Physics 725- Scientific Programming with Python

Good Coding Style

Alexander Wallau & Christoph Geron

24. Mai 2023

Universität Bonn

Inhaltsverzeichnis

Good Coding Style

PEP8

PEP8

PEP8

Python Enhancement Proposals

PEP8

Python Enhancement Proposals

PEP 8 - Style Guide for Python code

PEP8

A concern of the tutors from the computer science department for the benefit of all involved.

Exibit A

```python
# Bad code we want to improve:
x=1
y=[21,42]
def fn(x,y):
 x=x*2
 if x==y:
  print("This is correct")
  return True
 else:
  print("This is false")
  return False
z="data/"
x=fn(x,y)
print(x)
```

Blank lines

```python
x = 1
y = [21, 42]


def fn(x, y):
    x = x * 2
    if x == y:
        print("This is correct")
        return True
    else:
        print("This is false")
        return False


z = "data/"
x = fn(x, y)
print(x)
```

Internationalisation

All common programming languages are designed for the English language, as can be seen from the names of the built-in functions, e.g. *print()* or *type()*.
Good Code should be ïn English".

Descriptive Names

Choosing variable names is a science in itself. In geneal short name e.g. *x*, *y* or *a* are a poor choice. Good names are e.g. *res1*, x_valsor

Exibit A

```
number_to_double = 1
val_to_compare_to = [21, 42]


def check_if_double(number_to_double, val_to_compare_to):
    number_to_double = number_to_double * 2
    if number_to_double == val_to_compare_to:
        print("This is true")
        return True
    else:
        print("This is false")
        return False


data_path = "data/"
number_to_double = check_if_double(number_to_double, val_to_compare_to)
print(number_to_dubble)
```

Structuring

Blank lines can aid readability. Explicit rules for those are not yet conceived.

Exibit A

```python
number_to_double = 1
val_to_compare_to = [21, 42]


def check_if_double(number_to_double, val_to_compare_to):

    number_to_double = number_to_double * 2

    if number_to_double == val_to_compare_to:
        print("This is true")
        return True

    else:
        print("This is false")
        return False


data_path = "data/"

number_to_double = fn(number_to_double, val_to_compare_to)
print(number_to_dubble)
```

Names

Avoid using the same variable names in different sections of your code!

Names II

Variables outside a function should **always** have different names than the variables inside the function.
In addition, a new value should also have a new name.

Exibit A

```python
num1 = 1
x_vals = [21, 42]


def check_if_double(number_to_double, val_to_compare_to):

    number_to_dubble = number_to_double * 2

    if number_to_double == val_to_compare_to:
        print("This is true")
        return True

    else:
        print("This is false")
        return False


data_path = "data/"

res = check_if_double(num1, x_vals)
print(res)
```

Structure

Imports, Functions and Constants belong at the top!

Exibit A

```python
DATA_PATH = "data/"


def check_if_double(number_to_double, val_to_compare_to):

    number_to_dubble = number_to_double * 2

    if number_to_double == val_to_compare_to:
        print("This is true")
        return True

    else:
        print("This is false")
        return False


num1 = 1
x_vals = [21, 42]

res = check_if_double(num1, x_vals)
print(res)
```

How to comment

Code **must** be commented! There are two types of Comments:

- The one-liner: # this is a comment

How to comment

Code **must** be commented! There are two types of Comments:

- The one-liner: # this is a comment
- The Docstring: ”This is a potential multi-line comment”

How to comment

Code **must** be commented! There are two types of Comments:

- The one-liner: # this is a comment
- The Docstring: "This is a potential multi-line comment"

Comments are (mostly) written in the English imperative:

```
code = crazy_fn(True) ## call function, send message, recieve new code
```

Spacing

Put a space between # and the actual comment.

Spacing

Put a space between # and the actual comment.
If the comment is placed after the programme code, **two** spaces must be placed between the code and #.

Descriptiveness

The description of a function always includes:

- Why does this funtion exist/ What does this function do

Descriptiveness

The description of a function always includes:

- Why does this funtion exist/ What does this function do
- What are the necessary inputs? - do they require a specific type e.g. booleaan?

Descriptiveness

The description of a function always includes:

- Why does this funtion exist/ What does this function do
- What are the necessary inputs? - do they require a specific type e.g. booleaan?
- Does the function return something? - If so, then what exactly?

Exibit A

```python
DATA_PATH = "data/"  # path were data is expected and read from


def check_if_double(number_to_double, val_to_compare_to):
    """
    Take two params, double first one, compare both
    return boolean if both are equal
    """

    # double value
    number_to_dubble = number_to_double * 2  # multiply by two to double the value

    # compare values - print if is equal, return
    if number_to_double == val_to_compare_to:
        print("This is true")  # print that this is true to terminal
        return True  # return true if value was true

    # this is what we if the if check fails
    else:
        print("This is false")  # print in terminal that this is false
        return False  # return false


num1 = 1  # this value is one
x_vals = [21, 42]  # values meassured in last experiment

# verifiy that first value is twice the second one
res = check_if_double(num1, x_vals)  # call a function
print(res)  # print result to terminal
```

What comments?

Are there unnecessary comments? - Oh yes, loads actually.

Meme

Nick Moore 1 day ago

CODE COMMENTS BE LIKE

Exibit A

```python
DATA_PATH = "data/"  # path were data is expected and read from


def check_if_double(number_to_double, val_to_compare_to):
    """
    Take two params, double first one, compare both
    return boolean if both are equal
    """

    # double value
    number_to_dubble = number_to_double * 2

    # compare values - print if is equal, return
    if number_to_double == val_to_compare_to:
        print("This is true")
        return True

    else:
        print("This is false")
        return False


num1 = 1
x_vals = [21, 42]  # values meassured in last experiment

# verifiy that first value is twice the second one
res = check_if_double(num1, x_vals)
print(res)  # print result to terminal
```

snake_case

The current convention in python is that snake_case is being used instead of *camelCase*.

snake_case

The current convention in python is that snake_case is being used instead of *camelCase*.
In addition, variable and function names **always** begin with a lower case letter. Constants are excluded (these are written in CAPS and snake_case).

snake_case

The current convention in python is that snake_case is being used instead of *camelCase*.
In addition, variable and function names **always** begin with a lower case letter. Constants are excluded (these are written in CAPS and snake_case).
(Only classes are written in CamelCase and begin with a capital letter, but this is not relevant here).

Exibit A

```python
DATA_PATH = "data/"  # path were data is expected and read from


def WrongWrittenFunction(justForDemontration):
    """How did this get in here?! - Delete this! Now!!!"""
    print("iAmYourJava")

def check_if_double(number_to_double, val_to_compare_to):
    """
    Take two params, double first one, compare both
    return boolean if both are equal
    """

    # double value
    number_to_dubble = number_to_double * 2

    # compare values - print if is equal, return
    if number_to_double == val_to_compare_to:
        print("This is true")
        return True

    else:
        print("This is false")
        return False


num1 = 1
x_vals = [21, 42]  # values meassured in last experiment

# verifiy that first value is twice the second one
res = check_if_double(num1, x_vals)
print(res)  # print result to terminal
```

Line Length

Take a break and breathe deeply.

Line Length

Take a break and breathe deeply.

By convention, a line of code should contain no more than 160 characters.

Line Length

Take a break and breathe deeply.

By convention, a line of code should contain no more than 160 characters.

If you need more, you should either urgently break up the code a little further and store more intermediate values or look at how line breaks work.

Line Length

Take a break and breathe deeply.
By convention, a line of code should contain no more than 160 characters.
If you need more, you should either urgently break up the code a little further and store more intermediate values or look at how line breaks work.
If in doubt, the former will be the solution :)

Yes, it's very tempting to squeeze everything into one line to look cool. But it's cooler if you write your code in a way that is easy for you and others to understand.

Exibit B

```python
# bad practice
x = print(str(3.141592653 + 42 + int(input("Gib eine Zahl ein "))))
```

Exibit B

```
linenos# better practice
in_val = int(input("Gib eine Zahl ein "))
res = 3.141592653 + 42 + in_val
x = print(str(res))
# okayto cast res into a String since its not useful here, but it's about principle :)
```

Exibit C

```python
array = [1 if i % 2 != 0 else 0 for i in range(10)]  # create a list with alternating ones and zeroes
print(array)
```

Slicing

```python
# slicing - briefly: no spaces as long as no functions / calculations are included
string = "that's good code you're writing there :D"
string[1:4]
string[1 + 4 : 8]
string[: len(string) - 2 : -1]
string[::-1]
```

Parameters

```
linenos# If the list of parameters is too long, you can divide it into several lines.
def crazy_long_fn(first_val,
              second_val,
              third_val,
              fourth_val):
    print("Hello there :)")
```

Parameters II

```
# same for function calls
x = crazy_long_fn(52, 48,
                  21, 42)
```

Imports

```python
# Order von imports
# built in (standard library)
import math

# externally loaded libraries (z.B. über pip geladen)
import numpy
from matplotlib import pyplot as plt

# files / modules you have written by yourself
import my_math

# NEVER do: from x import *
```

Kwargs

```python
# Keyword arguments are great!
# kwargs can help to better document the code and show what kind of value is expected by default.
# but they are not a panacea!
# Parameters that are REQUIRED, otherwise the function makes no sense, should not be given a default value
def my_keyword_fn(first_in, second_in, overwrite=True, iterations=42):
    print("This function does nothing so far... but it's signature looks cool :D")
```

TL:DR

TL:DR: Be nice to your Tutors and Group Mates and write clean and pretty Code :-)

End

Thank you for participating

End

Thank you for participating

Contact details

Contact details:

- Mail: **wallau@uni-bonn.de**
- Discord: **A91202#0931**