SI100+ 2024 Lecture 3

函数

SI100+ 2024 Staff | 2024-08-26

Part.0 在开始之前

如何查询已有函数的用法

- 我们已经学会了调用 `print`, `input` 来完成一些基本操作
- 但是 Python 还有更多函数
- 前面提到了 `open('filename', '?')`，万一我忘记了 `?` 处应该填什么
- 如何查询?

我们已经知道的：

- Python 官方文档：https://docs.python.org/zh-cn/3/
- STFW: Search the *Friendly* Web（记得看看 Lecture x 阅读材料）
- 问 GPT?

其实还有更多的方法...

help()

这是什么?

- help() 可以查看某函数或对象的帮助
- 在交互式命令行中输入 help(x) 并按下回车，可以看到关于 x 的帮助（less 模式）
  - 如果 x 是函数那么则是对于这个函数的帮助
  - 如果 x 是某个类型的字面值 / 变量，那么则是对于这个类型的帮助
- 在其他情况下，help() 就相当于 "print(帮助内容)"

演示：Notebook 示例 3.0.1

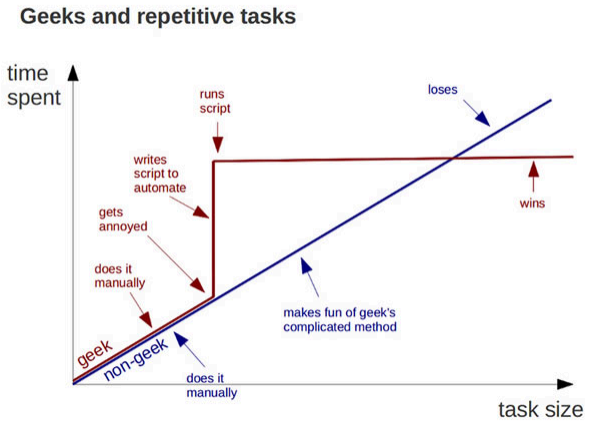
在交互式控制台（打开 Anaconda Prompt / Terminal，输入 Python 并回车后）

```
>>> help(print)
Help on built-in function print in module builtins:
print(*args, sep=' ', end='\n', file=None, flush=False)
    Prints the values to a stream, or to sys.stdout by default.
    sep
      string inserted between values, default a space.
    end
      string appended after the last value, default a newline.
    file
      a file-like object (stream); defaults to the current sys.stdout.
    flush
      whether to forcibly flush the stream.
~
(END)
```

- 这里有特殊的浏览模式，按方向键可以上下翻页，按 q 退出

[非课程内容] less

- Python 的控制台中的 help 使借用 less 命令输出的，在 linux 下，它是用来“展示”输出的（否则内容过多就会溢出屏幕而丢失）
- 在很久以前，计算机里只有 vi 编辑器，它规定的 :q 退出， J/K 用来上下翻页，Ctrl+U/D 快速上下翻页。而这成为了当时的习惯，被沿用下来。
- 当时 less 的作者 Mark Nudelman 只是想“方便的翻阅长长的报错”
  - 他当时用的 vi 版本不能打开这么大的日志文件
  - 另一个叫 more 的工具虽然能打开，但是不能向回翻 （现在似乎有了）
- less = vi 的操作模式 + more 的文件支持 的“查看器” （文件分页器）



- 极客的浪漫莫过于此
- 编程：重复事情上制胜的法宝

[非课程内容] 代码提示：Pylance 语法分析

- 将鼠标悬浮在 print 上，会弹出来一个小提示，会简单告诉你用法

```
(function) # 告诉我们 `print` 是一个函数
def print(
    *values: object,
    sep: str | None = " ",
    end: str | None = "\n",
    file: SupportsWrite[str] | None = None,
    flush: Literal[False] = False
) -> None: ...
```

- 除了鼠标悬浮，在你一边输入 print(...) 的时候，代码提示也会出现
- 并且会用醒目的标识来指示你正在输入哪个参数

```
# 尝试在 VSCode 中缓慢打出
print("Hello", end="!")
```

IPython 的小问号？

- 在 Jupyter Notebook / IPython 里，我们可以直接用 ？接在函数名字的末尾

```
In [1]: print?
Signature: print(*args, sep=' ', end='\n', file=None, flush=False)
# 上面告诉了我们用法格式/定义，下面这一串是 docstring（函数自身携带的文档的内容）
Docstring:
Prints the values to a stream, or to sys.stdout by default.

sep
  string inserted between values, default a space.
end
  string appended after the last value, default a newline.
file
  a file-like object (stream); defaults to the current sys.stdout.
flush
  whether to forcibly flush the stream.
# 最后告诉我们这是一个“内置函数或方法”
Type:      builtin_function_or_method
```

演示：Notebook 示例 3.0.2

目录

Part.1 Python 中的函数

为什么要有函数?

演示：Notebook 示例 3.1.1，3.1.2，3.1.3

- 你写了一个 Python 脚本，可以把摄氏度转换为华氏度
- 我想更方便的转换，不需要每次修改源代码 $\rightarrow$ 输入输出函数
- 更复杂的流程，我需要计算很多次
- 把你的代码复制粘贴很多次?
- 函数就是把一段代码“打包”起来，方便重复使用

即使转换的代码还没写，我只需要知道把需要转换的放在 `var_in`，然后 `var_out` 里面的就是转换完成的，不需要了解真正转换的时候是怎么样的。

在写转换的代码的时候也不需要思考转换之前都发生了什么，只需要把 `var_in` 转换后的值存储在 `var_out`

我们之前也做过这样“不用考虑发生了什么，只需要把东西放在指定的位置”的事：`print`, `type`

演示：Notebook 示例 3.1.4，3.1.5

函数 (function)

- 为什么要有函数?
  - 数学的函数，一个名字就省去了抄写复杂的定义： $\sin(x), \cos(x), \tan(x)$
  - 编程中，定义一个函数可以**防止重复的代码**（代码复用）
  - 让我们专注在更重要的内容上，隐藏过于复杂的细节
- 函数是什么?
  - 一段有名字的代码，可以通过那个名字（标识符）找到并开始运行它
  - 通俗理解，函数就是大段代码的“替身”

函数调用（call/invoke）

正确的让函数帮你做事的方法：

```
  print(   "Hello"   )
    ^   ^      ^      ^
    |   |      |      |
函数名 左括号   参数    右括号
```

缺东西，函数不做事：

```
  print  "Hello"
    ^       ^
    |       |
  函数名     参数
```

演示：Notebook 示例 3.1.6，3.1.7

函数调用（call/invoke）

调用可以没有参数，但是不能没有左右括号。

```
print() <- 里面是空的，没有参数
  ^
  |
函数名
```

当然有的函数也可以接受很多个参数

```
  print(   "Hello", "My", "SI100+"   )
    ^   ^       ^          ^             ^
    |   |       |          |             |
函数名  左括号    参数们    逗号分隔        右括号
```

演示：Notebook 示例 3.1.8

函数调用（call/invoke）

总结

- 调用函数必须有
  - 函数名，写在最前面
  - 一对括号，写在函数名后面
- 调用函数可以有
  - 参数
  - 很多个参数

为什么 `t = type(123)` 和上面的函数调用格式不太一样?

返回值（return value）

高中物理学：

$$v = \sqrt{\frac{gs_1^2}{2(H-h)}}$$

- 调用就是“套公式”
- 返回值就是“公式的结果”

```
v = calculate_that(g, s1, H, h)
```

所以，不写 `v = ...`，等号右侧的部分依然有其值

`t = type(123)`：把变量 `t` 赋值为 `type(123)` 这个函数执行的结果（返回值）

返回值（return value）

- 有的函数运行的没有明确的结果，比如 `print`
- 也存在没有返回值的函数
- 没有返回值 $\rightarrow$ 尝试获取返回值就会得到 `None`

```
1 result = print("Hello, World!")
2 print(result)       # None
3 print(type(result)) # <class 'NoneType'>
```

演示：Notebook 示例 3.1.9

内置函数初探

- `max(a, b)`: 返回 a 与 b 中最大值
- `min(a, b)`: 返回 a 与 b 中最小值
  - 支持像 `print` 那样，接收许多个参数，如 `max(a, b, c, d)`
- `abs(x)`: 返回 x 的绝对值

上面列出的都是有返回值的函数，阅读文档了解更多内置函数

https://docs.python.org/zh-cn/3/library/functions.html

方法 (method)

面向对象：有的“功能”和某些个体绑定

- 扫把和吸尘器都有“扫地”功能
- 矿泉水瓶没有“扫地”功能

演示：Notebook 示例 3.1.10

- 像是某种特殊的函数：扫把.扫地() → "abc".capitalize()
- 调用形式很相似，方法名、括号、括号内的参数
- 方法区别于函数，它不是 **随处可用** 的，而是 **依赖** 某个特定的字面值/变量
  - 例如，上述例子中，只有 str 类型的字面值/变量才能够使用 .capitalize()
  - P.S. 在 IPython / Notebook 中，也必须用 类型名.方法名? 来获取帮助
    - 但可以直接 help(变量/字面值.方法名)
- 我们并不会在这次课程中涉及到怎么写 **方法**，因为还需要很多面向对象的前置知识

Part.2 定义你的函数

代码块（Revisited）

Python 的特色之一是依据代码行的 **缩进 (indentation)** 确定代码块：

```
1 if 1 + 1 == 2:
2     print("basic statement passed!") # 我和 if 语句不在同一个代码块！
3
4 a = 10
5 b = 20
6 # 上面的两行在同一个代码块
```

定义函数

- 我希望我写的许多行 Python 被打包起来
- 使用缩进!

```
1 # 下面两行没有缩进
2 celsius = var_in
3 fahrenheit = celsius * 9 / 5 + 32
```

```
1 # 下面两行有缩进
2     celsius = var_in
3     fahrenheit = celsius * 9 / 5 + 32
```

定义函数

只是缩进并不能让 Python 理解你写的这部分是一个函数，你需要主动说明。

- 函数的“必须”和“可能”
  - **函数名**，你需要给你的函数起一个名字（复习前面的标识符命名规则）
  - **小括号**，里面装着**参数**

```
# 下面两行有缩进，但是没有说明这是函数
    celsius = var_in
    fahrenheit = celsius * 9 / 5 + 32
```

```
# 我们从外面拿进来 var_in，即：var_in 是函数的参数
celsius_to_fahrenheit(var_in)
    celsius = var_in
    fahrenheit = celsius * 9 / 5 + 32
```

定义函数

Python 的函数定义格式：

```
# 已经非常接近 Python 的函数定义了
celsius_to_fahrenheit(var_in)
    celsius = var_in
    fahrenheit = celsius * 9 / 5 + 32
```

```
# 完整的 Python 函数定义
def celsius_to_fahrenheit(var_in): # 注意这里必须有一个冒号
    celsius = var_in
    fahrenheit = celsius * 9 / 5 + 32
```

```
# 直接把 celsius 当作参数
def celsius_to_fahrenheit(celsius): # 注意这里必须有一个冒号
    fahrenheit = celsius * 9 / 5 + 32
```

定义函数

```
1 def celsius_to_fahrenheit(celsius):
2     fahrenheit = celsius * 9 / 5 + 32
```

- 要定义一个函数，我们会用关键字 def 开头，后接函数名称和一对圆括号
- 圆括号里面是**参数** (Parameter)，可以有多个，用逗号分隔
- 圆括号里写下的变量可以直接在函数里面使用，实际的值将由调用时 **真正的参数** 决定
  - 例如 f(2) 会自动将 x 赋值为 2，这样才计算得出了正确的结果
- **一定不要忘记那个冒号！**
  - 冒号相当于告诉 Python 后面的代码块是函数的内容
  - 代码块取决于**缩进**，冒号后面跟的是很多行开头是 4 个空格的代码，同属于一个代码块

返回语句

测试一下我们的函数能不能正常工作

```
1 def celsius_to_fahrenheit(celsius):
2     fahrenheit = celsius * 9 / 5 + 32
3 
4 result = celsius_to_fahrenheit(23)
```

演示： Notebook 示例 3.2.1

- 返回值是 None，也就是没有返回值
- 该怎么让 Python 知道我们想要返回 fahrenheit?

```
1 def celsius_to_fahrenheit(celsius):
2     fahrenheit = celsius * 9 / 5 + 32
3     return fahrenheit
```

演示： Notebook 示例 3.2.2

函数声明与执行的顺序

def 那一行也是没有缩进，在最底层代码块，运行到 def 的时候发生了什么？

演示：Notebook 示例 3.2.3

```
1 say_my_name("初音未来")
2
3 def say_my_name(name):
4     print("我去,", name, "!")
```

```
NameError
Cell In[3], line 3
      1 # 示例 3.2.3 - 课件 p27
----> 3 say_my_name("初音未来")
      5 def say_my_name(name):
      6     print("我去,", name, "!")

NameError: name 'say_my_name' is not defined
```

函数声明与执行的顺序

- 函数必须先声明才能调用
  - 什么是声明？就是从 `def` 一直到代码块结束！
- Python 顺序执行
- 执行到 `def` 那一行才相当于有了这个函数
- 调用相当于从调用的位置跳到函数体第一行
  - 函数体内依然顺序执行
  - 函数运行到哪里停止？
    - 执行了 `return` 语句 **演示**：Notebook 示例 3.2.4
    - 执行完函数体的最后一行
- 执行完函数体，返回到调用的位置

函数声明与执行的顺序

演示： Notebook 示例 3.2.5

```
1 def say_my_name(name):
2     print("我去,", name, "!")
3
4 print("是谁？")
5 say_my_name("初音未来")
6 print("你好！")
```

函数声明与执行的顺序

演示： Notebook 示例 3.2.5

```
1 def say_my_name(name):
2     print("我去,", name, "!")
3
4 print("是谁？")
5 say_my_name("初音未来")
6 print("你好！")
```

函数声明与执行的顺序

演示： Notebook 示例 3.2.5

```
1 def say_my_name(name):
2     print("我去,", name, "!")
3
4 print("是谁？")
5 say_my_name("初音未来")
6 print("你好！")
```

函数声明与执行的顺序

演示： Notebook 示例 3.2.5

```
1 def say_my_name(name):
2     print("我去,", name, "!")
3
4 print("是谁？")
5 say_my_name("初音未来")
6 print("你好！")
```

函数声明与执行的顺序

演示： Notebook 示例 3.2.5

```
1 def say_my_name(name):
2     print("我去,", name, "!")
3
4 print("是谁？")
5 say_my_name("初音未来")
6 print("你好！")
```

函数声明与执行的顺序

演示： Notebook 示例 3.2.5

```
def say_my_name(name):
    print("我去,", name, "!")

print("是谁？")
say_my_name("初音未来")
print("你好！")
```

函数声明与执行的顺序

演示：Notebook 示例 3.2.5

```
1 def say_my_name(name):
2     print("我去,", name, "!")
3
4 print("是谁？")
5 say_my_name("初音未来")
6 print("你好！")
```

Part.3 函数里里外外

参数：函数沟通的桥梁

- 参数按照**出现位置**可以简单分为形式参数 (Parameter) 和实际参数 (Argument)
  - 形式参数：就是写在定义处的参数 `def f(x1, x2):...`
    - 公式里面的字母们
    - 除了会作为变量使用之外，还能规定函数的形式（想要多少个参数）
  - 实际参数：就是调用处的参数 `f(1, 2)`
    - 套公式的时候代入的数
    - 这里的参数是实际拿来运算的值

参数把函数外的东西起了别名，带了到函数里面

全局变量和局部变量

- 函数体是一个代码块
- 最外层无缩进的部分也是一个代码块
- 重名了怎么办?

```python
def celsius_to_fahrenheit(celsius):
    fahrenheit = celsius * 9 / 5 + 32
    return fahrenheit

# 这里也有 celsius 和 fahrenheit，但是代码还是可以运行
celsius = 23
fahrenheit = celsius_to_fahrenheit(celsius)
```

李华：一个人尽皆知的故事

全局变量和局部变量

- 有人提到“李华”
  - 高考英语卷中的李华
- 如果你身边真的有一个李华同学
  - 你的同学李华

全局变量（Global Variable）：定义在最外层的变量，作用范围为全局

英语卷的李华，所有人（作用域）知道他

局部变量（Local Variable）：定义在有缩进的区域，作用范围为当前代码块

你的同学李华，只有你们学校（作用域）的人知道他

作用域（Scope）：变量的作用范围

作用域

演示：Notebook 示例 3.3.1

```
which_li_hua = "高考英语卷中"

def ShanghaiTech():
    which_li_hua = "上海科技大学"
    print("上科大的", which_li_hua)

def SUSTech(): # 假设隔壁大学没有李华
     # 隔壁大学人的第一反应肯定是高考英语卷中的李华
    print("隔壁大学", which_li_hua)

# 外面的人（最外层）没有听说过别的李华
print("外面的人", which_li_hua)
ShanghaiTech()
SUSTech()

# ShanghaiTech 的李华不会影响外面的人心中的李华
print("外面的人", which_li_hua)
```

Takeaway Message

- **RTFM**, **STFW**, `help()`, `?`
- 函数是代码的“替身”，可以把一段代码打包起来，方便重复使用
- 函数的定义格式：`def 函数名(参数):`
- 函数的调用格式：`函数名(参数)`
- 函数的返回值 & 返回语句：`return`
- 全局变量和局部变量 & 变量的作用域

Thanks for Listening

Any questions?