

# Advanced Softwareengineering

## Football World Cup Predictor

Github link:

[github.com/Arktius/Pet Project](https://github.com/Arktius/Pet_Project)

Denis Baskan – 878 571

01 September 2018

### Inhalt

Introduction.....	3
Crawl Websites.....	3
Predictor .....	4
Football Tournament.....	4
Result.....	4
UML .....	5
Use Case Diagram.....	5
Package Diagram .....	6
Activity Diagram – Web Scraper.....	7
Activity Diagram – Predictor .....	8
Sequence Diagram.....	9
Metrics.....	10
Sonar Cloud .....	10
Pylint.....	11
Clean Code Development.....	12
Version Control.....	12
Source Code Conventions .....	13
Precise Naming.....	13
Removing unnecessary comments.....	14
Vertical Separation .....	15
Continuous Delivery .....	16

AOP .....	17
DSL.....	18
Functional Programming .....	19
Final Data Structures .....	19
Side effect free functions .....	19
High order functions – functions as parameters and return values .....	19
Closures .....	19
Anonymous Functions.....	20
Logical Solver .....	21
Scala.....	22

## Introduction

In football world cup 2018, we could watch many exciting matches. Although Germany has performed very bad, I was motivated to develop code in order to predict scores. Therefore, we need data, but more about that later. The idea had come into my mind few weeks before the first match started. Friends and acquaintances have started betting on matches. We have started a little tournament where the first three places earn money. As two years ago in the European Cup, I have recognized that sometimes even the guys, who are fanatic footballers, perform worse than children without any knowledge about the teams. So I wanted to compare a neural network against them.



Source: <https://www.kuredu.com/wp-content/uploads/2018/06/fifa.jpg>

## Crawl Websites

What data could be relevant for our predictor? Football players usually play not more than three world cups. So my suggestion was to take many matches such as past World and European Cups, U20 and U21 matches, qualifying and other matches.

The domain [http://www.weltfussball.de/alle\\_spiele/](http://www.weltfussball.de/alle_spiele/) and their sub-domains provide many matches. As I wrote the code snippets, the data of qualifying and other matches were available on <http://www.sportdaten.t-online.de/fussball/>. Unfortunately, this website is no longer accessible. But I still have the data stored on my computer.

## Predictor

I decided to use an artificial neural network (ANN) to get some predictions. Inputs are two nations, year and kind e.g. World Cup, European Cup and so on. Outputs are two score vectors that contain probabilities. The two highest probabilities are our predictions for a given match.

## Football Tournament

You might have played a little tournament with your friends and family where points are given in respect to your predictions. If you want to compare your predictor against some of your revivals, there is a function to evaluate your results.

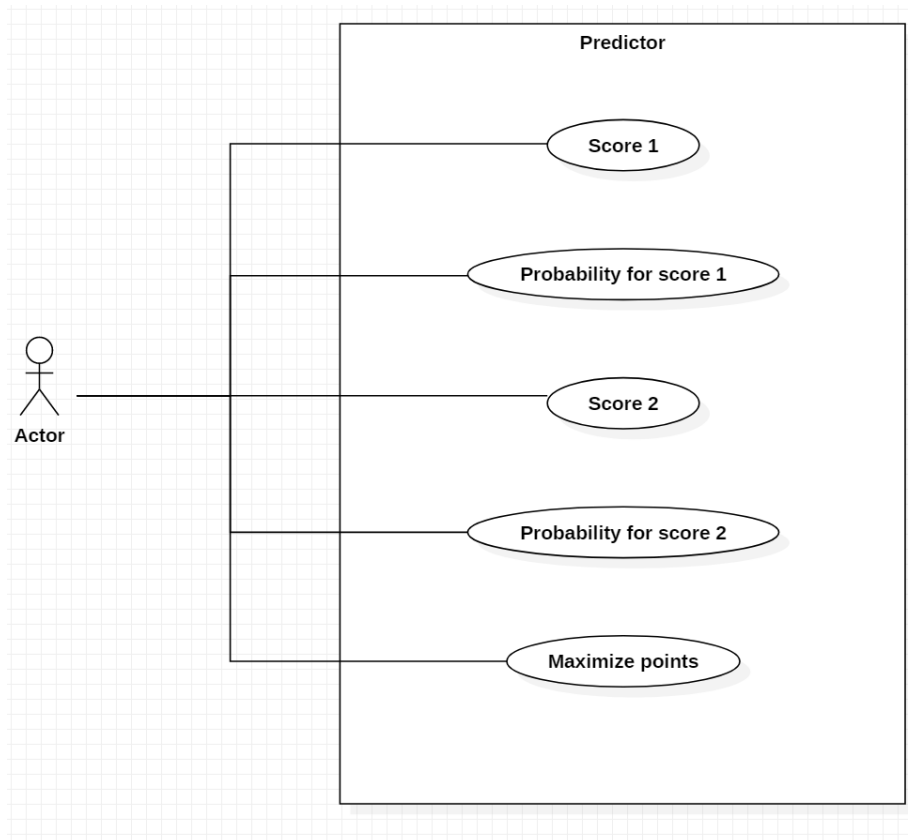
## Result

As in the introduction already described, predicting scores are pure luck. If you train your neural network for few more epochs, you will get totally different results. Nevertheless, this project was really interesting and exciting for me.

## UML

All diagrams were created by starUML. This software is for free and quite easy to get into.

### Use Case Diagram



*Figure 1: Use Case Diagram*

The actor wants to get predictions for the scores for the next match. He also might want to know how confident the predictions are.

## Package Diagram

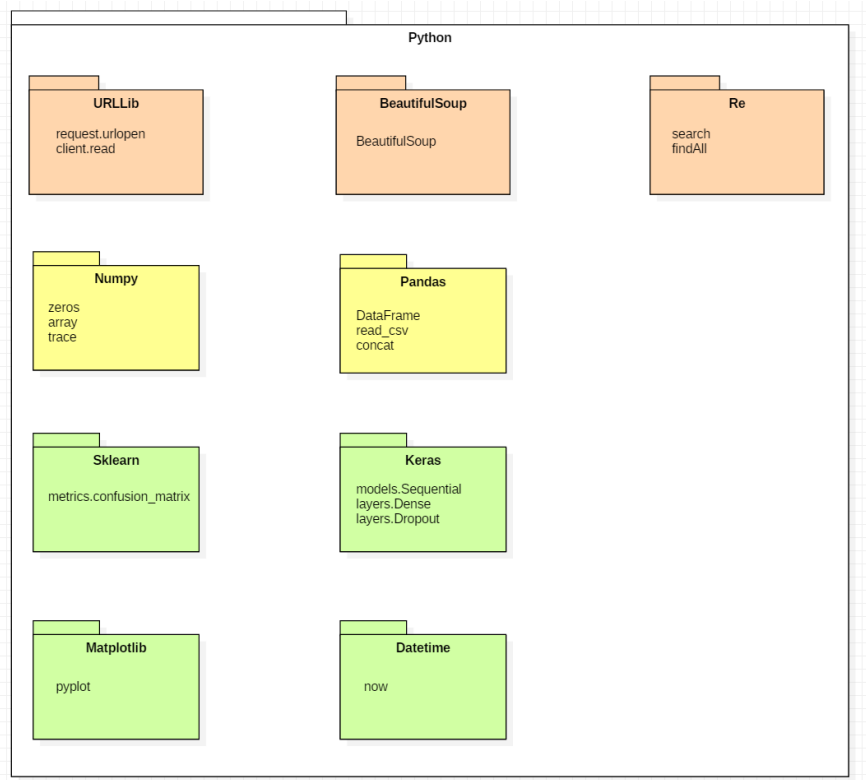


Figure 2: Package Diagram

These are the necessary packages in python that you need to run the code. Look at the requirements.txt file in the project folder if packages' versions are needed.

## Activity Diagram – Web Scraper

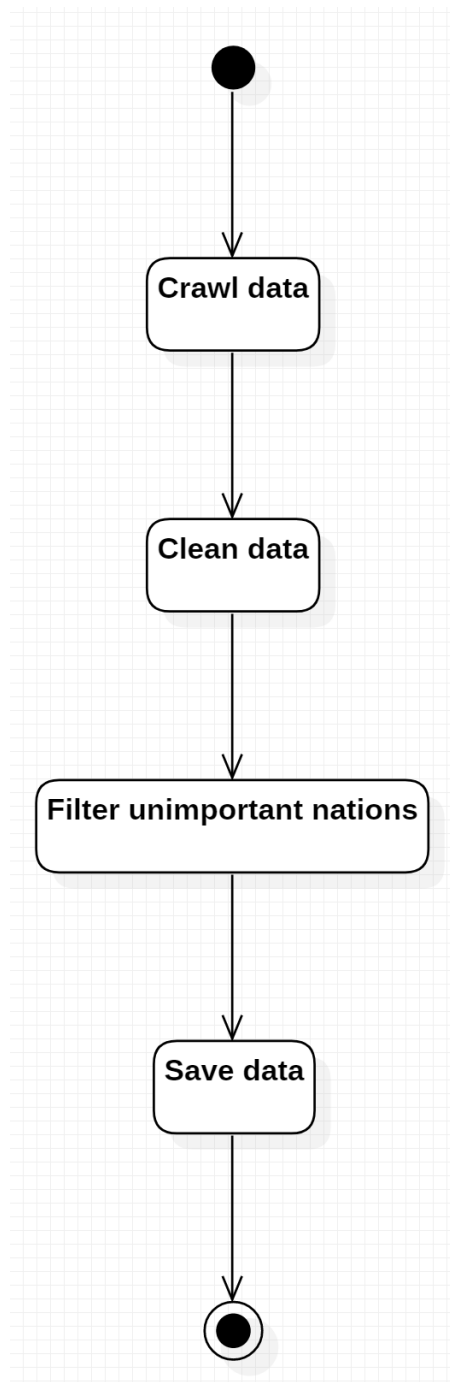


Figure 3: Activity Diagram – Web Scraper

This diagram shows the process of getting clean data from the web. This is one major part of this project. The quality and amount of data determine how good our predictions will be.

## Activity Diagram – Predictor

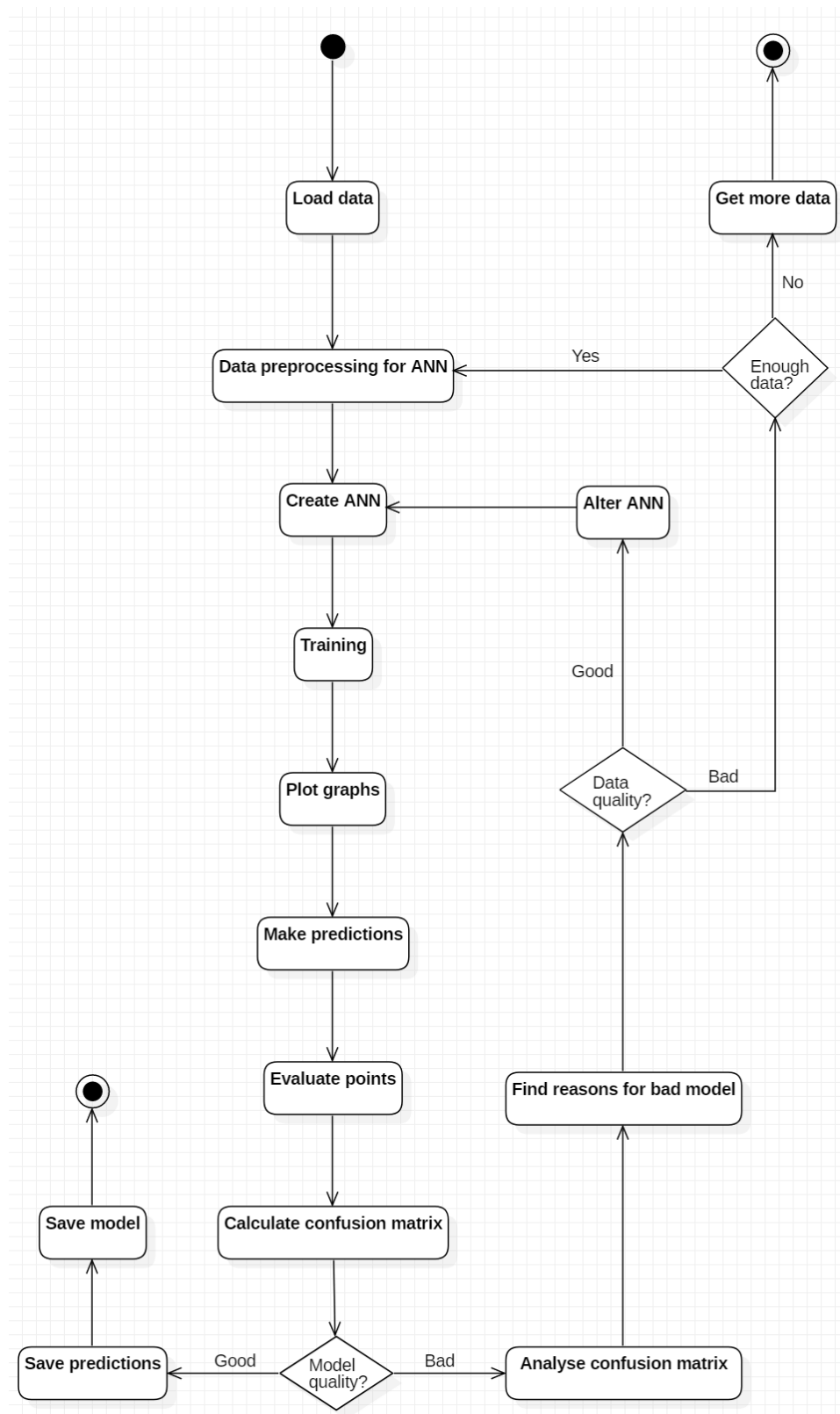


Figure 4: Activity Diagram – Predictor

This activity diagram shows the complex steps of the predictor.



## Sequence Diagram

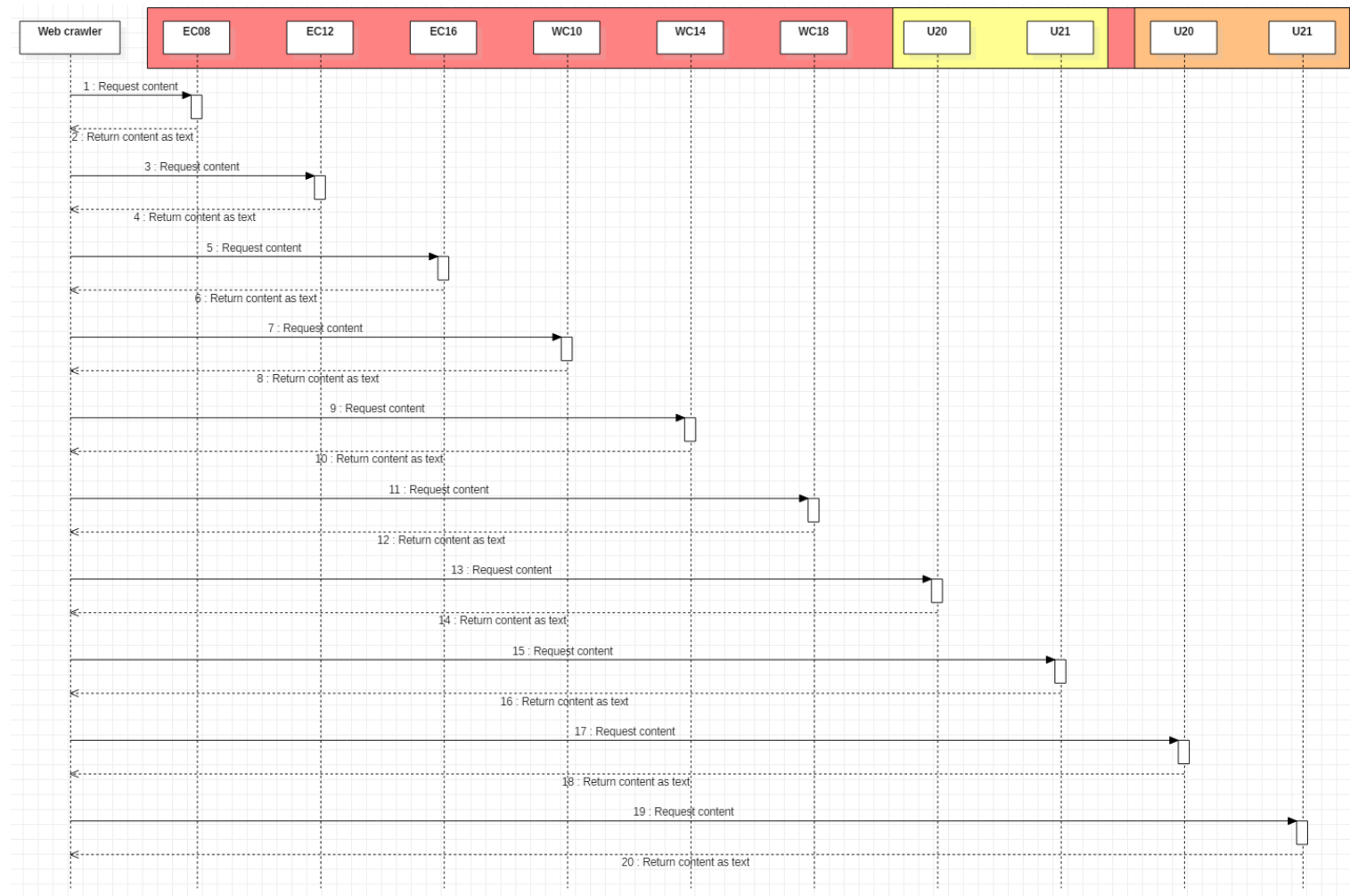


Figure 5: Sequence Diagram

This Sequence diagram shows the requests to the websites. Sometimes websites block frequent requests. Fortunately, I have not made this experience. But if it is the case, then use the sleep command to interrupt the requests.

# Metrics

## Sonar Cloud

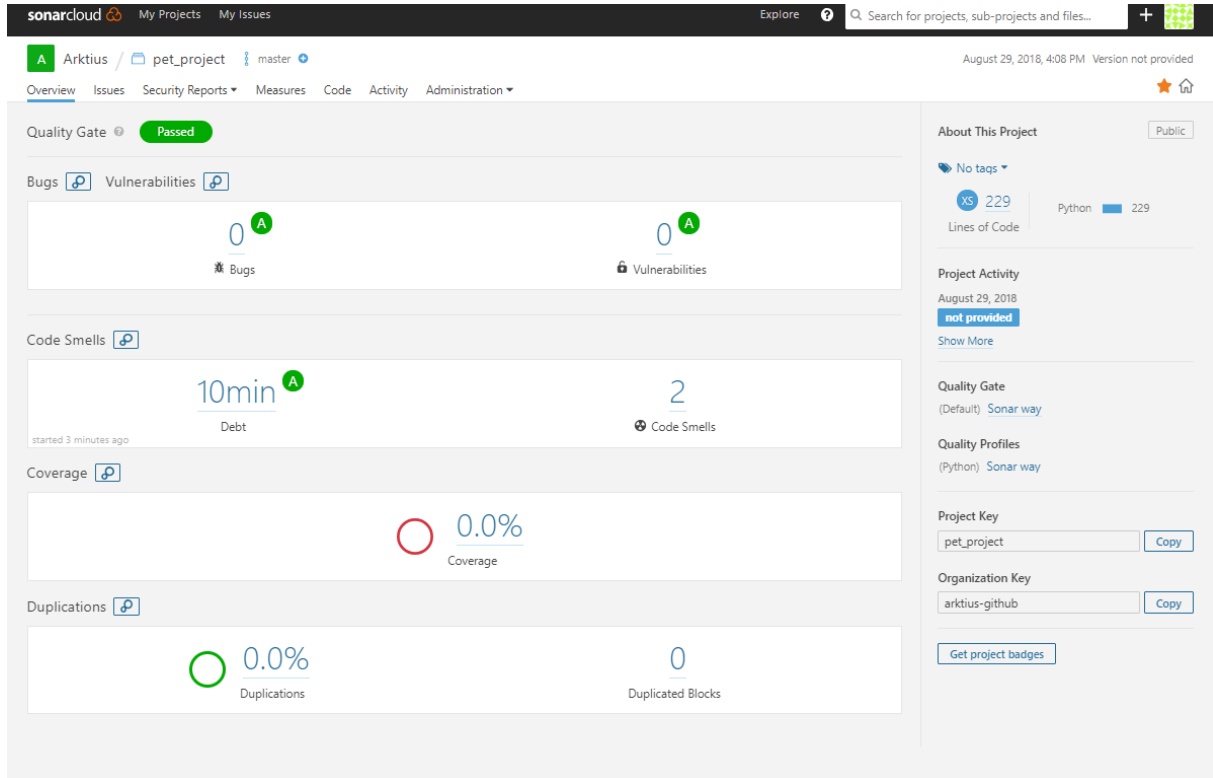


Figure 6: Sonar Cloud result

The result is pretty fine. Code smells were detected, because I commented two packages out which might be needed in the future.

## Pylint

```
M:\Stadium\Data_Science_Master\S3_WS1819\Advanced Softwareengineering\Pet_Project>pylint --output-format=colored main.py
***** Module main
main.py:7:123: C0303: Trailing whitespace (trailing-whitespace)
main.py:7:0: C0301: Line too long (123/100) (line-too-long)
main.py:8:82: C0303: Trailing whitespace (trailing-whitespace)
main.py:10:79: C0303: Trailing whitespace (trailing-whitespace)
main.py:37:45: C0303: Trailing whitespace (trailing-whitespace)
main.py:38:39: C0326: Exactly one space required after comma
print("\nDownloads have been finished.",time.time()-start)
      ^ (bad-whitespace)
main.py:46:8: C0326: Exactly one space required after comma
df[['s1','s2','year']] = df[['s1','s2','year']].astype('int')
      ^ (bad-whitespace)
main.py:46:13: C0326: Exactly one space required after comma
df[['s1','s2','year']] = df[['s1','s2','year']].astype('int')
      ^ (bad-whitespace)
main.py:46:33: C0326: Exactly one space required after comma
df[['s1','s2','year']] = df[['s1','s2','year']].astype('int')
      ^ (bad-whitespace)
main.py:46:38: C0326: Exactly one space required after comma
df[['s1','s2','year']] = df[['s1','s2','year']].astype('int')
      ^ (bad-whitespace)
main.py:49:17: C0326: Exactly one space required after comma
df = df[~df[['n1','n2','s1','s2','year']].duplicated()]
      ^ (bad-whitespace)
main.py:49:22: C0326: Exactly one space required after comma
df = df[~df[['n1','n2','s1','s2','year']].duplicated()]
      ^ (bad-whitespace)
main.py:49:27: C0326: Exactly one space required after comma
df = df[~df[['n1','n2','s1','s2','year']].duplicated()]
      ^ (bad-whitespace)
main.py:49:32: C0326: Exactly one space required after comma
df = df[~df[['n1','n2','s1','s2','year']].duplicated()]
      ^ (bad-whitespace)
main.py:52:12: C0326: Exactly one space required after comma
del_unimp(df,10)
      ^ (bad-whitespace)
main.py:55:22: C0326: No space allowed around keyword argument assignment
df.to_csv(path_or_buf = 'soccer_results_all.csv',sep = ';', index = False)
      ^ (bad-whitespace)
main.py:55:48: C0326: Exactly one space required after comma
df.to_csv(path_or_buf = 'soccer_results_all.csv',sep = ';', index = False)
      ^ (bad-whitespace)
main.py:55:53: C0326: No space allowed around keyword argument assignment
df.to_csv(path_or_buf = 'soccer_results_all.csv',sep = ';', index = False)
      ^ (bad-whitespace)
main.py:55:66: C0326: No space allowed around keyword argument assignment
df.to_csv(path_or_buf = 'soccer_results_all.csv',sep = ';', index = False)
      ^ (bad-whitespace)
main.py:56:46: C0326: No space allowed around keyword argument assignment
df[df['kind'] == "wm2018"].to_csv(path_or_buf = 'soccer_results_wc18.csv',sep = ';', index = False)
      ^ (bad-whitespace)
main.py:56:73: C0326: Exactly one space required after comma
df[df['kind'] == "wm2018"].to_csv(path_or_buf = 'soccer_results_wc18.csv',sep = ';', index = False)
      ^ (bad-whitespace)
main.py:56:78: C0326: No space allowed around keyword argument assignment
df[df['kind'] == "wm2018"].to_csv(path_or_buf = 'soccer_results_wc18.csv',sep = ';', index = False)
      ^ (bad-whitespace)
main.py:56:91: C0326: No space allowed around keyword argument assignment
df[df['kind'] == "wm2018"].to_csv(path_or_buf = 'soccer_results_wc18.csv',sep = ';', index = False)
      ^ (bad-whitespace)
main.py:57:46: C0326: No space allowed around keyword argument assignment
df[df['kind'] != "wm2018"].to_csv(path_or_buf = 'soccer_results.csv',sep = ';', index = False)
      ^ (bad-whitespace)
main.py:57:68: C0326: Exactly one space required after comma
df[df['kind'] != "wm2018"].to_csv(path_or_buf = 'soccer_results.csv',sep = ';', index = False)
      ^ (bad-whitespace)
main.py:57:73: C0326: No space allowed around keyword argument assignment
df[df['kind'] != "wm2018"].to_csv(path_or_buf = 'soccer_results.csv',sep = ';', index = False)
      ^ (bad-whitespace)
main.py:57:86: C0326: No space allowed around keyword argument assignment
df[df['kind'] != "wm2018"].to_csv(path_or_buf = 'soccer_results.csv',sep = ';', index = False)
      ^ (bad-whitespace)
main.py:60:0: C0305: Trailing newlines (trailing-newlines)
main.py:19:0: W0401: Wildcard import functions (wildcard-import)
main.py:22:0: C0103: Constant name "df" doesn't conform to UPPER_CASE naming style (invalid-name)
main.py:31:0: C0103: Constant name "start" doesn't conform to UPPER_CASE naming style (invalid-name)
main.py:49:0: C0103: Constant name "df" doesn't conform to UPPER_CASE naming style (invalid-name)
main.py:16:0: W0611: Unused numpy imported as np (unused-import)
main.py:19:0: W0614: Unused import OneHotEncoder from wildcard import (unused-wildcard-import)
main.py:19:0: W0614: Unused import re from wildcard import (unused-wildcard-import)
main.py:19:0: W0614: Unused import plot.nn from wildcard import (unused-wildcard-import)
main.py:19:0: W0614: Unused import plt from wildcard import (unused-wildcard-import)
main.py:19:0: W0614: Unused import LabelEncoder from wildcard import (unused-wildcard-import)
main.py:19:0: W0614: Unused import BeautifulSoup from wildcard import (unused-wildcard-import)
main.py:19:0: W0614: Unused import result from wildcard import (unused-wildcard-import)
main.py:19:0: W0614: Unused import prepro.nn from wildcard import (unused-wildcard-import)
main.py:19:0: W0614: Unused import np from wildcard import (unused-wildcard-import)
main.py:19:0: W0614: Unused import StandardScaler from wildcard import (unused-wildcard-import)
main.py:19:0: W0614: Unused import urllib from wildcard import (unused-wildcard-import)
main.py:17:0: C0411: standard import "import time" should be placed before "import pandas as pd" (wrong-import-order)

-----
Your code has been rated at -8.75/10 (previous run: -8.75/10, +0.00)
```

Figure 7: Pylint

Pylint was more informative than Sonar. Pylint has shown me way more things that need to be corrected such as all the whitespaces after a comma, name styles and unused imports.

# Clean Code Development

## Version Control

Since the beginning of this project, git was used as version control system. To handle the operations easily, I have been using GitHub for Desktop all the time. This is a free software where you automatically see changes. Operations such as push, pull and commit are done by a simple click on the belonging button.

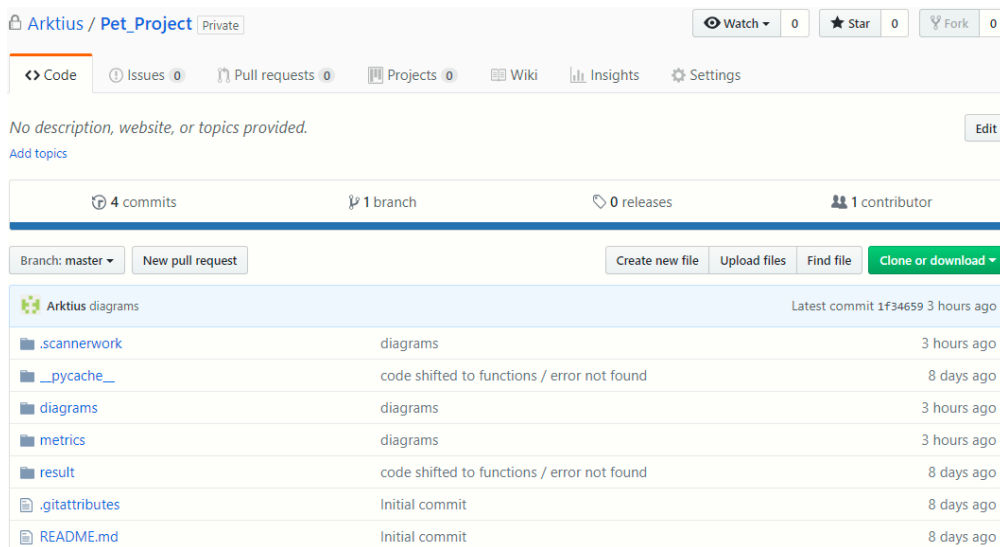


Figure 8: GitHub Repository

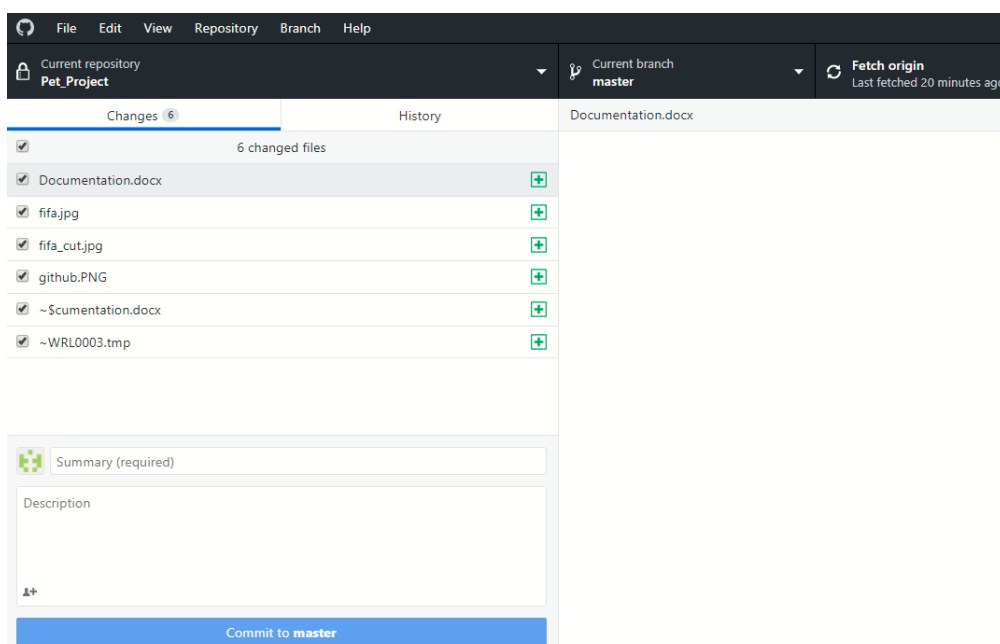


Figure 9: GitHub for Desktop

## Source Code Conventions

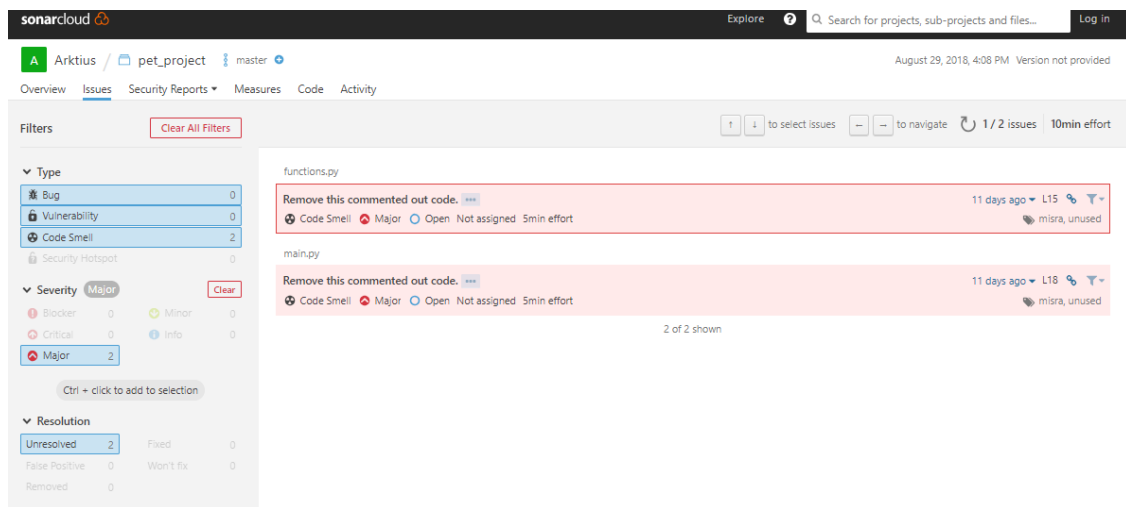


Figure 10: Sonar Cloud detects code smells

## Precise Naming

In this project, functions and variables were considered to be named as precise but still simple as possible. This function iterates through all domains or URLs called. There is one mistake highlighted in red. The iterator `i` should have named clearer. Since `cup` only contains URLs, the iterator could be named `'url'`. Whereas the well-named variables are highlighted in green.

```
def crawl_cups(df):  
    print() #prints empty line  
  
    domain = r'http://www.weltfussball.de/alle_spiele/' #domain of matches  
    cup = [] #list to store cups  
  
    #sub-URLs European / World Cup  
    cup.append(domain + r'em-2016-in-frankreich/')  
  
    #go through all European / World Cups  
    for i in range(0,len(cup)):  
  
        year = int(re.search('\d+',cup[i]).group())  
        kind = cup[i][39:41]  
  
        #crawl website  
        client = urllib.request.urlopen(cup[i])  
        page = client.read()  
        soup = BeautifulSoup(page,"lxml")
```

## Removing unnecessary comments

Since I have been working as a software developer in a company for more than a year, I tend to write more comments than might be needed. More comments lead to less questions from colleagues later on, but clear given names for variables and a well-documented paper make comments superfluous. Nevertheless, comments before loops and for bunches of code lines are reasonable. Bad examples are again highlighted in red, whereas good ones are in green.

```
def crawl_cups(df):
    print() #prints empty line

    domain = r'http://www.weltfussball.de/alle_spiele/' #domain of matches
    cup = [] #list to store cups

    #sub-URLs European / World Cup
    cup.append(domain + r'em-2016-in-frankreich/')

    #go through all European / World Cups
    for i in range(0,len(cup)):

        year = int(re.search('\d+',cup[i]).group())
        kind = cup[i][39:41]

        #crawl website
        client = urllib.request.urlopen(cup[i])
        page = client.read()
        soup = BeautifulSoup(page,"lxml")
```

## Vertical Separation

Variables should be used immediately after they are created. In this function the variables `year` and `kind` are good examples where this is not happened. They could have created inside the `if` in the inner for loop where they will be used.

```
#go through all European / World Cups
for i in range(0,len(cup)):
```

```
    year = int(re.search('\d+',cup[i]).group())
    kind = cup[i][39:41]
```

```
    #crawl website
    client = urllib.request.urlopen(cup[i])
    page = client.read()
    soup = BeautifulSoup(page,"lxml")
```

```
    #go through all matches in a cup
    for tr in soup.findAll('table',{'class': r"standard_tabelle"})[0].findAll('tr'):
        td = tr.findAll('td')
        if len(td) == 8:
            year = int(re.search('\d+',cup[i]).group())
            kind = cup[i][39:41]
            score = re.findall('\d+',td[5].getText())
            df.loc[len(df)] = [td[2].getText(),td[4].getText(),score[0],score[1],year,kind]
```

```
    #translate into english for printing
    if kind == 'em':
        kind = kind.replace('em','EC')
    else:
        kind = kind.replace('wm','WC')
```

```
    print("Data of {}-{} were downloaded.".format(kind,year))
```

## Continuous Delivery

Arktius / Pet\_Project

build failing

Current

Branches

Build History

Pull Requests

✓ master	Arktius	updated req.txt	→ #6 passed	→ 20558c5	1 min 45 sec	5 minutes ago
✗ master	Arktius	travis test works	→ #5 failed	→ 2a82ee9	1 min 46 sec	12 minutes ago
✓ master	Arktius	test file for travis	→ #4 passed	→ 5b68386	1 min 8 sec	18 minutes ago
✗ master	Arktius	travis python added	→ #3 failed	→ dda7f80	1 min 18 sec	34 minutes ago
✗ master	Arktius	travis updated	→ #2 failed	→ c872138	1 min 36 sec	38 minutes ago
✗ master	Arktius	travis yml file has being created	→ #1 failed	→ fea344f	15 sec	41 minutes ago

Figure 11: Build history in Travis-CI

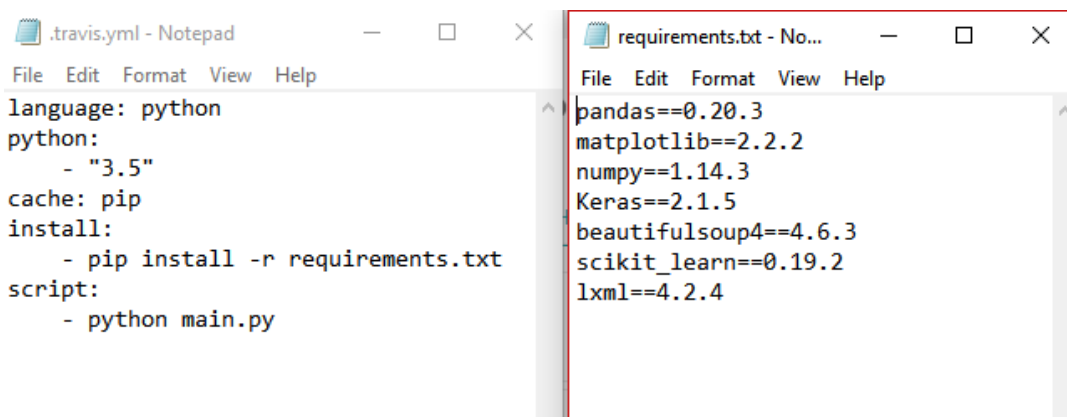


Figure 12: Needed files for Travis-CI

In Order to get the requirements of a project you can use the python package pipreqs. It can be installed by typing the following line in your terminal:

Pip install pipreqs

Navigate then to the folder above your project and run this command:

pipreqs --force Pet\_Project

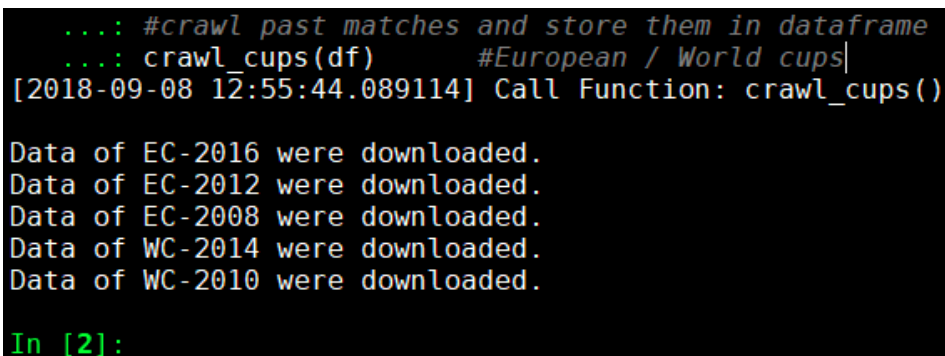


## AOP

Aspect Oriented Programming is applied by so called decorators in Python. These decorators are high order functions that wrap their arguments with code and usually precede a function. An important information for the user are the responses while the program is downloading data. Therefore, the timestamp could be printed out while data are being downloaded.

```
def time_execution(f):
    def wrapped(*args, **kws):
        now = datetime.datetime.now()
        print('[ ' + str(now) + ' ] Call Function: ' + f.__name__ + '()')
        return f(*args, **kws)
    return wrapped

@time_execution
def crawl_cups(df):
    #download data...
```



```
...: #crawl past matches and store them in dataframe
...: crawl_cups(df)           #European / World cups|
[2018-09-08 12:55:44.089114] Call Function: crawl_cups()

Data of EC-2016 were downloaded.
Data of EC-2012 were downloaded.
Data of EC-2008 were downloaded.
Data of WC-2014 were downloaded.
Data of WC-2010 were downloaded.

In [2]:
```

Figure 13: Decorator in usage

## DSL

DSL (Domain Specific Languages) are languages that are used to speak with machines or computers in a certain domain. SQL (Structured Query Language) is a well-known example for DSL to communicate with databases, especially relational databases.

Since my last project at work, I have been working with Neo4j. Neo4j is a graph-based database. That means, that you create nodes, which are connected to other nodes via edges. Both nodes and edges have attributes such as ID, name or certain values.

In the following code snippet, we will create an employee with skills, which he is experienced in. The code is written in Python using the package py2neo, that connects the python environment to the database.

```
#create employee node
graph.run("CREATE (:employee{employee: 'e_0', id: 0 })")

#create skills of employee
skill_list = ['Linux','C++','Python'] #only 3 skills for simplicity
_ = [graph.run("CREATE (:skill{skill: '%s'})" % (sk) for sk in skill_list)]

#connect both together
for skill in skill_list():
    graph.run("MATCH(n:employee),(m:skill) WHERE n.id='0' AND m.property='%s' MERGE (n)-[r:CAN{value: '1'}]->(m) " % (skill))
```



Figure 14: Neo4j - Employee 0 is experienced in certain skills

# Functional Programming

## Final Data Structures

Final data structures as known in other programming languages as final variables (Java) or constants (C++) are not implemented in Python. So, assign a value to a variable only once and never change it again.

## Side effect free functions

Since I only use each functions once, all functions are nearly side effect free. There are few exceptions where I have been using methods of Pandas Dataframes, but these calls are irreplaceable.

## High order functions – functions as parameters and return values

High order functions accept other functions as arguments or return functions as values. Decorators, as listed above in section AOP, are high order functions and return functions as a value.

## Closures

A Closure is a function object that remembers values in enclosing scopes even if they are not present in memory. To avoid the usage of global variables, Closures can be used. [\[source\]](#)

Example:

```
# Python program to illustrate
# nested functions
def outerFunction(text):
    text = text

    def innerFunction():
        print(text)

    innerFunction()

if __name__ == '__main__':
    outerFunction('Hey!')    #prints 'Hey!'
```

## Anonymous Functions

Anonymous functions are functions without a name. You can use the keyword `lambda` for this in Python. I could have used a `lambda` function to save multiple rows.

Usage of a `lambda` function in project:

```
def umlaute(df):
    #replace umlaute
    df[['n1','n2']] = df[['n1','n2']].replace(to_replace = 'Rumänien', value ='Rumaenien')
    df[['n1','n2']] = df[['n1','n2']].replace(to_replace = 'Österreich', value ='Oesterreich')
    ...

    #code with lambda function
    #replace = ['Rumänien',...]
    #value = ['Rumaenien',...]
    #f = lambda replace,val: df[['n1','n2']].replace(to_replace = replace, value = val)
    #df[['n1','n2']] = f(replace,value)
```

## Logical Solver

A logical solver is usually used in games such as Rock-Paper-Scissors or Tic-Tac-Toe, where you know the outcome depending on the turn and other possible variables. You can also predict the possibility, whether you'll win or not. The course of a Tic-Tac-Toe match can be displayed as a tree of possible turns. In Rock-Paper-Scissors it's even easier. You can directly get out, whether someone is going to win or lose against a certain turn, or the match ends in a draw.

In my program, logical programming is used to give points based on the predicted scores and the actual scores.

predicted score is actual score - 3 points  
score tendency is correct - 2 points  
predicted the winner correctly - 1 point

```
def result(fyp,fyt):  
    points = 0  
    for match in range(0,len(fyt),2):  
        if fyp[match:match+2] == fyt[match:match+2]:  
            points += 3  
        elif np.diff(fyp[match:match+2]) == np.diff(fyt[match:match+2]):  
            points += 2  
        elif (fyp[match] > fyp[match+1]) == (fyt[match] > fyt[match+1]):  
            points += 1  
  
    return points
```

## Scala

These two links helped me to install and get into Scala:

- <https://www.scala-lang.org/download/>
- <https://docs.scala-lang.org/getting-started-sbt-track/getting-started-with-scala-and-sbt-on-the-command-line.html>

After installing sbt and generating this Hello World program, you can see the print out.

```
[info] downloading https://repo1.maven.org/maven2/org/typelevel/cats-macros_2.12/1.1.0/
[info] downloading https://repo1.maven.org/maven2/org/typelevel/machinist_2.12/0.6.2/ma
[info] downloading https://repo1.maven.org/maven2/org/typelevel/cats-kernel_2.12/1.1.0/
[info] downloading https://repo1.maven.org/maven2/org/typelevel/cats-core_2.12/1.1.0/ca
[info] [SUCCESSFUL ] org.typelevel#cats-macros_2.12;1.1.0!cats-macros_2.12.jar (539ms)
[info] [SUCCESSFUL ] org.typelevel#cats-kernel_2.12;1.1.0!cats-kernel_2.12.jar (676ms)
[info] [SUCCESSFUL ] org.typelevel#machinist_2.12;0.6.2!machinist_2.12.jar (759ms)
[info] [SUCCESSFUL ] org.typelevel#cats-core_2.12;1.1.0!cats-core_2.12.jar (1170ms)
[info] Done updating.
[info] Compiling 1 Scala source to C:\Users\Admin\Documents\GitHub\Pet_Project\hello-wo
[info] Non-compiled module 'compiler-bridge_2.12' for Scala 2.12.6. Compiling...
[info]   Compilation completed in 8.071s.
[info] Done compiling.
[info] Packaging C:\Users\Admin\Documents\GitHub\Pet_Project\hello-world\target\scala-2
[info] Done packaging.
[info] Running Main
Hello, World!
[success] Total time: 15 s, completed 11-Oct-2018 11:14:16
1. Waiting for source changes... (press enter to interrupt)

sbt:hello-world>
```

Figure 15: Hello world in Scala

The following code snippet in Scala sums up all arguments. This could be utilized to sum up the points you have got during the World Cup 2018.

```
sbt:hello-world> ~run 0 0 1 2 0 3 1 2 3
[info] Running Main 0 0 1 2 0 3 1 2 3
The sum of my arguments is: 12
[success] Total time: 0 s, completed 11-Oct-2018 11:34:33
1. Waiting for source changes... (press enter to interrupt)

sbt:hello-world>
```

Figure 16: Sum up all arguments

The code is located under the following path:

...\Pet\_Project\hello-world\src\main\scala

```
object Main {  
  def main(args: Array[String]) {  
    try {  
      val elems = args map Integer.parseInt  
      println("The sum of my arguments is: " + elems.foldRight(0) (_ + _))  
    } catch {  
      case e: NumberFormatException =>  
        println("Usage: scala Main <n1> <n2> ... ")  
    }  
  }  
}
```

[Source: scala-lang.org](http://scala-lang.org)