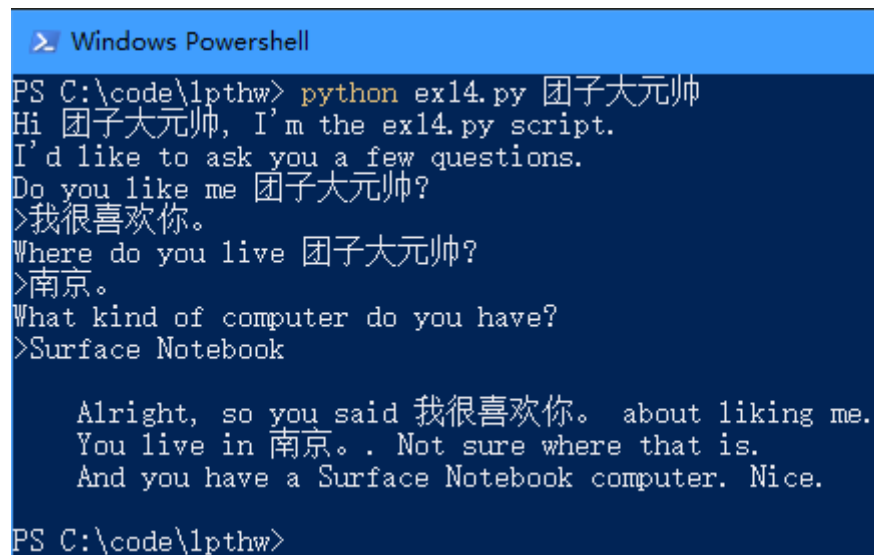
script, user_name = argv
prompt = '>'

print(f"Hi {user_name}, I'm the {script} script.")
print("I'd like to ask you a few questions.")
print(f"Do you like me {user_name}")
likes = input(prompt)

print(f"Where do you live {user_name}?")
lives = input(prompt)

print("What kind of computer do you have?")
computer = input(prompt)

print(f"""
    Alright, so you said {likes} about liking me.
    You live in {lives}. Not sure where that is.
    And you have a {computer} computer. Nice.
    """)
```



```
> Windows Powershell
PS C:\code\lpthw> python ex14.py 团子大元帅
Hi 团子大元帅, I'm the ex14.py script.
I'd like to ask you a few questions.
Do you like me 团子大元帅?
>我很喜欢你。
Where do you live 团子大元帅?
>南京。
What kind of computer do you have?
>Surface Notebook

    Alright, so you said 我很喜欢你。 about liking me.
    You live in 南京。。 Not sure where that is.
    And you have a Surface Notebook computer. Nice.

PS C:\code\lpthw>
```

将用户提示符设置为变量 `prompt`，这样就可以重复使用，并且修改会很方便

我又改了一版程序，注意多行字符串对输出变化的影响：

```
from sys import argv

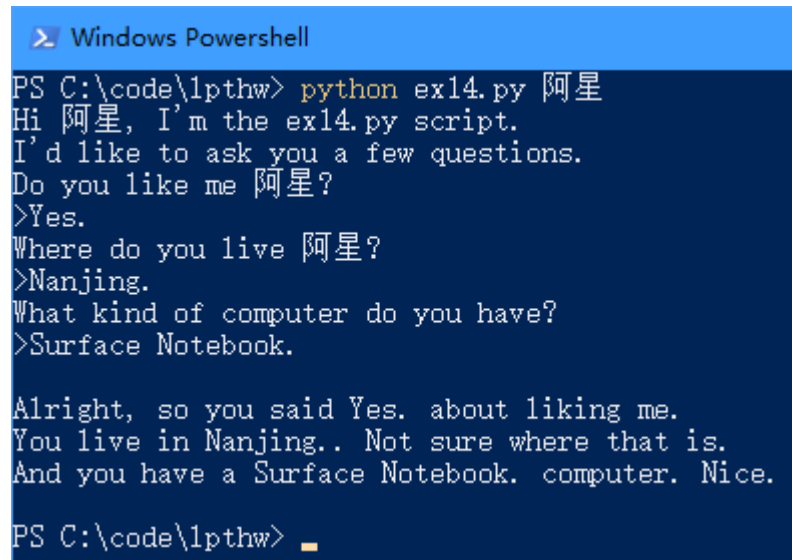
script, user_name = argv
prompt = '>'

print(f"Hi {user_name}, I'm the {script} script.")
print("I'd like to ask you a few questions.")
print(f"Do you like me {user_name}?")
likes = input(prompt)

print(f"Where do you live {user_name}?")
lives = input(prompt)

print("What kind of computer do you have?")
computer = input(prompt)

print(f"""
Alright, so you said {likes} about liking me.
You live in {lives}. Not sure where that is.
And you have a {computer} computer. Nice.
""")
```



The screenshot shows a Windows PowerShell window with a blue title bar. The command prompt is at C:\code\lpthw. The user runs 'python ex14.py 阿星'. The script outputs a greeting and asks three questions. The user responds with '>Yes.', '>Nanjing.', and '>Surface Notebook.'. The script then prints a summary of the user's answers.

```
PS C:\code\lpthw> python ex14.py 阿星
Hi 阿星, I'm the ex14.py script.
I'd like to ask you a few questions.
Do you like me 阿星?
>Yes.
Where do you live 阿星?
>Nanjing.
What kind of computer do you have?
>Surface Notebook.

Alright, so you said Yes. about liking me.
You live in Nanjing.. Not sure where that is.
And you have a Surface Notebook. computer. Nice.

PS C:\code\lpthw> _
```

习题15. 读取文件

ex15.py

```
from sys import argv
```

```

script, filename = argv

txt = open(filename)

print(f"Here's your file {filename}:")
print(txt.read())

txt.close()

print("Type the filename again:")
file_again = input(">")

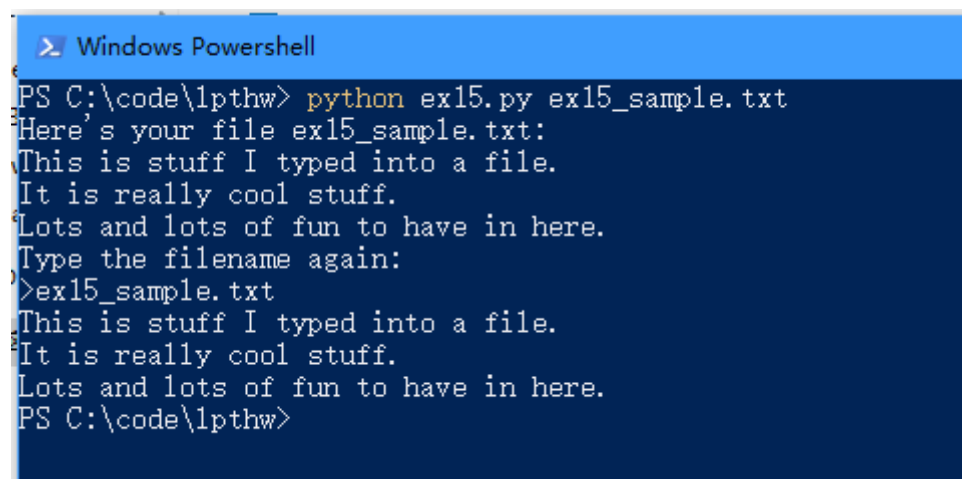
txt_again = open(file_again)

print(txt_again.read())

txt_again.close()

```

执行结果：



```

Windows Powershell
PS C:\code\lpthw> python ex15.py ex15_sample.txt
Here's your file ex15_sample.txt:
This is stuff I typed into a file.
It is really cool stuff.
Lots and lots of fun to have in here.
Type the filename again:
>ex15_sample.txt
This is stuff I typed into a file.
It is really cool stuff.
Lots and lots of fun to have in here.
PS C:\code\lpthw>

```

`txt.read()`：嘿，`txt`！执行你的 `read` 命令，无需任何参数！
命令实际上应该称为函数 `function` 或者方法 `method`

`txt = open(filename)` 返回的不是文件内容，而是文件对象 `file object`

多次打开同一文件是被允许的！

`from sys import argv` 其中 `sys` 是一个软件包，从 `sys` 包取出 `argv` 模块

在实际写程序的时候，不要把文件名 `ex15_sample.txt` “写死” `hardcode`
最好用 `input` 和 `argv`，询问用户需要打开哪个文件

习题16. 读写文件

```
from sys import argv

script, filename = argv

print(f"We're going to erase {filename}.")
print("If you don't want that, hit CTRL-C (^C).")
print("If you do want that, hit RETURN.")

input("?")

print("Opening the file...")
target = open(filename, 'w')

print("Truncating the file. Goodbye!")

target.truncate()

print("Now I'm going to ask you for three lines.")

line1 = input("line 1: ")
line2 = input("line 2: ")
line3 = input("line 3: ")

print("I'm going to write these to the file.")

target.write(line1)
target.write('\n')
target.write(line2)
target.write('\n')
target.write(line3)
target.write('\n')

print("And finally, we close it.")
target.close()
```

运行结果：

```
Windows Powershell
PS C:\code\lpthw> python ex16.py test.txt
We're going to erase test.txt.
If you don't want that, hit CTRL-C (^C).
If you do want that, hit RETURN.
?
Opening the file...
Truncating the file. Goodbye!
Now I'm going to ask you for three lines.
line 1: Mary had a little lamb
line 2: It's fleece was white as snow
line 3: It was also tasty
I'm going to write these to the file.
And finally, we close it.
PS C:\code\lpthw> cat test.txt
Mary had a little lamb
It's fleece was white as snow
It was also tasty
PS C:\code\lpthw> 
```

- `close()` 关闭文件
- `read()` 读取文件内容，可以把值赋给一个变量
- `readline()` 读取文本文件的一行
- `truncate()` 清空文件
- `write('stuff')` 将 'stuff' 写入文件
- `seek(0)` 将读写位置移动到文件开头

文件的访问模式：

写入 `write` 模式 `w`

读取 `read` 模式 `r`

追加 `append` 模式 `a`

访问模式的修饰符

最重要的是 `+`：同时读写

`w+` 读写，如果文件不存在，则新建，即新建读写

`r+` 读写，如果文件不存在，则报错，即报错读写

`a+` 追加读写，如果文件不存在，则创建

只写 `open(filename)`，默认以 `r` 只读模式打开

习题17. 更多文件操作

`ex17.py`

```
from sys import argv
from os.path import exists

script, from_file, to_file = argv

print(f"Coping from {from_file} to {to_file}")
```



```

# we could do these two on one line, how?
# indata = open(from_file).read()
in_file = open(from_file)
indata = in_file.read()

print(f"The input file is {len(indata)} bytes long")

print(f"Does the output file exist? {exists(to_file)}")
print("Ready, hit RETURN to continue, CTRL-C to abort")
input()

out_file = open(to_file, 'w')
out_file.write(indata)

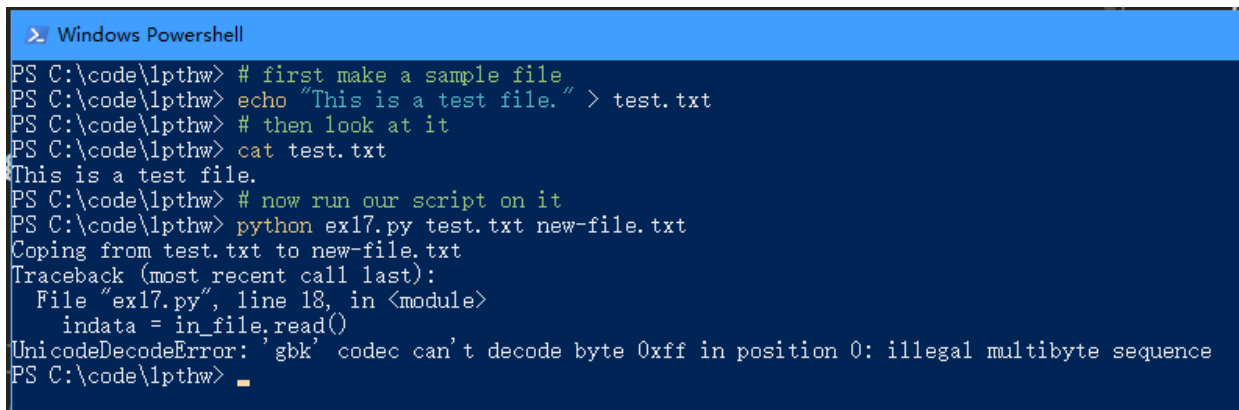
print("Alright, all done.")

out_file.close()
in_file.close()

```

`os.path.exists()` 命令，接收文件名字符串作为参数，如果存在返回 `True`，否则返回 `False`

报错！

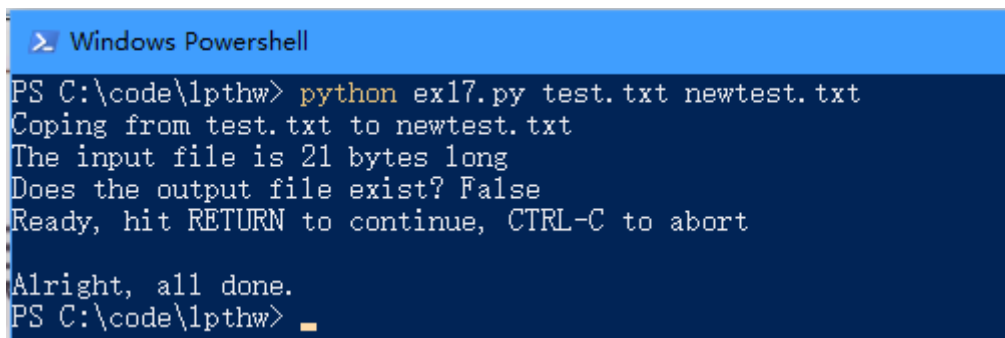


```

> Windows Powershell
PS C:\code\lpthw> # first make a sample file
PS C:\code\lpthw> echo "This is a test file." > test.txt
PS C:\code\lpthw> # then look at it
PS C:\code\lpthw> cat test.txt
This is a test file.
PS C:\code\lpthw> # now run our script on it
PS C:\code\lpthw> python ex17.py test.txt new-file.txt
Coping from test.txt to new-file.txt
Traceback (most recent call last):
  File "ex17.py", line 18, in <module>
    indata = in_file.read()
UnicodeDecodeError: 'gbk' codec can't decode byte 0xff in position 0: illegal multibyte sequence
PS C:\code\lpthw>

```

如果事先建立好 `test.txt` 文件，并选择 `utf-8` 编码另存文件，则不会出错



```

> Windows Powershell
PS C:\code\lpthw> python ex17.py test.txt newtest.txt
Coping from test.txt to newtest.txt
The input file is 21 bytes long
Does the output file exist? False
Ready, hit RETURN to continue, CTRL-C to abort

Alright, all done.
PS C:\code\lpthw>

```

`cat` 命令是 `concatenate` 拼接的意思，把两个文件拼接到一起，实际上最大的用途是打印文件内容到屏幕。通过 `man cat` 了解更多信息。

把所有脚本的命令写到一行里，用 `;` 实现。

`indata = open(from_file).read()`，一旦 `read()` 运行，文件就会被 `Python` 关掉。因为 `read()` 执行完毕后，文件对象没有引用，所以被垃圾回收器回收，不用再关闭文件。

【!】问题解决

问题源头还是在于编码

在 `PS` 中建立文件

```
> Windows Powershell
PS C:\code\lpthw> echo "This is a test file." > test.txt
PS C:\code\lpthw> cat test.txt
This is a test file.
PS C:\code\lpthw> _
```

用 `Ubuntu` 查看文件编码格式

```
jpch89@JPCH89-SURFACE:/mnt/c/code/lpthw$ file test.txt
test.txt: Little-endian UTF-16 Unicode text, with CR line terminators
jpch89@JPCH89-SURFACE:/mnt/c/code/lpthw$ _
```

把 `in_file = open(from_file)` 改成 `in_file = open(from_file, encoding = 'utf-16')` 成功！

```
PS C:\code\lpthw> python ex17.py test.txt new-file.txt
Copying from test.txt to new-file.txt
The input file is 21 bytes long
Does the output file exist? False
Ready, hit RETURN to continue, CTRL-C to abort

Alright, all done.
PS C:\code\lpthw> _
```

有趣的是，用 `file new-file.txt` 查看新复制的文件格式：

```
jpch89@JPCH89-SURFACE:/mnt/c/code/lpthw$ file new-file.txt
new-file.txt: ASCII text, with CRLF line terminators
```

是 `ASCII`

习题18. 命名、变量、代码和函数

`ex18.py`

```
# this one is like your scripts with argv
def print_two(*args):
```

```

arg1, arg2 = args
print(f"arg1: {arg1}, arg2: {arg2}")

# ok, that *args is actually pointless, we can just do this
def print_two_again(arg1, arg2):
    print(f"arg1: {arg1}, arg2: {arg2}")

# this just takes one argument
def print_one(arg1):
    print(f"arg1: {arg1}")

# this one takes on arguments
def print_none():
    print("I got nothin'.")

print_two("Zed", "Shaw")
print_two_again("Zed", "Shaw")
print_one("First!")
print_none()

```

函数 function

Python 使用 `def` 创建函数，也就是定义 `define` 的意思
下面是函数名：最好体现函数的功能

运行函数，调用函数和使用函数是一个意思。

函数名和变量名的命名规则一样，仅以字母、数字以及下划线组成，且不以数字开始。

*args 的意思

让 Python 把函数的所有参数都接收进来，然后放到名叫 `args` 的列表中去。

习题19. 函数和变量

ex19.py

```

def cheese_and_crackers(cheese_count, boxes_of_crackers):
    print(f"You have {cheese_count} cheeses!")
    print(f"You have {boxes_of_crackers} boxes of crackers!")
    print("Man that's enough for a party!")
    print("Get a blanket.\n")

print("We can just give the function numbers directly:")
cheese_and_crackers(20, 30)

print("OR, we can use variable from our script:")
amount_of_cheese = 10
amount_of_crackers = 50

```

```
cheese_and_crackers(amount_of_cheese, amount_of_crackers)

print("We can even do math inside too:")
cheese_and_crackers(10 + 20, 5 + 6)

print("And we can combine the two, variable and math:")
cheese_and_crackers(amount_of_cheese + 100, amount_of_crackers + 1000)
```

习题20. 函数和文件

ex20.py

```
from sys import argv
script, input_file = argv

def print_all(f):
    print(f.read())

def rewind(f):
    f.seek(0)

def print_a_line(line_count, f):
    print(line_count, f.readline())

current_file = open(input_file)

print("First let's print the whole file:\n")

print_all(current_file)

print("Now let's rewind, kind of like a tape.")

rewind(current_file)

print("Let's print three lines:")

current_line = 1
print_a_line(current_line, current_file)

# current_line = current_line + 1
current_file += 1
print_a_line(current_line, current_file)

# current_line = current_line + 1
current_file += 1
print_a_line(current_line, current_file)
```

`readline()` 怎么知道每一行在哪里？

`readline()` 扫描文件的每一个字节，直到找到一个 `\n` 为止，然后停止读取文件，并且返回此前发现的文件内容。文件 `f` 会记录每次调用 `readline()` 后的读取位置，这样它就可以在下次被调用时读取接下来的一行了。

为什么文件里会有间隔空行？

`readline()` 函数返回的内容本身包含 `\n`，`print` 打印时会添加 `\n`，这样多出来一个空行，可以在 `print` 加一个参数 `end=""`，这样就不会多打印 `\n` 了。

习题21. 函数可以返回某些东西

ex21.py

```
def add(a, b):
    print(f"ADDING {a} + {b}")
    return a + b

def subtract(a, b):
    print(f"SUBTRACTING {a} - {b}")
    return a - b

def multiply(a, b):
    print(f"MULTIPLYING {a} * {b}")
    return a * b

def divide(a, b):
    print(f"DIVIDING {a} / {b}")
    return a / b

print("Let's do some math with just functions!")

age = add(30, 5)
height = subtract(78, 4)
weight = multiply(90, 2)
iq = divide(100, 2)

print(f"Age: {age}, Height: {height}, Weight: {weight}, IQ: {iq}")

# A puzzle for the extra credit, type it in anyway.
print("Here is a puzzle.")

what = add(age, subtract(height, multiply(weight, divide(iq, 2))))

print("That becomes: ", what, "Can you do it by hand?")
```

使用 `float(input())` 来输入浮点数

习题22. 到现在为止你学到了什么

习题23. 字符串、字节串和字符编码

【!】字节串这个说法很好！

ex23.py

现代计算机极其复杂，不过简单讲，它们根本上就是一个巨大的开关阵列。
计算机用电来触发这些开关的开启或关闭。

`1` 表示开（有电、开启、接通），`0` 表示关（没电、关闭、切断）。

称这些 `0` 和 `1` 为**位 bit**。

用**字节 byte** 表示一个 `8` 位（`0` 和 `1`）的序列。

有了字节，确定了数字和字母之间的对应关系的约定，你就可以存储和展示文本了。

ASCII（**American Standard Code for Information Interchange**，美国信息交换标准代码）

```
>>> 0b1011010
90
>>> ord('Z')
90
>>> chr(90)
'Z'
```

ASCII 只能对英语和若干类似语言进行编码。

`1` 字节可以放 `256` 个数字，最多只能对应 `256` 个字符。

于是很多国家为各自的语言创造了不同的编码方式，但是很多编码都只能处理一种语言。

如果你要在一句泰语中插入一个英语的书籍名称，你就需要一个泰语的编码，一个英语的编码。

为了解决这个问题，有一群人发明了 **Unicode**，即通用编码 **universal encoding**。

你可以用 `32` 位编码一个 **Unicode** 字符，里面多余的空间我们用它们来存储“大便”和“微笑”之类的表情符。

但是 `32` 位是 `4` 字节，对于我们要编码的大部分类型的文本都是一种浪费。

我们也可以使用 `16` 位（`2` 字节），但对于大部分文本还是浪费。

所有把大部分常用字符用 `8` 位编码，如果需要编码更多的字符，就使用更大的数。

这一约定在 **Python** 里叫做 **UTF-8**。

在 **Python** 中，字符串 **string** 是 **UTF-8** 编码的字符序列，是显示和处理文本的基础。

bytes 则是 **Python** 用来存储 **UTF-8** 字符串的原始字节序列，用 `b''` 告诉 **Python** 你处理的是原始

字节串。

DBES 读作 deebess

decode bytes, encode strings

解码字节串：字节串转成字符串

编码字符串：字符串转成字节串

习题24. 更多的练习

ex24.py

一次性传入多个参数

***formula** 就是把一个元组当成多个参数传递进去

```
print("Let's practice everything.")
print('You\'d need to know \'bout escapes with \\ that do:')
print('\n newlines and \t tabs.')

poem = """
\tThe lovely world
with logic so firmly planted
cannot discern \n the needs of love
nor comprehend passion from intuition
and requires an explanation
\n\t\twhere there is none.
"""

print("-----")
print(poem)
print("-----")

five = 10 - 2 + 3 - 6
print(f"This should be five: {five}")

def secret_formula(started):
    jelly_beans = started * 500
    jars = jelly_beans / 1000
    crates = jars / 100
    return jelly_beans, jars, crates

start_point = 10000
beans, jars, crates = secret_formula(start_point)

# remember that this is another way to format a string
print("with a starting point of: {}".format(start_point))
# it's just like with an f"" string
```

```
print(f"We'd have {beans} beans, {jars} jars, and {crates} crates.")

start_point = start_point / 10

print("We can also do that this way:")
formula = secret_formula(start_point)
# this is an easy way to apply a list to a format string
print("We'd have {} beans, {} jars, and {} crates.".format(*formula))
```

习题25. 更多更多的练习

ex25.py

```
def break_words(stuff):
    """This function will break up words for us."""
    words = stuff.split(' ')
    return words

def sort_words(words):
    """Sorts the words."""
    return sorted(words)

def print_first_word(words):
    """Prints the first word after popping it off."""
    word = words.pop(0)
    print(word)

def print_last_word(words):
    """Prints the last word after popping it off."""
    word = words.pop(-1)
    print(word)

def sort_sentence(sentence):
    """Takes in a full sentence and returns the sorted words."""
    words = break_words(sentence)
    return sort_words(words)

def print_first_and_last(sentence):
    """Prints the first and last words of the sentence."""
    words = break_words(sentence)
    print_first_word(words)
    print_last_word(words)

def print_first_and_last_sorted(sentence):
    """Sorts the words then prints the first and last one."""
    words = sort_sentence(sentence)
```



```
print_first_word(words)
print_last_word(words)
```

首先用 `python ex25.py` 找到犯的错误，然后做练习。

```
PS C:\code\lpthw> python ex25.py
PS C:\code\lpthw> python
Python 3.6.6 |Anaconda custom (64-bit)| (default, Jun 28 2018, 11:27:44)
[MSC v.1900 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import ex25
>>> sentence = "All good things come to those who wait."
>>> words = ex25.break_words(sentence)
>>> words
['All', 'good', 'things', 'come', 'to', 'those', 'who', 'wait.']
>>> sorted_words = ex25.sort_words(words)
>>> sorted_words
['All', 'come', 'good', 'things', 'those', 'to', 'wait.', 'who']
>>> ex25.print_first_word(words)
All
>>> ex25.print_last_word(words)
wait.
>>> words
['good', 'things', 'come', 'to', 'those', 'who']
>>> ex25.print_first_word(sorted_words)
All
>>> ex25.print_last_word(sorted_words)
who
>>> sorted_words
['come', 'good', 'things', 'those', 'to', 'wait.']
>>> sorted_words = ex25.sort_sentence(sentence)
>>> sorted_words
['All', 'come', 'good', 'things', 'those', 'to', 'wait.', 'who']
>>> ex25.print_first_and_last(sentence)
All
wait.
>>> ex25.print_first_and_last_sorted(sentence)
All
who
```

得到模块帮助文档：`help(ex25)` 或者 `help(ex25.break_words)`

帮助文档就是定义函数时放在 `"""` 之间的字符串，这些特殊的字符串也被称作**文档注释**
documentation comment

总是要输入模块名很麻烦，可以 `from ex25 import *`

使用 `quit()` 退出 Python

Python List pop()方法



描述

pop() 函数用于移除列表中的一个元素（默认最后一个元素），并且返回该元素的值。

语法

pop()方法语法：

```
list.pop([index=-1])
```

参数

- obj -- 可选参数，要移除列表元素的索引值，不能超过列表总长度，默认为 index=-1，删除最后一个列表值。

返回值

该方法返回从列表中移除的元素对象。

习题26. 恭喜你，现在可以考试了！

```
import sys

print("How old are you?", end=' ')
age = input()
print("How tall are you?", end=' ')
height = input()
print("How much do you weigh?", end=' ')
weight = input()

print(f"So, you're {age} old, {height} tall and {weight} heavy.")

script, filename = sys.argv

txt = open(filename)

print(f"Here's your file {filename}:")
print(txt.read())

print("Type the filename again:")
file_again = input("> ")

txt_again = open(file_again)

print(txt_again.read())
```

```
print("Let's practice everything.")
print('You\'d need to know \'bout escapes with \\ that do \n newlines and
\t tabs.')
```

```
poem = """
\tThe lovely world
with logic so firmly planted
cannot discern \n the needs of love
nor comprehend passion from intuition
and requires an explanation
\n\t\twhere there is none.
"""
```

```
print("-----")
print(poem)
print("-----")
```

```
five = 10 - 2 + 3 - 6
print(f"This should be five: {five}")
```

```
def secret_formula(started):
    jelly_beans = started * 500
    jars = jelly_beans / 1000
    crates = jars / 100
    return jelly_beans, jars, crates
```

```
start_point = 10000
beans, jars, crates = secret_formula(start_point)
```

```
# remember that this is another way to format a string
print("With a starting point of: {}".format(start_point))
# it's just like with an f"" string
print(f"We'd have {beans} beans, {jars} jars, and {crates} crates.")
```

```
start_point = start_point / 10
```

```
print("We can also do that this way:")
formula = secret_formula(start_point)
# this is an easy way to apply a list to a format string
print("We'd have {} beans, {} jars, and {} crates.".format(*formula))
```

```
people = 20
cats = 30
dogs = 15
```

```
if people < cats:
    print("Too many cats! The world is doomed!")

if people < cats:
    print("Not many cats! The world is saved!")

if people < dogs:
    print("The world is drooled on!")

if people > dogs:
    print("The world is dry!")

dogs += 5

if people >= dogs:
    print("People are greater than or equal to dogs.")

if people <= dogs:
    print("People are less than or equal to dogs.")

if people == dogs:
    print("People are dogs.")
```

习题27. 记住逻辑关系

逻辑术语

and 与

or 或

not 非

!= 不等于

== 等于

>= 大于等于

<= 小于等于

True 真

False 假

真值表

【!】 在第83页，省略！

习题28. 布尔表达式练习

布尔逻辑表达式 `boolean logic expression`

```
>>> 'test' and 'test'
'test'
>>> 1 and 1
1
>>> False and 1
False
>>> True and 1
1
```

Python 和很多编程语言一样，都是给布尔表达式返回两个被操作对象中的一个，而非 `True` 或 `False`。

`!=` 和 `<>`，用前者！

有没有短路逻辑？

有的。任何以 `False` 开头的 `and` 语句都会直接处理成 `False`，不会继续检查后面的语句。任何包含 `True` 的 `or` 语句，只要处理到 `True`，就不会继续向下推算，而是直接返回 `True`。

习题29. `if` 语句

`ex29.py`

```
people = 20
cats = 30
dogs = 15

if people < cats:
    print("Too many cats! The world is doomed!")

if people > cats:
    print("Not many cats! The world is saved!")

if people < dogs:
    print("The world is drooled on!")

if people > dogs:
    print("The world is dry!")

dogs += 5

if people >= dogs:
```

```
print("People are greater than or equal to dogs.")

if people <= dogs:
    print("People are less than or equal to dogs.")

if people == dogs:
    print("People are dogs.")
```

习题30. else和if

`if` 语句为代码创建了一个分支，如果这个布尔表达式为真，就运行接下来的代码，否则就跳过这一段。

`if` 语句的下一行需要 4 个空格的缩进。`if` 行尾的冒号告诉 `Python` 接下来你要创建一个新的代码块，缩进告诉 `Python` 这些代码处于该代码块中。跟前面创建函数时的冒号是一回事。不缩进会报错，因为 `Python` 的规则里，只要一行以冒号：结尾，它接下来的内容就应该有缩进。

ex30.py

```
people = 30
cars = 40
trucks = 15

if cars > people:
    print("We should take the cars.")
elif cars < people:
    print("We should not take the cars.")
else:
    print("We can't decide.")

if trucks > cars:
    print("That's too many trucks.")
elif trucks < cars:
    print("Maybe we could take the trucks.")
else:
    print("We still can't decide.")

if people > trucks:
    print("Alright, let's just take the trucks.")
else:
    print("Fine, let's stay home then.")
```

如果多个 `elif` 块都是 `True`，`Python` 只会运行第一个为 `True` 的块。

习题31. 作出决定

ex31.py

```
print("""You enter a dark room with two doors.
Do you go through door #1 or door #2?""")

door = input("> ")

if door == "1":
    print("There's a giant bear here eating a cheese cake.")
    print("What do you do?")
    print("1. Take the cake.")
    print("2. Scream at the bear.")

    bear = input("> ")

    if bear == "1":
        print("The bear eats your face off. Good job!")
    elif bear == "2":
        print("The bear eats your leg off. Good job!")
    else:
        print(f"Well, doing {bear} is probably better.")
        print("Bear runs away.")

elif door == "2":
    print("You stare into the endless abyss at Cthulhu's retina.")
    print("1. Blueberries.")
    print("2. Yellow jacket clothespins.")
    print("3. Understanding revolvers yelling melodies.")

    insanity = input("> ")

    if insanity == "1" or insanity == "2":
        print("Your body survives powered by a mind of jello.")
        print("Good job!")
    else:
        print("The insanity rots your eyes into a pool of muck.")
        print("Good job!")

else:
    print("You stumble around and fall on a knife and die. Good job!")
```

可以用多个 `if-else` 来代替 `elif` 吗？

可以，但是 `Python` 需要检查每一处的 `if/else`，而不会像 `if/elif/else` 一样，检查到第一个 `True` 就停下来。

怎样判断一个数是否处于某个值域中？

- 经典语法 `1 <= x < 10`
- 或者 `x in range(1, 10)`

习题32. 循环和列表

列表 `list`，就是从头到尾按顺序存放东西的容器。

以左方括号 `[` 开头打开列表，然后写下你要放入列表的东西，用逗号隔开，最后用右方括号 `]` 表明列表结束。

`ex32.py`

```
the_count = [1, 2, 3, 4, 5]
fruits = ['apples', 'oranges', 'pears', 'apricots']
change = [1, 'pennies', 2, 'dimes', 3, 'quarters']

# this first kind of for-loop goes through a list
for number in the_count:
    print(f"This is count {number}")

# same as above
for fruit in fruits:
    print(f"A fruit of type: {fruit}")

# also we can go through mixed lists too
# notice we have to use {} since we don't know what's in it
for i in change:
    print(f"I got {i}")

# we can also build lists, first start with an empty one
elements = []

# then use the range function to do 0 to 5 counts
for i in range(0, 6):
    print(f"Adding {i} to the list.")
    # append is a function that lists understand
    elements.append(i)

# now we can print them out too
for i in elements:
    print(f"Element was: {i}")
```

可以创建二位列表，比如 `[[1, 2, 3], [4, 5, 6]]`

在 Ruby 语言里，列表和数组都叫做数组。

在 Python 中，又都叫做列表。

`range(1, 3)` 含首不含尾

`elements.append()` 在列表的尾部追加元素

习题33. while 循环

ex33.py

```
i = 0
numbers = []

while i < 6:
    print(f'At the top is {i}')
    numbers.append(i)

    i = i + 1
    print('Numbers now:', numbers)
    print(f'At the bottom i is {i}')

print('The numbers:')

for num in numbers:
    print(num)
```

只要循环语句中的布尔值为 `True`，`while` 循环就会不停地执行它下面的代码块。

如果你的一行以冒号结尾 `:`，就意味着接下来的内容是一个新的代码块，新的代码块是需要被缩进的。

`while` 循环有时会永远无法停止。

- 尽量少用 `while` 循环，大部分时候 `for` 循环是更好的选择
- 重复检查 `while` 语句，确定你测试的布尔表达式最终会变成 `False`
- 如果不确定，在 `while` 循环的开始和结尾打印出你要测试的值，看看它的变化。

`Ctrl + C` 终止程序

`for` 循环只能对一些东西的集合进行循环，`while` 循环可以对任何对象进行循环。

习题34. 访问列表的元素

序数 `ordinal number`，它们表示的是事物的顺序。

要抓取列表中的内容，列表的每一个元素都应该有一个地址，或者一个索引 `index`，最好的方式就是使用以 `0` 开头的索引，取元素会更容易。这类数被称为基数 `cardinal number`，它意味着你可以任意抓取元素，所以需要有一个 `0` 号元素。

把序数转换成基数的公式：基数 = 序数 - 1

序数等于有序，从 `1` 开始；基数等于随机选取，从 `0` 开始。

作者不建议阅读 `Dijkstra` 关于数的主题

习题35. 分支和函数

`ex35.py`

```
from sys import exit

def gold_room():
    print('This room is full of gold. How much do you take?')

    choice = input('> ')
    if '0' in choice or '1' in choice:
        how_much = int(choice)
    else:
        dead('Man, learn to type a number.')

    if how_much < 50:
        print("Nice, you're not greedy, you win!")
        exit(0)
    else:
        dead("You greedy bastard!")

def bear_room():
    print("There is a bear here.")
    print("The bear has a bunch of honey.")
    print("The fat bear is in front of another door.")
    print("How are you going to move the bear?")
    bear_moved = False

    while True:
        choice = input('> ')

        if choice == "take honey":
```

```

        dead("The bear looks at you then slaps your face off.")
    elif choice == "taunt bear" and not bear_moved:
        print('The bear has moved from the door.')
        print('You can go through it now.')
        bear_moved = True
    elif choice == 'taunt bear' and bear_moved:
        dead('The bear gets pissed off and chews your leg off.')
    elif choice == 'open door' and bear_moved:
        gold_room()
    else:
        print('I got no idea what that means.')

def cthulhu_room():
    print('Here you see the great evil Cthulhu.')
    print('He, it, whatever stares at you and you go insane.')
    print('Do you flee for your life or eat your head?')

    choice = input('> ')

    if 'flee' in choice:
        start()
    elif 'head' in choice:
        dead('Well that was tasty!')
    else:
        cthulhu_room()

def dead(why):
    print(why, 'Good job!')
    exit(0)

def start():
    print('You are in a dark room.')
    print('There is a door to your right and left.')
    print('Which one do you take?')

    choice = input('> ')

    if choice == 'left':
        bear_room()
    elif choice == 'right':
        cthulhu_room()
    else:
        dead('You stumble around the room until you starve.')

start()

```

`exit(0)` 可以终止某个程序，其中的数字参数表示是否是遇到错误而终止的。

`0` 表示没有错误，`1` 表示发生错误，可以用其他数字来区别错误。

习题36. 设计和调试

调试程序的技巧是使用 `print` 在各个想要检查的关键点将变量打印出来，从而检查那里是否有错。让程序一部分一部分地运行起来。不要等写了一大堆代码文件后才去运行它们，写一点儿，运行一点儿，再修改一点儿。

写软件最好的办法：

1. 罗列待办任务。
2. 挑出最简单的任务。
3. 在源代码文件中写下注释，作为你完成任务代码的指南。
4. 在注释下面写一些代码。
5. 快速运行你的代码，看它是否工作。
6. 循环“写代码，运行代码进行测试，修正代码”的过程
7. 从任务列表中划掉这条任务，挑出下一个最简单的任务，重复上述步骤。
8. 随时更新任务列表，添加新的任务，删除不必要的任务。

习题37. 复习各种符号

关键字

`and` 逻辑与
`as` `with-as` 语句的一部分
`assert` 断言（确保）某东西为真
`break` 立即停止循环
`class` 定义类
`continue` 停止当前循环的后续步骤，再做一次循环
`def` 定义函数
`del` 从字典中删除
`elif` `else if` 条件
`else` `else` 条件
`except` 如果发生异常，运行此处代码
`exec` 将字符串作为 `Python` 脚本运行
`finally` 不管是否发生异常，都运行此处代码
`for` 针对物件集合执行循环
`from` 从模块中导入特定部分
`global` 声明全局变量
`if` `if` 条件
`import` 将模块导入当前文件以供使用
`in` `for` 循环的一部分，或者是 `x` 是否在 `y` 中的条件判断
`is` 类似于 `==`，判断是否一样

实际上和 `==` 不一样！

`lamda` 创建短匿名函数
`not` 逻辑非
`or` 逻辑或
`pass` 代表空代码块
`print` 打印字符串
`raise` 出错后引发异常
`return` 返回值并退出函数
`try` 尝试执行代码，出错后转到 `except`
`while` `while` 循环
`with` 将表达式作为一个变量，然后执行代码块
`yield` 暂停函数，返回到调用函数的代码中

数据类型

`True` 布尔值“真”
`False` 布尔值“假”
`None` 表示“不存在”或者“没有值”
`bytes` 字节串存储，可能是文本、`PNG` 图片、文件等
`strings` 存储文本信息
`numbers` 存储整数
`floats` 存储小数
`lists` 存储列表
`dicts` 存储键-值映射

字符串转义序列

`\\` 反斜杠
`\'` 单引号
`\"` 双引号
`\a` 响铃
`\b` 退格符
`\f` 表单填充符
`\n` 换行符
`\r` 回车
`\t` 制表符
`\v` 垂直制表符

老式字符串格式

`Python 2` 用这些格式化字符串实现 `f` 字符串的功能
`%d` 十进制整数
`%i` 和 `%d` 一样
`%o` 八进制数 `"%o" % 1000 = '1750'`
`%u` 无符号整数
`%x` 小写十六进制数

`%X` 大写十六进制数

`%e` 指数表示, 小写 `e`

`%E` 指数表示, 大写 `E`

`%f` 浮点实数

`%F` 和 `%f` 一样

`%g` `%f` 和 `%e` 中较短的一种

`%G` 和 `%g` 一样, 但是是大写

`%c` 字符格式 `"%c" % 34 == ' '`

`%r` `Repr` 格式 (调试格式) `"%r" % int == "<type 'int'>"`

`%s` 字符串格式

`%%` 百分号本身

运算符

`+` 加

`-` 减

`*` 乘

`**` 幂

`/` 除

`//` 除后向下取整

地板除

`%` 字符串翻译, 或者求余数

`<` 小于

`>` 大于

`<=` 小于等于

`>=` 大于等于

`==` 等于

`!=` 不等于

`()` 括号

`[]` 方括号

`{}` 花括号

`@` 修饰器符

`,` 逗号

`:` 冒号

`.` 点

`=` 赋值

`;` 分号

`+=` 加后赋值

`-=` 减后赋值

`*=` 乘后赋值

`/=` 除后赋值

`//=` 除后舍余并赋值

`%=` 求余后赋值

`**=` 求幂后赋值

阅读代码

把需要理解的代码打印出来，一次打印几页就行了。

通读代码并做笔记。

不使用纸质打印时，可以用注释符号 `#` 在程序中加入笔记。

研究一下流程图 `flow chart`。

习题38. 列表的操作

列表的追加 `append` 操作

`mystuff.append('hello')` 相当于是 `append(mystuff, 'hello')`，传了两个参数

`ex38.py`

```
ten_things = 'Apples Oranges Crows Telephone Light Sugar'

print("Wait there are not 10 things in that list. Let's fix that.")

stuff = ten_things.split(' ')
more_stuff = ['Day', 'Night', 'Song', 'Frisbee', 'Corn', 'Banana',
              'Girl', 'Boy']

while len(stuff) != 10:
    next_one = more_stuff.pop()
    print('Adding: ', next_one)
    stuff.append(next_one)
    print(f'There are {len(stuff)} items now.')

print('There we go: ', stuff)

print("Let's do some things with stuff.")

print(stuff[1])
print(stuff[-1]) # whoa! fancy
print(stuff.pop())
print(' '.join(stuff)) # what? cool!
print('#'.join(stuff[3:5])) # super stellar!
```

数据结构是组织数据的方法。

习题39. 字典，可爱的字典

你可以用数值作为索引，来获取列表中的项。

字典所做的是，让你可以通过任何东西（不只是数值）找到元素。

并不准确，先这么理解

`del` 删除字典中的内容。

字典的关键理念是**映射**或者**关联**。

ex39.py

```
# create a mapping of state to abbreviation
states = {
    'Oregon': 'OR',
    'Florida': 'FL',
    'California': 'CA',
    'New York': 'NY',
    'Michigan': 'MI'
}

# create a basic set of states and some cities in them
cities = {
    'CA': 'San Francisco',
    'MI': 'Detroit',
    'FL': 'Jacksonville'
}

# add some more cities
cities['NY'] = 'New York'
cities['OR'] = 'Portland'

# print out some cities
print('-' * 10)
print('NY State has: ', cities['NY'])
print('OR State has: ', cities['OR'])

# print some states
print('-' * 10)
print("Michigan's abbreviation is: ", states['Michigan'])
print("Florida's abbreviation is: ", states['Florida'])

# do it by using the state then cities dict
print('-' * 10)
print("Michigan has: ", cities[states['Michigan']])
print("Florida has: ", cities[states['Florida']])
```



```

# print every state abbreviation
print('-' * 10)
for state, abbrev in list(states.items()):
    print(f"{state} is abbreviated {abbrev}")

# print every city in state
print('-' * 10)
for abbrev, city in list(cities.items()):
    print(f"{abbrev} has the city {city}")

# now do both at the same time
print('-' * 10)
for state, abbrev in list(states.items()):
    print(f"{state} state is abbreviated {abbrev}")
    print(f"and has city {cities[abbrev]}")

print('-' * 10)
# safely get a abbreviation by state that might not be there
state = states.get('Texas')

if not state:
    print('Sorry, no Texas.')

# get a city with a default value
city = cities.get('TX', 'Does Not Exit')
print(f"The city for the state 'TX' is: {city}")

```

有序字典 `collections.OrderedDict`

`OrderedDict` 的 `Key` 会按照插入的顺序排列，不是 `Key` 本身排序

```

from collections import OrderedDict
od = OrderedDict([('a', 1), ('b', 2), ('c', 3)])

```

习题40. 模块、类和对象

模块和字典差不多：都是从 `Y` 获取 `X`

1. 模块是包含函数和变量的 `Python` 文件。
2. 可以导入 `import` 这个文件。
3. 然后可以使用 `.` 操作符访问模块中的函数和变量。

`Python` 里边有一个非常常用的模式：

1. 拿一个类似“键=值”风格的容器；

2. 通过“键”的名称获取其中的“值”。

类和模块差不多

通过类，可以把一组函数和数据放到一个容器中，从而用 `.` 运算符访问它们。

使用类而非模块的原因如下：你可以用类创建出很多互不干涉的对象，而对于模块来说，一次导入之后，整个程序里就只有这么一份内容。

对象和 `import` 差不多

模块有一个导入 `import` 的概念，对于类，有一个实例化 `instantiate` 的概念。

当你将一个类实例化以后，你得到的就叫对象 `object`。

将类实例化的方法就是像调用函数一样地调用一个类。

事实上，类和对象与模块是完全不同的东西。

【!】这里作者讲的一点都不好。不要看他说的！

ex40.py

```
class Song(object):

    def __init__(self, lyrics):
        self.lyrics = lyrics

    def sing_me_a_song(self):
        for line in self.lyrics:
            print(line)

happy_bday = Song(["Happy birthday to you",
                  "I don't want to get sued",
                  "So I'll stop right there"])

bulls_on_parade = Song(["They rally around the family",
                       "With pockets full of shells"])

happy_bday.sing_me_a_song()

bulls_on_parade.sing_me_a_song()
```

为什么创建 `__init__` 或者别的类函数时需要多加一个 `self` 变量？
可以区别实例的属性 `self.age` 和局部变量 `age`。

习题41. 学习面向对象术语

类 `class`

对象 `object`

实例 `instance`

`def`

`self`

继承 `inheritance`：指一个类可以继承另一个类的特性，和父子关系类似。

组合 `composition`：指一个类可以将别的类作为它的部件构建起来，有点儿像车子和车轮的关系。

属性 `attribute`

是什么 `is-a`

有什么 `has-a`

`ex41.py`

`str` 转换成字符串，可以指定编码格式 `encoding`

`random.shuffle()` 用法

Python shuffle() 函数



描述

`shuffle()` 方法将序列的所有元素随机排序。

语法

以下是 `shuffle()` 方法的语法:

```
import random

random.shuffle (lst )
```

注意: `shuffle()`是不能直接访问的，需要导入 `random` 模块，然后通过 `random` 静态对象调用该方法。

参数

- `lst` -- 可以是一个序列或者元组。

返回值

返回随机排序后的序列。

`random.sample()` 用法

`random.sample(population, k)` 从 `population` 中随机选 `k` 个元素

字符串的 `count`

数有多少个子字符串：`string.count(substring)`

字符串的 `replace`

`string.replace('aaa', 'bbb', 1)` 把 `aaa` 换成 `bbb`，换 1 次

```
for sentence in snippet, phrase:
```

这一句的意思是让 `sentence = snippet`，然后再让 `sentence = phrase`
总共循环 2 次

习题42. 对象、类及从属关系

类 `class` 不是一个具体的东西，而是一个用来描述具有同类属性的实例的概括性的词汇。
对象 `object` 是类的实例 `instance`。

`is-a` 是什么：类之间的继承关系

`has-a` 有什么：类和属性、方法之间的关系

`ex42.py`

```
## Animal is-a object (yes, sort of confusing) look at the extra credit
class Animal(object):
    pass

## Dog is-a Animal.
class Dog(Animal):

    def __init__(self, name):
        ## Dog has-a name
        self.name = name

## Cat is-a Animal.
class Cat(Animal):

    def __init__(self, name):
        ## Cat has-a name.
        self.name = name

## Person is-a object.
class Person(object):

    def __init__(self, name):
        ## Person has-a name.
        self.name = name

        ## Person has-a pet of some kind
        self.pet = None

## Employee is-a Person.
class Employee(Person):

    def __init__(self, name, salary):
        ## ?? hmm what is this strange magic?
        super(Employee, self).__init__(name)
        ## Employee has-a salary.

## Fish is-a object.
class Fish(object):
    pass
```

```
## Salmon is-a Fish.
class Salmon(Fish):
    pass

## Halibut is-a Fish.
class Halibut(Fish):
    pass

## rover is-a Dog
rover = Dog('Rover')

## satan is-a Cat
satan = Cat('Satan')

## mary is-a Person
mary = Person('Mary')

## mary has-a pet
mary.pet = satan

## frank is-a Employee
frank = Employee('Frank', 12000)

## frank has-a pet
frank.pet = rover

## flipper is-a Fish
flipper = Fish()

## crouse is-a Salmon
crouse = Salmon()

## harry is-a Halibut
harry = Halibut()
```

类定义语法

```
class Name(object)
```

在 Python 2 中，`class Name(object)` 是新式类，`class Name` 是经典类，新式类比经典类多继承了 `object` 类的一些属性和方法。

在 Python 3 中，所有的类都隐式继承于 `object`，所以写不写 `object` 都是一样的，但是 Python 之禅说**显式优于隐式**，所以本书作者建议还是这么写。

多重继承 `multiple inheritance`

搜索 `python 3 super` , 理解 `super(Employee, self).__init__(name)`

Python `super()` 函数



描述

`super()` 函数是用于调用父类(超类)的一个方法。

`super` 是用来解决多重继承问题的, 直接用类名调用父类方法在使用单继承的时候没问题, 但是如果使用多继承, 会涉及到查找顺序 (MRO)、重复调用 (钻石继承) 等种种问题。

MRO 就是类的方法解析顺序表, 其实也就是继承父类方法时的顺序表。

语法

以下是 `super()` 方法的语法:

```
super(type[, object-or-type])
```

参数

- `type` -- 类。
- `object-or-type` -- 类, 一般是 `self`

Python3.x 和 Python2.x 的一个区别是: Python 3 可以使用直接使用 `super().xxx` 代替 `super(Class, self).xxx` :

MRO 方法解析顺序 (Method Resolution Order)

习题43. 基本的面向对象分析和设计

面向对象编程 **OOP**

面向对象分析 **OOA**

面向对象设计 **OOD**

自顶向下 **top-down**

从很抽象的概念入手, 逐渐细化, 直到概念变成具体的可用代码实现的东西

`ex42_classes.py`

```
from sys import exit
from random import randint
from textwrap import dedent

class Scene(object):

    def enter(self):
        print('This scene is not yet configured.')
        print('Subclass it and implement enter().')
        exit(1)

# Engine类
class Engine(object):
    # 初始化的时候接收一个地图对象
    def __init__(self, scene_map):
```

```

        self.scene_map = scene_map
# 开始游戏
def play(self):
    # 地图对象调用开场函数，赋值给当前场景
    # 地图对象的开场函数调用了下一场景函数，参数是初始场景
    # 返回值是初始场景类生成的初始场景对象CentralCorridor()
    current_scene = self.scene_map.opening_scene()
    # 最后一个场景是完结场景
    last_scene = self.scene_map.next_scene('finished')
    # 当前场景不是完结场景时
    while current_scene != last_scene:
        # 下一个场景名的值是当前场景调用enter()函数
        # 每个场景类都有enter()函数
        # enter()函数给出了一些场景提示和选择
        # 根据用户的选择，返回不同的字符串
        # 返回的字符串可能是当前场景，可能是'death'，也可能是下一场景
        next_scene_name = current_scene.enter()
        # 把下一场景名称的字符串给Map类的next_scene
        # 返回的是根据字符串在字典里面查到的对象
        # 然后重新进入循环
        current_scene = self.scene_map.next_scene(next_scene_name)

    # be sure to print out the last scene
    # 假如已经是'finished'场景
    # 那么还需要进入场景一次
    # 提示成功信息
    # 至此Engine.play()完成，游戏结束
    current_scene.enter()

```

```

class Death(Scene):
    quips = [
        "You died. You kinda suck at this.",
        "Your mom would be proud...if she were smarter.",
        "Such a luser.",
        "I have a small puppy that's better at this.",
        "You're worse than your Dad's jokes."
    ]

    def enter(self):
        print(Death.quips[randint(0, len(self.quips) - 1)])
        exit(1)

```

```

class CentralCorridor(Scene):

```

```

    def enter(self):
        print(dedent("""
The Gothons of Planet Percal #25 have invaded your ship and
destroyed your entire crew. You are the last surviving
member and your last mission is to get the neutron destruct
bomb from the Weapons Armory, put it in the bridge, and

```


blow the ship up after getting into an escape pod.

You're running down the central corridor to the Weapons Armory when a Gothon jumps out, red scaly skin, dark grimy teeth, and evil clown costume flowing around his hate filled body. He's blocking the door to the Armory and about to pull a weapon to blast you.

```
"""))
```

```
action = input('> ')
```

```
if action == 'shoot!':
```

```
    print(dedent("""
```

```
        Quick on the draw you yank out your blaster and fire  
        it at the Gothon. His clown costume is flowing and  
        moving around his body, which throws off your aim.
```

```
        Your laser hits his costume but misses him entirely.
```

```
        This completely ruins his brand new costume his mother  
        bought him, which makes him fly into an insane rage  
        and blast you repeatedly in the face until you are  
        dead. Then he eats you.
```

```
    """))
```

```
    return 'death'
```

```
elif action == 'dodge!':
```

```
    print(dedent("""
```

```
        Like a world class boxer you dodge, weave, slip and  
        slide right as the Gothon's blaster cranks a laser  
        past your head. In the middle of your artful dodge  
        your foot slips and you bang your head on the metal  
        wall and pass out. You wake up shortly after only to  
        die as the Gothon stomps on your head and eats you.
```

```
    """))
```

```
    return 'death'
```

```
elif action == 'tell a joke':
```

```
    print(dedent("""
```

```
        Lucky for you they made you learn Gothon insults in  
        the academy. You tell the one Gothon joke you know:  
        Lbhe zbgure vf fb sng, jura fur fvgf nebhaq gur ubhfr,  
        fur fvgf nebhaq gur ubhfr. The Gothon stops, tries  
        not to laugh, then busts out laughping and can't move.  
        While he's laughping you run up and shoot him square in  
        the head putting him down, then jump through the  
        Weapon Armory door.
```

```
    """))
```

```
    return 'laser_weapon_armory'
```

```
else:
```

```
    print('DOES NOT COMPUTE!')
```

```
    return 'central_corridor'
```

```
class LaserWeaponArmory(Scene):
```

```
    def enter(self):
```

```
        print(dedent("""
```

```
        You do a dive roll into the Weapon Armory, crouch and scan  
        the room for more Gothons that might be hiding. It's dead  
        quiet, too quiet. You stand up and run to the far side of  
        the room and find the neutron bomb in its container.
```

```
        There's a keypad lock on the box and you need the code to  
        get the bomb out. If you get the code wrong 10 times then  
        the lock closes forever and you can't get the bomb. The  
        code is 3 digits.
```

```
        """))
```

```
        code = f"{randint(1, 9)}{randint(1, 9)}{randint(1, 9)}"
```

```
        guess = input("[keypad]> ")
```

```
        guesses = 0
```

```
        while guess != code and guesses < 10:
```

```
            print("BZZZEDD!")
```

```
            guess += 1
```

```
            guess = input("[keypad]> ")
```

```
            if guess == code:
```

```
                break
```

```
        if guess == code:
```

```
            print(dedent("""
```

```
            The container clicks open and the seal breaks, letting  
            gas out. You grab the neutron bomb and run as fast as  
            you can to the bridge where you must place it in the  
            right spot.
```

```
            """))
```

```
            return 'the_bridge'
```

```
        else:
```

```
            print(dedent("""
```

```
            The lock buzzes one last time and then you hear a  
            sickening melting sound as the mechanism is fused  
            together. You decide to sit there, and finally the  
            Gothons blow up the ship from their ship and you die.
```

```
            """))
```

```
            return 'death'
```

```
class TheBridge(Scene):
```

```
    def enter(self):
```

```
        print(dedent("""
```

```
        You burst onto the Bridge with the neutron destruct bomb  
        under your arm and surprise 5 Gothons who are trying to  
        take control of the ship. Each of them has an even uglier  
        clown costume than the last. They haven't pulled their
```

```

weapons out yet, as they see the active bomb under your
arm and don't want to set it off.
"""))

action = input('> ')

if action == 'throw the bomb':
    print(dedent("""
    In a panic you throw the bomb at the group of Gothons
    and make a leap for the door. Right as you drop it a
    Gothon shoots you right in the back killing you. As
    you die you see another Gothon frantically try to
    disarm the bomb. You die knowing they will probably
    blow up when it goes off.
    """))
    return 'death'

elif action == 'slowly place the bomb':
    print(dedent("""
    You point your blaster at the bomb under your arm and
    the Gothons put their hands up and start to sweat.
    You inch backward to the door, open it, and then
    carefully place the bomb on the floor, pointing your
    blaster at it. You then jump back through the door,
    punch the close button and blast the lock so the
    Gothons can't get out. Now that the bomb is placed
    you run to the escape pod to get off this tin can.
    """))
    return 'escape_pod'
else:
    print('DOES NOT COMPUTE!')
    return 'the_bridge'

```

```

class EscapePod(Scene):

```

```

    def enter(self):
        print(dedent("""
        You rush through the ship desperately trying to make it to
        the escape pod before the whole ship explodes. It seems
        like hardly any Gothons are on the ship, so your run is
        clear of interference. You get to the chamber with the
        escape pods, and now need to pick one to take. Some of
        them could be damaged but you don't have time to look.
        There's 5 pods, which one do you take?
        """))

        good_pod = randint(1, 5)
        guess = input("[pod #]> ")

        if int(guess) != good_pod:
            print(dedent(f""

```

```

        You jump into pod {guess} and hit the eject button.
        The pod escapes out into the void of space, then
        implode as the hull ruptures, crushing your body
        into jam jelly.
        """))
    return 'death'

else:
    print(dedent(f"""
        You jump into pod {guess} and hit the eject button.
        The pod easily slides out into space heading to
        the planet below. As it flies to the planet, you look
        back and see your ship implode then explode like a
        bright star, taking out the Gothon ship at the same
        time. You won!
        """))
    return 'finished'

```

```

class Finished(Scene):

```

```

    def enter(self):
        print("You won! Good job.")
        return 'finished'

```

```

class Map(object):

```

```

    # 场景是一个字典
    # 键是场景名称，值是场景类生成的场景对象
    scenes = {
        'central_corridor': CentralCorridor(),
        'laser_weapon_armory': LaserWeaponArmory(),
        'the_bridge': TheBridge(),
        'escape_pod': EscapePod(),
        'death': Death(),
        'finished': Finished()
    }

```

```

    def __init__(self, start_scene):
        # 初始化场景
        # 这里接收的是'central_corridor'字符串
        self.start_scene = start_scene

```

```

    # 下一个场景，要传入下一个场景的名称
    def next_scene(self, scene_name):
        val = Map.scenes.get(scene_name)
        # 返回字典里的值
        # 该值是一个场景对象
        return val

```

```

    # 开场
    def opening_scene(self):

```

```
return self.next_scene(self.start_scene)
```

```
# 新建Map对象，初始场景为central_corridor  
a_map = Map('central_corridor')  
a_game = Engine(a_map)  
a_game.play()
```

有限状态机 `finite state machine`

习题44. 继承与组合

大部分使用继承 `inheritance` 的场合都可以用组合取代或简化，而多重继承则需要不惜一切地避免。

什么是继承

继承就是用来指明一个类的大部分或者全部功能都是从一个父类中获得的。

隐式继承

如果将函数放到基类中，那么所有的子类，将会自动获得这些函数功能。

`ex44a.py`

```
class Parent(object):  
  
    def implicit(self):  
        print('PARENT implicit()')  
  
class Child(Parent):  
    pass  
  
dad = Parent()  
son = Child()  
  
dad.implicit()  
son.implicit()
```

显式覆盖

子类中定义的同名函数取代父类的函数。

`ex44b.py`

```
class Parent(object):
```

```

def override(self):
    print('PARENT override()')

class Child(Parent):

    def override(self):
        print('CHILD override()')

dad = Parent()
son = Child()

dad.override()
son.override()

```

在运行前或运行后替换

这是显式覆盖的一个特例，用 `super()` 实现在父类定义的内容之前或者之后再修改行为。

ex44c.py

```

class Parent(object):

    def altered(self):
        print('PARENT altered()')

class Child(Parent):

    def altered(self):
        print('CHILD, BEFORE PARENT altered()')
        super(Child, self).altered()
        print('CHILD, AFTER PARENT altered()')

dad = Parent()
son = Child()

dad.altered()
son.altered()

```

3 种方式组合使用

ex44d.py

```

class Parent(object):

    def override(self):
        print('PARENT override()')

    def implicit(self):
        print('PARENT implicit()')

```

```

    def altered(self):
        print('PARENT altered()')

class Child(Parent):

    def override(self):
        print('CHILD override()')

    def altered(self):
        print('CHILD, BEFORE PARENT altered()')
        super(Child, self).altered()
        print('CHILD, AFTER PARENT altered()')

dad = Parent()
son = Child()

dad.implicit()
son.implicit()

dad.override()
son.override()

dad.altered()
son.altered()

```

多重继承

你定义的类继承了一个或多个类：

```

class SuperFun(Child, BadStuff):
    pass

```

Python 按照 **MRO** (**method resolution order**) 方法解析顺序查找函数，还用到了一个叫做 **C3** 的算法。

super() 和 **__init__** 搭配使用

组合

组合可以直接用别的类和模块，而非依赖于隐式继承。

ex44e.py

```

class Other(object):

    def override(self):
        print('OTHER override()')

    def implicit(self):
        print('OTHER implicit()')

```

```

def altered(self):
    print('OTHER altered()')

class Child(object):

    def __init__(self):
        self.other = Other()

    def implicit(self):
        self.other.implicit()

    def override(self):
        print('CHILD override()')

    def altered(self):
        print('CHILD, BEFORE OTHER altered()')
        self.other.altered()
        print('CHILD, AFTER OTHER altered()')

son = Child()

son.implicit()
son.override()
son.altered()

```

继承和组合的应用场合

继承与组合的问题说到底还是为了解决关于代码复用的问题。

继承通过创建一种让你在基类里隐含父类的功能的机制来解决这个问题。

而组合则是利用模块和别的类种的函数调用达到了相同的目的。

三大指导原则：

1. 不惜一切代价地避免多重继承，因为它太复杂以至于很不可靠。
2. 如果有一些代码会在不同位置和场合应用到，那就用组合来把它们做成模块。
3. 只有在代码的可复用部分之间有清楚的关联，可以通过一个单独的共性联系起来的时候，才使用继承。或者，现有代码或者别的不可抗拒因素所限非用继承不可，那就用吧。

巩固练习

阅读<http://www.python.org/dev/peps/pep-0008/>

对象是不是就是类的副本？

有的语言，比如 `JavaScript` 就是这样。

这样的语言叫做原型 `prototype` 语言。

这种语言里的类和对象除了用法以外没有多少不同。

不过在 `Python` 里类其实像是用来创建对象的模板。

习题45. 你来制作一款游戏

函数的风格

- 函数和方法都是一回事。
- 给类的一些函数命名时，是让它们去做事情，不如命名为一个动词，比如 `list` 的 `pop` 函数。
- 让函数保持简单小巧。
- 函数名全部小写，用 `_` 连接。
- 用一致的方式组织函数的参数。

类的风格

- 类名用驼峰式写法 `camel case`。
- `__init__` 不要做太多事情，会让类变得难以使用。
- 不要使用来自模块的变量或者全局变量。
- 不要一根筋地维持风格一致性。

代码风格

- 代码字符之间留一些空白，可以方便阅读。
- 如果一段代码你无法朗读出来，那么这段代码的可读性可能就有问题。
- 学着模仿别人的风格写 `Python` 程序，直到哪天你找到自己的风格为止。
- 不要把自己的风格太当回事。
- 如果你发现有人写代码的风格你很喜欢，那就模仿他们的风格。

好的注释

- 一定要写注释。
- 描述清楚为什么要这样实现，而不是要这样实现。
- 为函数写文档注释，用一两句话写用法。
- 注释要短小精悍，更改代码之后注释也要更改。

习题46. 项目骨架

骨架目录具备让项目运行起来的所有基本内容。

包含你的项目文件布局，自动测试代码，模块及安装脚本。

当建立新项目的时候，只要复制目录，改改目录的名字，再编辑里面的文件就行了。

macOS / Linux 配置

- 测试 `pip` 是否安装好：`pip list` 或者 `pip3.6 list`
可以忽略弃用 `deprecation` 警告
- 安装 `virtualenv`：`sudo pip3.6 install virtualenv`
- 测试是否安装好 `whereis virtualenv`

使用 `virtualenv` 来创建虚拟的 `Python` 安装环境，管理不同项目的包版本更加容易。
`mkdir ~/.venvs` 在用户家目录创建了叫 `.venvs` 的目录，用来存储你所有的虚拟环境。
`virtualenv --system-site-packages ~/.venvs/lpthw` 执行 `virtualenv`，包含系统站点包，创建了虚拟环境
`source ~/.venvs/lpthw/bin/activate` 激活虚拟环境

【NOTE】如果使用 `--no-site-packages` 那就安装了一个不带任何第三方包的纯净 `Python` 运行环境。

`which python` 查看 `Python` 版本和路径
`python` 进入 `Python`
`quit()` 退出

在虚拟环境中，安装测试框架 `nose`
`pip install nose`

Windows 10 配置

`cd ~` 到起始目录
`python` 检查 `Python` 版本
`pip list` 检查 `pip`，可以忽略弃用 `deprecation` 警告。
`pip install virtualenv` 安装 `virtualenv`
`mkdir .venvs` 创建虚拟环境文件夹
`virtualenv --system-site-packages .venvs/lpthw` 安装虚拟环境
`.\.venvs\lpthw\Scripts\activate` 激活虚拟环境
`pip install nose` 安装 `nose`

创建骨架项目目录

`mkdir projects` 在 `projects` 里面存储自己的各个项目
`cd projects/`
`mkdir skeleton` 新项目的基础目录
`cd skeleton`
`mkdir bin NAME tests docs`

`NAME` 是项目的主模块

使用骨架时，你可以将其重命名为你的项目的主模块名称

设置初始文件

Linux / macOS

`touch NAME/__init__.py`
`touch tests/__init__.py`

Windows PowerShell

`new-item -type file NAME/__init__.py`

```
new-item -type file tests/__init__.py
```

创建了空的 `Python` 模块目录，可以将代码放入其中。

然后建立 `setup.py`，这个文件在安装项目的时候会用到。

```
setup.py
```

```
try:
    from setuptools import setup
except ImportError:
    from distutils.core import setup

config = {
    'description': 'My Project',
    'author': 'jpch89',
    'url': 'URL to get it at.',
    'download_url': 'Where to download it.',
    'author_email': 'jpch89@outlook.com',
    'version': '0.1',
    'install_requires': ['nose'],
    'packages': ['NAME'],
    'scripts': [],
    'name': 'projectname'
}

setup(**config)
```

然后需要一个简单的测试专用的骨架文件，叫 `tests/NAME_tests.py`。

```
NAME_tests.py
```

```
def setup():
    print('SETUP!')

def teardown():
    print('TEAR DOWN!')

def test_basic():
    print('I RAN!')
```

测试你的配置

不能在 `tests/` 目录下运行 `nosetests`，要在它的上级目录。

```
(lpthw) C:\code\lpthw\ex46\projects\skeleton>nosetests
.
-----
Ran 1 test in 0.031s
```

OK

使用这个骨架

1. 复制这份骨架目录，把名字改成新项目的名字
2. 将 `NAME` 更名为你的项目的名字，或者你想给自己的根模块起的名字
3. 编辑 `setup.py`，让它包含新项目的相关信息。
4. 重命名 `tests/NAME_tests.py`，把 `NAME` 换成你的模块的名字。
5. 使用 `nosetests` 检查有无错误
6. 开始写代码。

小测验

1. 阅读如何使用 `nose`。
2. 阅读关于 `setup.py` 文件的文档。
3. 创建一个项目，在模块里写一些代码，并让这个模块可以运行。
4. 在 `bin` 目录下放一个可以运行的脚本，找材料学习怎样创建可以在系统下运行的 `Python` 脚本。
5. 在 `setup.py` 中加入 `bin` 里的脚本，这样你安装时就可以连 `bin` 的脚本也安装进去。
6. 使用 `setup.py` 安装你的模块，并确保安装的模块可以正常使用，最后使用 `pip` 将其卸载。

`setup.py` 的配置字典中应该放什么？

读读 `distutils` 的文档。

`bin/` 文件夹是一个标准的位置，用来存放在命令行上运行的脚本，但这不是存放模块的地方。

习题47. 自动化测试

更进一步，写出懂得你写的其他代码的代码。

编写测试用例

建议使用 `nosetest` 命令来运行测试，也可以用 `python3.6 ex47_tests.py` 来运行，但是不方便。

写一个小游戏，放在 `ex47/game.py` 中：

```
class Room(object):

    def __init__(self, name, description):
        self.name = name
        self.description = description
        self.paths = {}

    def go(self, direction):
```

```
return self.paths.get(direction, None)
```

```
def add_paths(self, paths):  
    self.paths.update(paths)
```

以 `test_` 开头的测试函数就是所谓的测试用例 `test case`。

最重要的函数是 `assert_equal`，保证了你设置的变量以及你在 `Room` 里设置的路径与你期望的相符。如果错误，`nosetests` 会打印出错消息。

`ex47_tests.py`

测试指南

1. 测试文件要放在 `tests/` 目录下，并命名为 `BLAH_test.py`，否则 `nosetests` 不会执行文件。还可以防止测试代码与别的代码混淆。
2. 为你创建的每一个模块写一个测试文件。
3. 测试用例（函数）保持简短。
4. 就算测试用例有点乱，也要让他们保持整洁，删除重复代码。
5. 别太把测试当回事。有时候最好重新设计代码。

巩固练习

1. 阅读与 `nosetest` 相关文档，并了解其他替代方案。
2. 了解一下 `Python` 的 `doc tests`，看看是否更喜欢这种方式。

习题48. 用户输入进阶

异常 `exception` 指的是运行某个函数时得到的错误。

你的函数遇到错误时，就会引发 `raise` 一个异常，然后你就要去处理 `handle` 这个异常。

处理异常用 `try` 和 `except` 这两个关键字：

把要“试着”运行的代码放到 `try` 块里，再将出错后要运行的代码放到 `except` 块里。

扫描器中使用 `ex48_convert.py` 来检测某个东西是不是数值。

`ex48_convert.py`

```
def convert_number(s):  
    try:  
        return int(s)  
    except ValueError:  
        return None
```

测试优先挑战

先写自动化测试，假装代码已经可以工作了，然后写代码让测试通过。

当你还没想好代码实现，但已经想好代码的使用方式时，这个方法尤为有用。

测试优先结合伪代码，得到简单的编程流程：

1. 写一段失败的测试代码
2. 写出测试所需的函数/模块/类的骨架
3. 在骨架中写注释，描述它的工作方式
4. 将注释换成代码，直到测试通过
5. 重复上述过程

lexicon_tests.py

```
from nose.tools import *
from ex48 import lexicon

def test_directions():
    assert_equal(lexicon.scan('north'), [('direction', 'north')])
    result = lexicon.scan('north south east')
    assert_equal(result, [('direction', 'north'),
                           ('direction', 'south'),
                           ('direction', 'east')])

def test_verbs():
    assert_equal(lexicon.scan('go'), [('verb', 'go')])
    result = lexicon.scan('go kill eat')
    assert_equal(result, [('verb', 'go'),
                           ('verb', 'kill'),
                           ('verb', 'eat')])

def test_stops():
    assert_equal(lexicon.scan('the'), [('stop', 'the')])
    result = lexicon.scan('the in of')
    assert_equal(result, [('stop', 'the'),
                           ('stop', 'in'),
                           ('stop', 'of')])

def test_nouns():
    assert_equal(lexicon.scan('bear'), [('noun', 'bear')])
    result = lexicon.scan('bear princess')
    assert_equal(result, [('noun', 'bear'),
                           ('noun', 'princess')])

def test_numbers():
    assert_equal(lexicon.scan('1234'), [('number', 1234)])
    result = lexicon.scan('3 91234')
    assert_equal(result, [('number', 3),
                           ('number', 91234)])
```

```
def test_errors():
    assert_equal(lexicon.scan('ASDFADFASDF'),
                  [('error', 'ASDFADFASDF')])
    result = lexicon.scan('bear IAS princess')
    assert_equal(result, [('noun', 'bear'),
                          ('error', 'IAS'),
                          ('noun', 'princess')])
```

下面是我写的 `lexicon.py`:

`lexicon.py`

```
glossary = {}

glossary['direction'] = ('north',
                        'south',
                        'east',
                        'west',
                        'down',
                        'up',
                        'left',
                        'right',
                        'back')

glossary['verb'] = ('go',
                   'stop',
                   'kill',
                   'eat')

glossary['stop'] = ('the',
                   'in',
                   'of',
                   'from',
                   'at',
                   'it')

glossary['noun'] = ('door',
                   'bear',
                   'princess',
                   'cabinet')

def convert_number(s):
    try:
        return int(s)
    except ValueError:
        return None
```

```

def scan(stuff):
    words = stuff.split()
    sentence = []
    for word in words:
        item = ()
        if convert_number(word):
            item = ('number', convert_number(word))
            sentence.append(item)

    for key, value in glossary.items():
        if word in value:
            item = (key, word)
            sentence.append(item)

    if item == ():
        sentence.append(('error', word))

    return sentence

```

习题49. 创建句子

parser.py

```

class ParserError(Exception):
    pass

class Sentence(object):

    def __init__(self, subject, verb, obj):
        # remember we take ('noun', 'princess') tuples and convert them
        self.subject = subject[1]
        self.verb = verb[1]
        self.object = obj[1]

    def peek(word_list):
        if word_list:
            word = word_list[0]
            return word[0]
        else:
            return None

    def match(word_list, expecting):
        if word_list:
            word = word_list.pop(0)

            if word[0] == expecting:

```



```

        return word
    else:
        return None
else:
    return None

def skip(word_list, word_type):
    while peek(word_list) == word_type:
        match(word_list, word_type)

def parse_verb(word_list):
    skip(word_list, 'stop')

    if peek(word_list) == 'verb':
        return match(word_list, 'verb')
    else:
        raise ParserError('Expected a verb next.')

def parse_object(word_list):
    skip(word_list, 'stop')
    next_word = peek(word_list)

    if next_word == 'noun':
        return match(word_list, 'noun')
    elif next_word == 'direction':
        return match(word_list, 'direction')
    else:
        raise ParserError('Expected a noun or direction next.')

def parse_subject(word_list):
    skip(word_list, 'stop')
    next_word = peek(word_list)

    if next_word == 'noun':
        return match(word_list, 'noun')
    elif next_word == 'verb':
        return ('noun', 'player')
    else:
        raise ParserError('Expected a verb next.')

def parse_sentence(word_list):
    subj = parse_subject(word_list)
    verb = parse_verb(word_list)
    obj = parse_object(word_list)

    return Sentence(subj, verb, obj)

```

尝试语法分析器

```
>>> from ex48.parser import *
```

```

>>> x = parse_sentence([('verb', 'run'), ('direction', 'north')])
>>> x.subject
'player'
>>> x.verb
'run'
>>> x.object
'north'
>>> x = parse_sentence([('noun', 'bear'), ('verb', 'eat'), ('stop',
'the'), ('noun', 'honey')])
>>> x.subject
'bear'
>>> x.verb
'eat'
>>> x.object
'honey'

```

应该测试的东西：

要包括异常测试，输入一个错误的句子会抛出异常。

用 `assert_raises` 来检查异常。

查看 `nose` 文档学习 `assert_raises` 的用法。

```
assert_raises(exception, callable, parameters)
```

下面是我写的测试代码：

```
tests/parser_tests.py
```

```

from nose.tools import *
from ex48 import parser

# test subject(noun) verb noun
def test_svn():
    x = parser.parse_sentence([('noun', 'Angel'), ('verb', 'kill'), ('noun', 'monster')])
    assert_equals(x.subject, 'Angel')
    assert_equals(x.verb, 'kill')
    assert_equals(x.object, 'monster')

# test subject(noun) verb direction
def test_svd():
    x = parser.parse_sentence([('stop', 'the'), ('noun', 'bear'), ('verb', 'run'), ('direction', 'south')])
    assert_equals(x.subject, 'bear')
    assert_equals(x.verb, 'run')
    assert_equals(x.object, 'south')

# test verb verb noun
def test_vvn():
    assert_raises(parser.ParserError, parser.parse_sentence, [('verb', 'kill'), ('verb', 'kill'), ('noun', 'monster')])

```

习题50. 你的第一个网站

安装 flask

它是一个 web 框架，所谓框架就是让某件事情做起来更容易的软件包。

Linux 和 macOS : `sudo pip install flask`

Windows : `pip install flask`

写一个简单的项目

```
cd ex50
mkdir gothonweb
cd gothonweb
mkdir bin gothonweb tests docs templates
touch gothonweb/__init__.py
touch tests/__init__.py
```

然后写一个 `app.py`

```
from flask import Flask
app = Flask(__name__)

@app.route('/')
def hello_world():
    greetings = 'Hello, World!'
    return greetings

if __name__ == '__main__':
    app.run()
```

在虚拟环境下运行 `python3.6 app.py`

得到显示信息后用浏览器打开 `http://localhost:5000/`

```
(lpthw) C:\code\lpthw\ex50\gothonweb>py app.py
* Serving Flask app "app" (lazy loading)
* Environment: production
  WARNING: Do not use the development server in a production environmen
t.
  Use a production WSGI server instead.
* Debug mode: off
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
127.0.0.1 - - [29/Jul/2018 17:25:01] "GET / HTTP/1.1" 200 -
```

```
127.0.0.1 - - [29/Jul/2018 17:25:01] "GET /favicon.ico HTTP/1.1" 404 -
```

下面是 `flask` 打印的日志 `log` 信息

发生了什么

1. 浏览器建立了到你的计算机的网络连接，你的计算机的名字叫做 `localhost`，这是一个标准称谓，表示的就是网络中你的这台计算机，不管实际名字是什么，都可以使用 `localhost` 来访问。网络端口是 `5000`。
2. 连接成功后，浏览器对 `app.py` 发出了 `HTTP` 请求 `request`，要求访问的 `URL` 为 `/`，通常这是网站的第一个 `URL`。
3. `flask` 找到了 `def hellow_world`，它就调用这个函数来处理请求。该函数运行后返回一个字符串，以供 `flask` 将其传递给浏览器。
4. 最后，`flask` 完成了对浏览器请求的处理，将响应 `response` 回传给浏览器，于是看到了现在的页面。

修正错误

删掉 `greetings`，刷新页面，得到 `Internal Server Error`，和一些日志。
还可以用调试模式运行 `flask`，但是由于在网上运行调试模式不安全，所以必须显式声明。

`Linux` 下：

```
export FLASK_DEBUG=1
```

`Windows` 下：

```
set FLASK_DEBUG=1
```

然后 `python app.py`，刷新浏览器之后可以看到更详细的页面，通过实时控制台还能获得更多信息。

警告

为调试模式带来风险的，正是 `flask` 的实时调试控制台及其改进的输出。
攻击者可以利用这些信息远程控制你的计算机。

创建基本的模板文件

下一步是将 `Hello World` 以较大的绿色字体显示出来。

写一个 `index.html`

```
<!DOCTYPE html>
<html lang="en">
```

```
<head>
  <meta charset="UTF-8">
  <title>Gothons Of Planet Percal #25</title>
</head>
<body>
{% if greeting %}
  I just wanted to say
  <em style="color: green; font-size: 2em;">{{ greeting }}</em>.
{% else %}
  <em>Hello</em>, world!
{% endif %}
</body>
</html>
```

修改 `app.py`

```
from flask import Flask
from flask import render_template

app = Flask(__name__)

@app.route('/')
def index():
    greeting = 'Hello, World!'
    return render_template('index.html', greeting = greeting)

if __name__ == '__main__':
    app.run()
```

刷新浏览器可以看到结果，也可以查看网页源代码，发现模板被渲染成了有效的 `HTML` 源代码。

模板工作原理

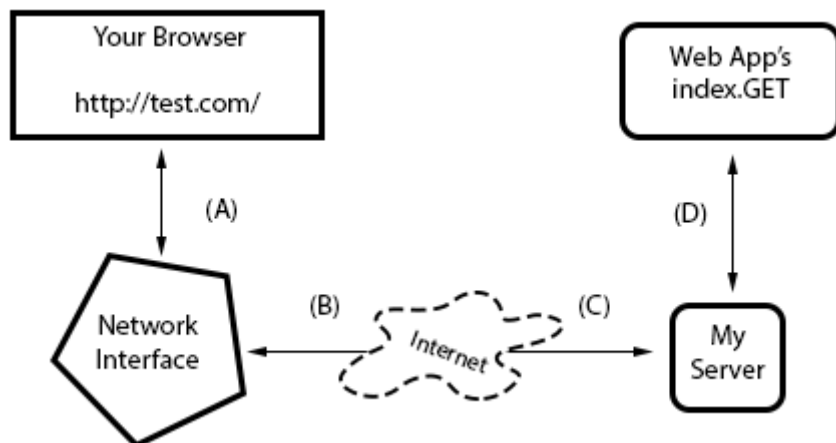
1. 导入 `render_template` 函数。
2. 它知道如何去 `templates/` 目录加载模板 `.html` 文件，因为这是 `flask` 的默认设置。
3. 在 `templates/index.html` 中，有些像正常 `HTML` 的内容，也有一些类似代码的东西放在特殊标记中。其中一个 `{% %}`，它里面放的是可执行代码，`if` 语句，`for` 循环之类，另一个是 `{{ }}`，里面放的是要转换为 `HTML` 输出文本的变量。模板的知识可以去看 `Jinja2` 的文档。

巩固练习

到 <http://flask.pocoo.org/docs/0.12/> 阅读文档，它跟 `flask` 是同一个项目。

习题51. 从浏览器获取输入

使用表单改进你的 **Web** 应用程序，并将用户的信息保存到他们的会话 **session** 中。



浏览器 browser：接收你在地址栏中输入的请求，然后使用该信息向该网址对应的服务器提出请求。

地址 address：类似 `http://test.com/` 一样的 **URL**。**URL** 是 **Uniform Resource Locator**，统一资源定位器。前面 `http` 指出你要使用的协议，这里是 **Hyper-Text Transport Protocol** 超文本传输协议。`ftp` 是文件传输协议 **File Transport Protocol**。

`http://test.com` 这部分是主机名 **hostname**，是一个便于人阅读和记忆的地址，主机名会被匹配到一串叫做 **IP** 地址的数字上面，类似网络中呼叫一台计算机的号码，通过号码可以访问这台计算机。最后，**URL** 中还可以跟随一个路径，比如 `http://test.com/book/` 中的 `/book/`，对应服务器上的某个文件或者某些资源，通过访问这样的网址，可以向服务器发出请求，然后获得这些资源。

连接 connection：一旦浏览器知道了你用的协议 `http`，你想联络的服务器

`http://test.com` 以及要从服务器上获得的资源，它就要去创建一个连接。在这个过程中，浏览器让操作系统（**operating system**，**OS**）打开计算机的一个端口 **port**，通常是 **80** 端口，端口准备好以后，操作系统会回传给你的程序一个类似文件的东西，它所做的事情就是通过网络传输和接收数据，让你的计算机和 `http://test.com` 这个网站所属的服务器之间实现数据交流。

请求 request：浏览器通过你提供的地址建立了连接，现在它需要从远程服务器要到它（或你）想要的资源。如果在 **URL** 结尾加了 `/book/`，那你想要的就是 `/book/` 对应的文件或资源，大部分的服务器会直接为你调用 `/book/index.html` 文件。为了得到服务器上的内容，你必须先向服务器发送一个请求。资源不一定要是文件，比如我们返回的就是 **Python** 代码生成的一些东西。

服务器 server：服务器指的是浏览器另一端连接的计算机。它知道如何回应浏览器请求的文件和资源。

响应 response：响应就是你的服务器回复你的请求而发回至浏览器的 **HTML**，它里面可能有 **CSS**、**JavaScript** 或者图像等内容。

表单的工作原理

更改 `app.py` :

```
from flask import Flask
from flask import render_template
from flask import request

app = Flask(__name__)

@app.route('/hello')
def index():
    name = request.args.get('name', 'Nobody')
    if name:
        greeting = f'Hello, {name}'
    else:
        greeting = 'Hello, world'

    return render_template('index.html', greeting = greeting)

if __name__ == '__main__':
    app.run()
```

访问<http://localhost:5000/hello>

浏览器显示 : I just wanted to say Hello, Nobody.

访问<http://localhost:5000/hello?name=Frank>

修改代码 :

```
from flask import Flask
from flask import render_template
from flask import request

app = Flask(__name__)

@app.route('/hello')
def index():
    name = request.args.get('name', 'Nobody')
    greet = request.args.get('greet', 'Hello')
    greeting = f'{greet}, {name}'

    return render_template('index.html', greeting = greeting)

if __name__ == '__main__':
    app.run()
```

使用网址访问 : <http://localhost:5000/hello?name=Frank&greet=Hola>

得到 : I just wanted to say Hola, Frank.

使用没有参数的网址访问：<http://localhost:5000/hello>

得到：I just wanted to say Hello, Nobody.

创建 HTML 表单

hello_form.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Sample Web Form</title>
</head>
<body>
<h1>Fill Out This Form</h1>
<form action="/hello" method="POST">
  A Greeting: <input type="text" name="greet">
  <br/>
  Your Name: <input type="text" name="name">
  <br/>
  <input type="submit">
</form>
</body>
</html>
```

修改 app.py：

```
from flask import Flask
from flask import render_template
from flask import request

app = Flask(__name__)

@app.route('/hello', methods = ['POST', 'GET'])
def index():
    greeting = 'Hello World'

    if request.method == 'POST':
        name = request.form['name']
        greet = request.form['greet']
        greeting = f'{greet}, {name}'
        return render_template('index.html', greeting=greeting)
    else:
        return render_template('hello_form.html')

if __name__ == '__main__':
    app.run()
```


访问地址<http://localhost:5000/hello>

输入内容，提交

得到的页面还是<http://localhost:5000/hello>，并没有包含你提交的参数。

`<form action="/hello" method="POST">` 告诉浏览器以下内容。

1. 从表单中的各个栏位手机用户输入的数据
2. 让浏览器使用一种 `POST` 类型的请求，将这些数据发送给服务器。这是另外一种浏览器请求，他会将 `URL` 表单栏位隐藏起来。
3. 将这个请求发送至 `/hello` `URL`，正如在 `action="/hello"` 中写的一样。

创建布局模板 `layout template`

修改 `index.html`

```
{% extends "layout.html"%}

{% block content %}

{% if greeting %}
    I just wanted to say
    <em style="color: green; font-size: 2em;">{{ greeting }}</em>.
{% else %}
    <em>Hello</em>, world!
{% endif %}
{% endblock %}
```

修改 `hello_form.html`

```
{% extends "layout.html" %}

{% block content %}

<h1>Fill Out This Form</h1>

<form action="/hello" method="POST">
    A Greeting: <input type="text" name="greet">
    <br/>
    Your Name: <input type="text" name="name">
    <br/>
    <input type="submit">
</form>

{% endblock %}
```

将每一个页面顶部和底部的反复用到的“样板代码”剥掉，单独放到 `templates/layout.html` 中。

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Gothons From Planet Percal #25</title>
</head>
<body>

{% block content %}

{% endblock %}

</body>
</html>
```

这个文件和普通的模板文件类似，只是其他模板的内容将传递给它，然后它会将其他模板的内容包裹起来。任何写在这里的内容都无需写在别的模板中。

别的 `HTML` 模板会被插入到 `{% block content %}` 部分。

在模板顶端写了 `{% extends "layout.html" %}`，所以 `flask` 知道使用这个 `layout.html` 作为布局。

为表单撰写自动测试代码

`tests/app_tests.py`

```
from nose.tools import *
from app import app

app.config['TESTING'] = True
web = app.test_client()

def test_index():
    rv = web.get('/', follow_redirects = True)
    assert_equal(rv.status_code, 404)

    rv = web.get('/hello', follow_redirects = True)
    assert_equal(rv.status_code, 200)
    assert_in(b'Fill Out This Form', rv.data)

    data = {'name': 'Zed', 'greet': 'Hola'}
    rv = web.post('/hello', follow_redirects = True, data = data)
    assert_in(b'Zed', rv.data)
    assert_in(b'Hola', rv.data)
```

习题52. 创建 Web 游戏

重构习题 43 中的游戏

重构：清理旧代码或者为旧代码添加新功能的过程。

复制 `ex47.py` 的代码到 `gothonweb/planisphere.py` 中

复制 `tests/ex47_tests.py` 的代码到 `tests/planisphere_tests.py` 中

运行 `nosetests`

重构 `planisphere.py`

```
class Room(object):

    def __init__(self, name, description):
        self.name = name
        self.description = description
        self.paths = {}

    def go(self, direction):
        return self.paths.get(direction, None)

    def add_paths(self, paths):
        self.paths.update(paths)

central_corridor = Room('Central Corridor',
    """
    The Goths of Planet Percal #25 have invaded your ship and
    destroyed your entire crew. You are the last surviving
    member and your last mission is to get the neutron destruct
    bomb from the Weapons Armory, put it in the bridge, and
    blow the ship up after getting into an escape pod.
    You're running down the central corridor to the Weapons
    Armory when a Gothon jumps out, red scaly skin, dark grimy
    teeth, and evil clown costume flowing around his hate
    filled body. He's blocking the door to the Armory and
    about to pull a weapon to blast you.
    """)

laser_weapon_armory = Room('Laser Weapon Armory',
    """
    Lucky for you they made you learn Gothon insults in
    the academy. You tell the one Gothon joke you know:
    Lbhe zbgure vf fb sng, jura fur fvgf nebhaq gur ubhfr,
    fur fvgf nebhaq gur ubhfr. The Gothon stops, tries
    not to laugh, then busts out laughping and can't move.
    While he's laughping you run up and shoot him square in
    the head putting him down, then jump through the
    Weapon Armory door.
    You do a dive roll into the Weapon Armory, crouch and scan
    the room for more Goths that might be hiding. It's dead
    quiet, too quiet. You stand up and run to the far side of
```

the room and find the neutron bomb in its container. There's a keypad lock on the box and you need the code to get the bomb out. If you get the code wrong 10 times then the lock closes forever and you can't get the bomb. The code is 3 digits.

```
""")
```

```
the_bridge = Room('The Bridge',  
""")
```

The container clicks open and the seal breaks, letting gas out. You grab the neutron bomb and run as fast as you can to the bridge where you must place it in the right spot.

You burst onto the Bridge with the neutron destruct bomb under your arm and surprise 5 Goths who are trying to take control of the ship. Each of them has an even uglier clown costume than the last. They haven't pulled their weapons out yet, as they see the active bomb under your arm and don't want to set it off.

```
""")
```

```
escape_pod = Room('Escape Pod',  
""")
```

You point your blaster at the bomb under your arm and the Goths put their hands up and start to sweat. You inch backward to the door, open it, and then carefully place the bomb on the floor, pointing your blaster at it. You then jump back through the door, punch the close button and blast the lock so the Goths can't get out. Now that the bomb is placed you run to the escape pod to get off this tin can. You rush through the ship desperately trying to make it to the escape pod before the whole ship explodes. It seems like hardly any Goths are on the ship, so your run is clear of interference. You get to the chamber with the escape pods, and now need to pick one to take. Some of them could be damaged but you don't have time to look. There's 5 pods, which one do you take?

```
""")
```

```
the_end_winner = Room('The End',  
""")
```

You jump into pod 2 and hit the eject button. The pod easily slides out into space heading to the planet below. As it flies to the planet, you look back and see your ship implode then explode like a bright star, taking out the Gothon ship at the same time. You won!

```
""")
```

```
the_end_loser = Room('The End',  
""")
```

You jump into a random pod and hit the eject button.
The pod escapes out into the void of space, then
implode as the hull ruptures, crushing your body
into jam jelly.

```
"""  
  
escape_pod.add_paths({  
    '2': the_end_winner,  
    '*': the_end_loser  
})  
  
generic_death = Room('death', 'You died.')  
  
the_bridge.add_paths({  
    'throw the bomb': generic_death,  
    'slowly place the bomb': escape_pod  
})  
  
laser_weapon_armory.add_paths({  
    '0132': the_bridge,  
    '*': generic_death  
})  
  
central_corridor.add_paths({  
    'shoot!': generic_death,  
    'dodge!': generic_death,  
    'tell a joke': laser_weapon_armory  
})  
  
START = 'central_corridor'  
  
def load_room(name):  
    """  
    There is a potential security problem here.  
    Who gets to set name? Can that expose a variable?  
    """  
    return globals().get(name)  
  
def name_room(room):  
    """  
    Same possible security problem. Can you trust room?  
    What's a better solution than this globals lookup?  
    """  
    for key, value in globals().items():  
        if value == room:  
            return key
```

注意：`globals()` 函数会以字典类型返回当前位置的全部全局变量。

重构 `planisphere_tests.py`

```

from nose.tools import *
from gothonweb.planisphere import *

def test_room():
    gold = Room('GoldRoom',
        """This room has gold in it you can grab. There's a
        door to the north.""")
    assert_equal(gold.name, 'GoldRoom')
    assert_equal(gold.paths, {})

def test_room_paths():
    center = Room('Center', 'Test room in the center.')
    north = Room('North', 'Test room in the north.')
    south = Room('South', 'Test room in the south.')

    center.add_paths({'north': north, 'south': south})
    assert_equal(center.go('north'), north)
    assert_equal(center.go('south'), south)

def test_map():
    start = Room('Start', 'You can go west and down a hole.')
    west = Room('Trees', 'There are trees here, you can go east.')
    down = Room('Dungeon', "It's dark down here, you can go up.")

    start.add_paths({'west': west, 'down': down})
    west.add_paths({'east': start})
    down.add_paths({'up': start})

    assert_equal(start.go('west'), west)
    assert_equal(start.go('west').go('east'), start)
    assert_equal(start.go('down').go('up'), start)

def test_gothon_game_map():
    start_room = load_room(START)
    assert_equal(start_room.go('shoot!'), generic_death)
    assert_equal(start_room.go('dodge!'), generic_death)

    room = start_room.go('tell a joke')
    assert_equal(room, laser_weapon_armory)

```

创建引擎

app.py

```

from flask import Flask, session, redirect, url_for, escape, request
from flask import render_template
from gothonweb import planisphere

app = Flask(__name__)

```

```

@app.route('/hello', methods = ['POST', 'GET'])
def index():
    # this is used to "setup" the session with starting values
    session['room_name'] = planisphere.START
    return redirect(url_for('game'))

@app.route('/game', methods = ['GET', 'POST'])
def game():
    room_name = session.get('room_name')

    if request.method == 'GET':
        if room_name:
            room = planisphere.load_room(room_name)
            return render_template('show_room.html', room = room)
        else:
            # why is there here? do you need it?
            # 在这里是为了防止用户不经过 /hello 直接进入 /game
            return render_template('you_died.html')
    else:
        action = request.form.get('action')

        if room_name and action:
            room = planisphere.load_room(room_name)
            next_room = room.go(action)

            if not next_room:
                session['room_name'] = planisphere.name_room(room)
            else:
                session['room_name'] = planisphere.name_room(next_room)
            return redirect(url_for('game'))

# YOU SHOULD CHANGE THIS IF YOU PUT ON THE INTERNET
app.secret_key = 'A0Zr98j/3yX R~XHH!jmN]LWX/,?RT'

if __name__ == '__main__':
    app.run()

```

删掉 `templates/hello_form.html` 和 `templates/index.html`
 然后重建 `show_room.html` 和 `you_died.html`

`show_room.html`

```

{% extends "layout.html" %}

{% block content %}

<h1> {{ room.name }} </h1>

<pre>
    {{ room.description}}

```

```
</pre>
```

```
{% if room.name in ["death", "The End"] %}}
    <p><a href="/">Play Again?</a></p>
{% else %}
    <p>
        <form action="/game" method="POST">
            - <input type="text" name="action"> <input type="SUBMIT">
        </form>

    </p>
{% endif %}

{% endblock %}
```

you_died.html

```
<h1>You Died!</h1>

<p>Looks like you bit the dust.</p>
<p><a href="/">Play Again</a></p>
```

期末考试

1. 修正 `bug`
2. 编写自动测试 `app_tests.py`
3. 美化 `HTML`
4. 网页登陆系统，让用户可以保存高分
5. 完成游戏地图
6. 帮助系统，查询房间里面可以做哪些事情
7. 创建多个地图，让玩家可以选择地图。这样引擎可以支持多个不同的游戏。
8. 利用习题 48 和习题 49 创建更好的输入处理器

常见问题回答

在游戏中使用了 `session` 会话，不能用 `nosetests` 测试。

阅读 `Flask` 测试文档，找到 `Other Testing Trick`，看看如何在测试中伪造会话。

接下来的路

- 《“笨办法”学 `Ruby`》
- `The Django Tutorial`
- `SciPy`
- `PyGame`
- `Pandas`
- `Natural Language Tool Kit`

- TensorFlow
- Requests
- ScraPy
- Kivy
- 《“笨办法”学 C 语言》

选择一个项目，通读它的文档和简易教程。

将文档中的代码自己录入一遍，并让他们正常运行。

读完之后试着写点东西出来，哪怕别人写过的也可以，只要做出点东西来就可以了。

怎样学习任何一种编程语言

1. 找到关于这种编程语言的书或者介绍性读物。
2. 通读这本书，把里面的代码都录入一遍并使其运行起来。
3. 一边读书一边写代码，同时做好笔记。
4. 使用这种编程语言实现一些你用另一种熟悉的编程语言做过的程序组件。
5. 阅读别人用这种编程语言编的代码，试着仿照他们的方式编写代码。

老程序员的建议

编程语言这东西并不重要，重要的是你用这些编程语言做的事情。

编程语言的真正目的：作为你的工具来做有趣的事情。

你最好的选择是将自己的编程技术作为自己的其他职业的秘密武器。

附录：命令行快速入门

简介：废话少说，命令行来也

编程语言是控制计算机的进阶方式，命令行则算是编程语言的小弟。

练习1. 准备工作

打开终端

macOS

- `command` + 空格
- 搜索栏输入 `Terminal`
- 点击黑盒子一样的程序
- `Ctrl` 点击 `dock` , 拉出菜单, 在打开的菜单中选择 `Options -- Keep In Dock`

Linux

在窗口管理器 `window manager` 里面搜索 `Shell` 或者 `Terminal` 即可

Windows

使用 `PowerShell` 替代 `cmd.exe`

- 开始菜单
- 搜索 `powershell`
- 回车

暂时不要用除了 `bash` 以外的 `shell` , 比如 `zsh` 。

Linux / macOS

- `pwd` 打印工作目录
- `hostname` 计算机在网络中的名称
- `mkdir` 创建目录
- `cd` 更改目录
- `ls` 列出目录的内容
- `rmdir` 删除目录
- `pushd` 推入目录
- `popd` 弹出目录
- `cp` 复制文件或目录
- `mv` 移动文件或目录
- `less` 逐页查看文件
- `cat` 打印整个文件
- `xargs` 执行参数
- `find` 寻找文件
- `grep` 在文件中查找内容
- `man` 阅读手册
- `apropos` 寻找恰当的手册页面
- `env` 查看你的环境
- `echo` 打印一些参数
- `export` 导出 / 设定一个新的环境变量
- `exit` 退出 `shell`
- `sudo` 成为超级用户 `root` , 危险命令 !

Windows

- `pwd` 打印工作目录
- `hostname` 计算机在网络中的名称
- `mkdir` 创建目录
- `cd` 更改目录

- `ls` 列出目录的内容
- `rmdir` 删除目录
- `pushd` 推送目录
- `popd` 弹出目录
- `cp` 复制文件或目录
- `robocopy` 更可靠的复制命令
- `mv` 移动文件或目录
- `more` 逐页查看文件
- `type` 打印整个文件
- `forfiles` 在一大堆文件上面运行一条命令。
- `dir -r` 寻找文件
- `select-string` 在文件中查找内容
- `help` 阅读手册
- `helpctr` 寻找恰当的手册页面
- `echo` 打印一些参数
- `set` 导出 / 设定一个新的环境变量
- `exit` 退出 `shell`
- `runas` 称为超级用户 `root` , 危险命令 !

练习2. 路径、文件夹和目录 (`pwd`)

`pwd` 打印工作目录

`print working directory`

练习3. 如果你迷失了

`pwd`

`cd ~`

弄清楚自己所在的位置 , 然后返回 `home` 目录

练习4. 创建目录 (`mkdir`)

`mkdir` 创建目录 `make directory`

```
pwd
cd ~
mkdir temp
mkdir temp/stuff
mkdir temp/stuff/things
mkdir -p temp/stuff/things/orange/apple/pear/grape
```

在 `Windows` 中，`\` 反斜杠也可以。

创建一个带空格的目录，方法是添加双引号，比如 `mkdir "I have fun"`。
假如文件夹已经存在，会报错。

在 `Windows` 中，创建多个目录 `mkdir 目录1,目录2`

在 `Unix` 中，创建多个目录 `mkdir 目录1 目录2`

练习5. 更改目录 (`cd`)

每次练习之前都先 `pwd` 然后 `cd ~`

`cd ..` 返回上一级目录

`cd ../../` 返回两级目录

为包含空格的目录加一个引号：`cd "I Have Fun"`

练习6. 列出目录中的内容 (`ls`)

`ls` 列出你当前所在目录的内容

`Unix` 用 `ls -lR`

- `-l` 是详细信息 (`long` 长格式)
- `-R` 是递归子文件夹 `recursive`

`Windows` 用 `dir -R`

练习7. 删除目录 (`rmdir`)

`rmdir` 目录名称

假如在 `macOS` 上 `rmdir`，你确定它是空的，但是拒绝删除，实际上在该目录中有一个名为 `.DS_Store` 的文件。

这种情况下，用 `rm -rf <dir>` 即可，`<dir>` 用实际的目录名代替。

练习8. 在多个目录中切换 (`pushd` 和 `popd`)

这些命令可以让你临时跑到某个不同的目录中，然后再回到之前的目录，并方便地在两个目录之间切换。

`pushd` 命令会将你所在的当前目录“推送” (`push`) 到一个列表中以供后续使用，然后让你转到

另一个目录中。

它的意思大致是：记住我现在的位置，然后到这个地方去。

`popd` 命令会将上次你推送的目录从列表中弹出（`pop`），然后让你回到这个被“弹出”的目录。

在 `Unix` 中，不带参数的 `pushd` 可以在两个目录之间来回切换。

在 `Unix` 中，创建多级目录，使用 `mkdir -p a/b/c`，这样即使中间目录不存在，也能创建成功。

练习9. 创建空文件（`touch` / `New-Item`）

`Unix` 中：`touch iamcool.txt`

`Windows` 中：`New-Item iamcool.txt -type file`

`New-Item` 命令还可以创建目录

在 `Unix` 中，删除非空目录会报错，在 `Windows` 中，会得到一个提示。

练习10. 复制文件（`cp`）

`cp awesome.txt something/`

`ls something/`

`cp -r something newplace`

注意 `Windows` 还能这么写 `cp -recurse something newplace`

在结尾放一个斜杠 `/`，这样做的目的是保证键入的名称确实是一个目录，如果这个目录不存在，会看到出错信息。

练习11. 移动文件（`mv`）

移动 `move` 文件，或者换种说法，重命名 `rename` 文件。

很简单，给出旧文件名和新文件名即可。

练习12. 查看文件内容（`less` / `more`）

`less test.txt`

退出 `less`，使用 `q` 键即可。

`more test.txt`

直接显示内容。可以用空格向下翻页，`w` 向上翻页，也可以用方向键。

在 Windows 中，只能用 `more test.txt` 。只能用空格逐页浏览。

练习13. 流文件内容显示 (`cat`)

```
cat test.txt
```

`cat` 命令会将整个文件一次输出到屏幕，不会分页也不会中间停顿。

在 Windows 中尝试：`cat ex12.txt,ex13.txt` 可以合并输出。

在 Unix 中尝试：`cat ex12.txt ex13.txt` 也可以合并输出，但是不用逗号。

练习14. 删除文件 (`rm`)

```
rm 文件1
```

```
rm 文件1 文件2 ... 文件n
```

```
rm -rf 目录名 递归删除文件
```

在 Windows 中使用 `rm -r 目录名`

练习15. 退出终端 (`exit`)

```
exit
```

更多任务

Unix 的命令清单如下：

- `xargs`
- `sudo`
- `chmod`
- `chown`

Windows 的命令如下：

- `forfiles`
 - `runas`
 - `attrib`
 - `icacls`
-