The Programming for Life Sciences Assessment Matrix

The assessment of the final assignment of the course Programming for Life Sciences is done using the assessment matrix/rubric below. The matrix consists of three parts, Programming, Documentation, and Practical Work. Each part lists aspects of the work, which are given as rows, with the level indicators written in the cells of that row. The right-most level that applies to the work is marked. The column that is named 'sufficient' lists the minimal requirement for passing. If any aspect of the work falls in the category 'insufficient', the course is failed. If an aspect of Programming is not naturally part of the final assignment, the assessors may base the grading on the other aspects and on the performance in intermediate assignments.

Students are encouraged to use the matrix to assess their own level and progress during the course, and to set targets for improvement. During the course, teaching assistants are available to provide feedback on the current performance in relation to the matrix.

insufficient	sufficient	(very) good	Snake charmer	pythonista

Programming						**60%**
Programs	can execute statements in the interpreter can print “Hello World” in a script	can write a script as a collection of statements can find and use external modules	can write a program as logical entity to execute a task with clear separation between the process and input data	can write a module to organize and reuse code	can organize and distribute code as package	
Functions	Copies similar code within a script	can write simple functions to organize code and avoid code duplication understands and respects the concept of variable scope	can generalize functions and separate code in general and specific functions can write functions as logical, encapsulated units, using arguments and return values for input and output	can write functions that handle variable argument and keyword argument lists can write and use generators can use functions as first class objects can write closures	can write and use coroutines	
Flow control	can use **if/elif/else** clauses for conditional execution of code can use **for** loops for repeated actions on collections can use **while** loops for repeated conditional execution	can use **break** and **continue** statements in for and while loops to optimize and interrupt repeating actions when needed can use **enumerate**, **zip** in conjunction with loops	Boolean arithmatics Basic use of try-except	can use **for/else** and **while/else** clauses can use list comprehensions for conversion and filtering of collections can use recursion advanced try-except (handling specific errors, and else/finally)	can use generator and dict comprehensions can use a functional programming paradigm	
Error handling	can recognize and solve common SyntaxError, NameError and TypeError instances	can use the traceback to locate and solve (non-semantic) errors can understand and locate pervasive and obvious semantic errors	can understand and solve semantic errors using print statements when/where needed	uses unittests to locate and solve errors can raise appropriate errors when needed	can use a debugger to identify, locate and solve errors can write new exceptions when needed	
Input/output	can use print() to print can use input() to obtain input	can extract data from text files to be used in code can write data to text files	can read and parse data from text files and store it in correct python data structures can use string formatting to write structured output files can use module functions for reading and writing files.	can read from and write to binary file types can use streams for reading and writing	can read from and write to network connections	
Data	can use variables can use simple data types (**str**, **int**, **float**, **bool**) and operations on these can use collections (**list**, **tuple**, **dict**, **set**) and operations on these	can use nested data types can use datatypes from modules can interconvert between different data types	can match data structure with algorithm can work with multi-dimensional arrays	can write classes to organize data and associated methods (OOP) can use advanced data types, like from the collections module	Can use inheritance Can use abstract base classes	

Documentation						**25%**
Help	Can read a help page provided by others	can find and read help provided within Python, using help() can find and read help online in the Python and NumPy documentation	can implement solutions to problems based on reading internal and external Python and Numpy documentation can read, understand and use solutions to problems from generic sources (internet, StackOverflow) to solve own problems	can reformulate and generalize problems can help others to understand and solve problems	Can guide others in their development	
Code	Doesn’t use comments	uses comments to describe what the code does or should do uses variable and function names that reflect content can read code of this level when PEP8 formatted	uses coding style in line with PEP8 writes docstrings for every function and for every class, program or module uses comments to explain design choices in code	writes a README and/or help that explains how the program is used	includes the scope and limitations of the program in the README and provides information to support alternative uses or modification of parts of the code	
Report	describes the context, the problem and break down of the problem	reflects on the steps required to solve the problem describes the code and the structure of the code reflects on the limitations of the code	explains the structure of the code and the design choices made explains how the code solves the problem reflects on the range of problems the code can be applied to and possible extensions	Indicates which parts of the code need modification to make the code applicable for other problems.	explains or shows how to modify the code to become applicable for other problems	

Practical work						**15%**
Effort		actively tries to solve problems before asking assistance	actively searches for solutions online before asking assistance			
Skill	writes code without testing	tries code snippets before implementing	uses a systematic approach to try, implement and test code			
Attitude	is disruptive and/or offensive uses offensive statements/comments in code		is respectful and cooperative assists peers to understand code	assists peers to solve problems		