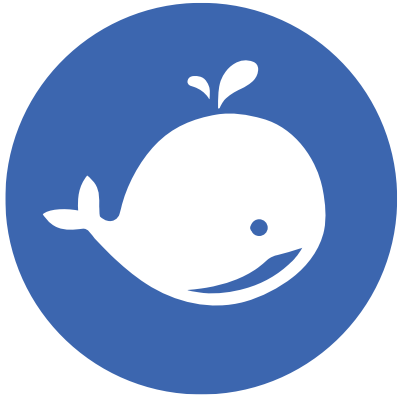


Datawhale

高校论坛第三期

--代码风格

张凯旋

目录

Datawhale

CONTENTS

/01 风格介绍

代码风格

Datawhale

```
# 输入身高和体重
height=float(input("请输入您的身高："))
weight = float(input("请输入您的体重："))
bmi= weight/(height * height)# 计算BMI指数
print("您的BMI指数为：" + str(bmi))  #输出BMI指数
# 判断身材是否合理
if bmi < 18.5:print("体重过轻 ~@_@~")
if bmi >= 18.5 and bmi < 24.9:
    print("正常范围，注意保持 (-_-)")
if bmi >= 24.9 and bmi < 29.9:print("体重过重 ~@_@~")
if bmi >= 29.9 :
    print("肥胖 ~@_@~")
```

```
'''
   @ 功能：根据身高、体重计算BMI指数
   @ author:无语
   @ create:2017-10-31
'''
# 输入身高和体重
height = float(input("请输入您的身高："))
weight = float(input("请输入您的体重："))
bmi = weight/(height * height)          # 计算BMI指数
print("您的BMI指数为：" + str(bmi))    # 输出BMI指数

# 判断身材是否合理
if bmi < 18.5:
    print("体重过轻 ~@_@~")
if bmi >= 18.5 and bmi < 24.9:
    print("正常范围，注意保持 (-_-)")
if bmi >= 24.9 and bmi < 29.9:
    print("体重过重 ~@_@~")
if bmi >= 29.9 :
    print("肥胖 ~@_@~")
```

- **“修饰”的前提一定是“跑通”代码**，也就是说你得先确保你的code没有任何的bug
- 这种“修饰”并不是对每一个人都强制学习的内容，他并不影响你快乐学习接下来的所有课程。

/02 缩进与换行

代码布局 - 缩进

Datawhale

1. 如果有开始定界符，其余行的缩进需与开始定界符对齐。
2. 需要额外的4个空格（长度等于一个Tab键），以区分开传入参数，和其他内容。
3. 空格一般用于添加以上这种缩进，Tab键一般用于保持行与行之间的一致性。
4. 多行if语句衔接，需要一个额外的缩进，以区分其他内容

代码布局 - 缩进

Datawhale

```
# Arguments on first line forbidden when not using vertical alignment.
foo = long_function_name(var_one, var_two,
    var_three, var_four)

# Further indentation required as indentation is not distinguishable.
def long_function_name(
    var_one, var_two, var_three,
    var_four):
    print(var_one)

# Add some extra indentation on the conditional continuation line.
if (this_is_one_thing and
    that_is_another_thing):
    do_something()
```

代码布局 - 缩进

Datawhale

```
# Aligned with opening delimiter.
foo = long_function_name(var_one, var_two,
                         var_three, var_four)

# Add 4 spaces (an extra level of indentation) to distinguish arguments from the rest.
def long_function_name(
        var_one, var_two, var_three,
        var_four):
    print(var_one)

# Add some extra indentation on the conditional continuation line.
if (this_is_one_thing
        and that_is_another_thing):
    do_something()
```

代码布局 – 换行

Datawhale

1. 将所有行限制为最多 79 个字符。
2. 一般语句接受“隐式”延续，但是with语句等不支持，需要使用反斜杠 \ 来衔接。
3. 另一个这样的例子是assert语句。
4. 多行if语句的缩进详见上一小节。
5. 通常不鼓励使用复合语句（同一行上的多个语句）。

代码布局 – 换行

Datawhale

No

```
1 with open('/path/to/some/file/you/want/to/read') as file_1,open('/path/to/some/file/being/written'
```

Yes

```
1 with open('/path/to/some/file/you/want/to/read') as file_1, \
2         open('/path/to/some/file/being/written', 'w') as file_2:
3     file_2.write(file_1.read())
```

代码布局 – 换行

Datawhale

```
# Wrong:
if foo == 'blah': do_blah_thing()
do_one(); do_two(); do_three()
```

```
# Correct:
if foo == 'blah':
    do_blah_thing()
do_one()
do_two()
do_three()
```

代码布局 – 换行

Datawhale

```
# operators sit far away from their operands
income = (gross_wages +
          taxable_interest +
          (dividends - qualified_dividends) -
          ira_deduction -
          student_loan_interest)
```

```
# easy to match operators with operands
income = (gross_wages
          + taxable_interest
          + (dividends - qualified_dividends)
          - ira_deduction
          - student_loan_interest)
```

/03 导入规范

导入规范

Datawhale

- 假设我们现在有这样一个文件目录：

```
Tree
|____ m1.py
|____ m2.py
|____ Branch
      |____m3.py
      |____m4.py
```

在m2.py写入代码：

```
def printSelf():
        print('In m2')
```

在m3.py写入代码：

```
def printSelf():
        print('In m3')
```

导入规范

Datawhale

- **import 本地模块/包**

1. 当我们需要导入本地自己封装好的一些模块时，需要通过import来导入。
2. 如果我们需要在m1.py文件中导入同目录下的m2.py文件，直接导入即可。

导入规范

Datawhale

- **import Python库的模块/包**

1. Python的基础库自带了丰富的模块和包，无需自己逐一实现，只需要一键import即可享用。
2. 使用 from x import y 其中x是包前缀，y是没有前缀的模块名称。

导入规范

Datawhale

- 假设我们现在有这样一个文件目录：

```
Tree
|____ m1.py
|____ m2.py
|____ Branch
      |____m3.py
      |____m4.py
```

在m2.py写入代码：

```
def printSelf():
        print('In m2')
```

在m3.py写入代码：

```
def printSelf():
        print('In m3')
```

导入规范

Datawhale

- import 本地模块/包

```
1 import os
2 import m2
3 m2.printSelf()
4 >>> In m2    # 输出的内容
```

- import Python库的模块/包

```
1 # wrong
2 import os
3 import m3
4 m3.printSelf()
```

导入规范

Datawhale

- import Python库的模块/包

```
# wrong
import os
import m3
m3.printSelf()
```

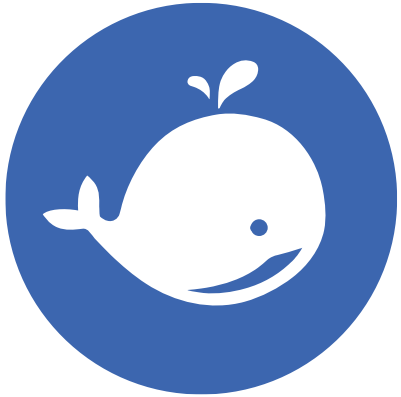
```
# correct
from Branch import m3
m3.printSelf()
>>> In m3    # 输出的内容
```

```
Tree
|____ m1.py
|____ m2.py
|____ Branch
      |____m3.py
      |____m4.py
```

目录

Datawhale

CONTENTS



Datawhale

高校论坛第三期

--代码风格

徐辉

目录

Datawhale

CONTENTS

/01 关于空格

关于空格

Datawhale

强迫症式

```
i = i + 1

submitted += 1

x = x * 2 - 1

hypot2 = x * x + y * y

c = ( a + b ) * ( a - b )
```

凌乱式

```
i=i+1

submitted +=1

x = x * 2 - 1

hypot2 = x*x + y * y

c =(a + b) * (a - b)
```

建议式

```
i = i + 1

submitted += 1

x = x*2 - 1

hypot2 = x*x + y*y

c = (a+b) * (a-b)
```

关于空格

Datawhale

1.紧接在圆括号、方括号或大括号内，不需要多余空格；

2.在逗号、分号或冒号之前，尾随逗号之后均不需要多余空格；

3.在切片中，两个冒号必须应用相同的间距；

4.紧接在开始函数调用的参数列表的左括号之前，不需要多余空格；

5.赋值（或其他）运算符周围需要多个空格以使其与另一个运算符对齐；

/02 代码注释

代码注释

Datawhale

- 注释就是对代码的解释和说明，其目的是让人们能够更加轻松地了解代码。
- 注释是编写程序时，写程序的人给一个语句、程序段、函数等的解释或提示，能提高程序代码的可读性。
- 在有处理逻辑的代码中，源程序有效注释量必须在20%以上。

单行注释

```
# Importing the OS module.
import os

epochs = 20  # Represents the number of iterations.
```

多行注释

```
'''
This is the first line of comment.
The specific annotation content is to be determined.
'''

"""
This is the first line of comment.
The specific annotation content is to be determined.
"""
```

/03 命名规则

命名规则

Datawhale

•变量名只能包含字母、数字和下划线。变量名可以字母或下划线打头，但不能以数字打头，例如，可将变量命名为message_1，但不能将其命名为1_message。

•变量名不能包含空格，但可使用下划线来分隔其中的单词。例如，变量名greeting_message 可行，但变量名greeting message会引发错误。

```
# Wrong
1_second = "1 sec"
send message = "You are right."
```

```
# Correct
one_second = "1 sec"
send_message = "You are right.
```

/04 命名规范

命名规范

Datawhale

•不要将Python关键字和函数名用作变量名，即不要使用Python保留用于特殊用途的单词，如print。

•变量名与函数名应既简短又具有描述性。例如，name比n好， student_name比s_n好。

代码判断

1

```
print = 10
print
```

2

```
int = "1024"
print(int(int))
```

3

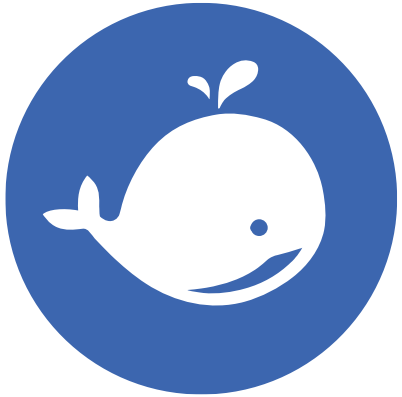
```
l=1;o=0
```

函数命名

```
# Non-standard writing
def A():
    pass

def Fibonacci(a, b):
    return a + b
```

```
# Standard writing
def calculator(num_1, num_2, operator):
    if operator == "+":
        return num1 + num_2
    elif operator == "-":
        return num_1 - num_2
    else:
        print("Operator unrecognized.")
```



Thank you