Day II - Part I - Best Practices

December 3, 2020

1 Best Practices

In this chapter we will discuss about a number of pillars of software development:

- Documentation
- Modularisation
- Automated testing

2 Documentation

[23]:
```python
import math

def f(u, v):
    l = len(u)
    s = 0
    for i in range(l):
        s += (u[i] - v[i]) ** 2
    return math.sqrt(s)
```

Consider this function `f`.

1. Is it clear for you to understand what the fuction does?
2. Would you make any changes to the function so that is easier for you to understand its usage?

In software developement documentation is very important. Documentation allows ourselves, our collaborators and the future contributors to understand the usage of written code.

There are several ways that we can document source code. The two main ways covered in this workshop are:

- A "manual" for each part of your code
- Meaningful variable and function names

Adding meaningful variable and function names

[ ]:
```python
import math

def euclidean_distance(u, v):
    vector_length = len(u)
    distance = 0
```

```python
    for i in range(length):
        distance += (u[i] - v[i]) ** 2
    return math.sqrt(distance)
```

Adding a "manual" for each part of your code

[5]:
```python
import math

def euclidean_distance(u, v):
    """
    Computes the Euclidean distance between two vectos `u` and `v`.

    The Euclidean distance between `u` and `v`, is defined as:

    \sqrt{(u_1 - v_1) ^ 2 + ... + (u_n - v_n) ^ 2}

    Parameters
    ----------
    u : list
        Input vector.
    v : list
        Input vector.

    Returns
    -------
    euclidean : double
        The Euclidean distance between vectors `u` and `v`.
    """
    vector_length = len(u)
    distance = 0
    for i in range(length):
        distance += (u[i] - v[i]) ** 2
    return math.sqrt(distance)
```

Adding Pythonic tweaks

[7]:
```python
import math

def euclidean_distance(u, v):
    """
    Computes the Euclidean distance between two vectos `u` and `v`.

    The Euclidean distance between `u` and `v`, is defined as:

    \sqrt{(u_1 - v_1) ^ 2 + ... + (u_n - v_n) ^ 2}

    Parameters
    ----------
```

```
    u : list
        Input vector.
    v : list
        Input vector.

    Returns
    -------
    euclidean : double
        The Euclidean distance between vectors `u` and `v`.
    """
    distance = 0

    for u_i, v_i in zip(u, v):
        distance += (u_i - v_i) ** 2

    return math.sqrt(distance)
```

3 Testing

Considering now the function `euclidean_distance` how can we be sure that it is correct?

We know that the euclidean distance of the two vectors $u = (2, -1)$ and $v = (-2, 2)$ is:

$$
\begin{aligned}
dist((2, -1), (-2, 2)) =& \sqrt{(2 - (-2))^2 + ((-1) - 2)^2} \\
=& \sqrt{(4)^2 + (-3)^2} \\
=& \sqrt{16} + \sqrt{9} \\
=& \sqrt{25} \\
=& 5.
\end{aligned}
$$

[8]:
```
u = (2, -1)
v = (-2, 2)

euclidean_distance(u, v)
```

[8]: 5.0

[9]:
```
assert euclidean_distance(u, v) == 5
```

4 Modularization

Consider now this function `f` and repeat the discussion.

1. Is it clear for you to understand what the fuction does?
2. Would you make any changes to the function so that is easier for you to understand its usage?

[47]:
```
def f(u, v, isM, isE):

    if isM == 1:
        l = len(u)
        s = 0
        for i in range(l):
            s += abs(u[i] - v[i])
        return s

    if isE == 1:
        l = len(u)
        s = 0
        for i in range(l):
            s += (u[i] - v[i]) ** 2
        return math.sqrt(s)
```

Documentation is an improtant practice which makes computer code more understable. A further important practice is modularization. Modularization does not only make code more readable, but it is also easier to test modularized code.

[48]:
```
def manhattan_distance(u, v):
    """
    Computes the Manhattan distance (Taxicab distance) between two vectos `u`␣
↪and `v`.

    The Manhattan distance between `u` and `v`, is defined as:

    \sum_{i=1}^{N}|u_i - v_i|

    Parameters
    ----------
    u : list
        Input vector.
    v : list
        Input vector.

    Returns
    -------
    manhattan : double
        The Manhattan distance between vectors `u` and `v`.
    """
    distance = 0
    for u_i, v_i in zip(u, v):
        distance += abs(u_i - v_i)

    return distance
```

[49]:
```
def get_distance(u, v, mode):
    if mode == "euclidean":
        return euclidean_distance(u, v)

    if mode == "manhattan":
        return manhattan_distance(u, v)

    return 'Please use a feasible mode.'
```

[37]:
```
u = [10, 20, 15, 10, 5]
v = [12, 24, 18, 8, 7]
```

[39]:
```
assert manhattan_distance(u, v) == 13
```

[35]:
```
assert euclidean_distance(u, v) == 6.082762530298219
```

[43]:
```
import numpy as np

assert np.isclose(euclidean_distance(u, v), 6.0827, rtol=1e-03)
```

[44]:
```
assert get_distance(u, v, mode='euclidean') == euclidean_distance(u, v)
```

[45]:
```
assert get_distance(u, v, mode='manhattan') == manhattan_distance(u, v)
```

Example with three distance measures

[17]:
```
def hamming_distance(u, v):
    """
    Computes the Hamming distance between two strings `u` and `v`.

    The Hamming distance between two equal-length strings of symbols
    is the number of positions at which the corresponding symbols are
    different.

    Parameters
    ----------
    u : string
        Input string.
    v : string
        Input string.

    Returns
    -------
    hamming : double
        The Hamming distance between strings `u` and `v`.
    """
    distance = 0
```

```
    for element_one, element_two in zip(u, v):
        if element_one != element_two:
            distance += 1
    return distance
```

[18]:
```
def get_distance(u, v, mode):
    if mode == "euclidean":
        return euclidean_distance(u, v)

    if mode == "hamming":
        return hamming_distance(u, v)

    if mode == "manhattan":
        return manhattan_distance(u, v)

    return 'Please use a feasible mode.'
```