Exam for the course DAT171 Object oriented programming in Python

Time: 15th March 2022 8:30-13:30

Teachers: Thomas Svedberg (031-772 1522), Mikael Öhman (073-6837674)

Permitted aids: Cay Horstmann: Python for everyone. Manuals and lecture notes are available on the computers.

Teacher will visit the rooms: Around 10:00 and 12:00

Formalities: In each file you hand in, you should write your examination code as well as the computer "number".

The documentation and lectures notes are available on `C:\__EXAM__`. Verify that these files are there immediately. Handing in the code should be done under the same folder: `C:\__EXAM__\Assignments\`, when you are finished. If you do not store the files here, they are not included in the exam. Save the files for each question in the appropriate folder, e.g. `C:\__EXAM__\Assignments\Question1\`. *Make sure you only hand in one solution for each question, otherwise you will receive **zero** points.* Complex sections of your code should have a descriptive comment of what is achieved.

When you finish the exam you should log out and fill in the (empty) exam cover page like normal and hand this in at the end of the exam.

Corrections: The results will be announced through Ladok, on or before March 31st. The review will be the same day (March 31st) 12:20-13:00 at Hörsalsvägen 7b, 3:rd floor.

Grading: There is a total of 25 points which yields grades at the standard 40%, 60%, and 80% limits:

```python
def grade(points):
    if points >= 20:
        return 'Grade 5'
    elif points >= 15:
        return 'Grade 4'
    elif points >= 10:
        return 'Grade 3'
    else:
        return 'Fail'
```

Question 1: Listing files and sizes (6p)

To get an overview of how much space your files use you want to create a small directory listing application.

Part A (1.5p)

Write a function `read_directory` that takes a directory path as input and returns a dictionary with all files and their corresponding file size in bytes.

Hint: Look for useful functions in `os` and `os.path`.

Example:

```
path = 'example_directory'  # Could also be '../path/to/directory'
dinfo = read_directory(path)
print(dinfo)
```

Output:

```
{'comic image.jpg': 79538, 'baby.gif': 5880697, 'a_tiny_file.txt': 16,
 'PBF248-Transmission.png': 278912}
```

Part B (2p)

Sizes of computer files are typically listed in terms of Kibi-, Mebi- and Gibi-bytes. Shortened as KiB, MiB, GiB respectively.

- 1 KiB = 1024 B
- 1 MiB = 1024 KiB = 1024^2 B
- 1 GiB = 1024 MiB = 1024^3 B

I.e. simply using 1024 instead of 1000 (like with Kilo, Mega, Giga)

Write a helper function `format_bytes` that takes a number of bytes as input and converts it to Kibi, Mebi or Gibi when the values are larger than 1024, 1024^2, 1024^3 respectively. The method must return a string with the value and a unit. It must avoid showing decimals when the number is an integer, and show a maximum of 2 decimals. See examples below for the expected behaviour.

Example:

```
print(format_bytes(42))
print(format_bytes(1024))
print(format_bytes(12288))
print(format_bytes(200000))
print(format_bytes(1234567890))
```

Output:

```
42B
1KiB
12KiB
195.31KiB
1.15GiB
```

Part C (2.5p)

Write a function `show_directory_info` that takes the path and directory information from Part A and displays all the sizes like in the example below.

- The total size should be printed next to the path on the first line.
- For readability, the directory content should be displayed with a indentation using `|--` and `+--` on the last line.
- Sort the files according to size (largest to smallest).

Example:

```
show_directory_info(path, dinfo)
```

Example:

```
example_directory: 5.95MiB
|-- baby.gif: 5.61MiB
|-- PBF248-Transmission.png: 272.38KiB
|-- comic image.jpg: 77.67KiB
+-- a_tiny_file.txt: 16B
```

Question 2 Fitting data (6p)

Note: In this task NumPy and SciPy **must** be used for any array-type of numerical data as well as for doing the interpolation etc.

You have performed an experiment and want to analyse the data you collected. More precisely you want to find out if the data can be described using a function.

Part A (0.5p)

Read the data from the file named "values.txt" into NumPy-arrays x and y. The results are stored in text format, where the first column is the x-values, and the second column the y-values.

Part B (2.5p)

Using a suitable method from SciPy, construct splines of *order* 2 and smoothing set to **None**, **0** and **15** respectively, using the given x-y-data.

In a 2-row by 1-column subplot plot, plot the original x-y-values using point markers and the derived splines using lines in the top plot.

Note: Remember that you should have (a lot) more data values for the splines than for the original data!

For full points, the plot must have proper x- and y-labels, a legend describing the different curves as well as a suitable title.

What seems to be the best smoothing for the spline? (Answer with a comment in the code!)

Part C (3p)

Looking at the results from Part A, you decide the curve probably corresponds well to a polynomial. Let's see just how well the data can be fitted to different polynomials.

With polynomials of order 2, 3, 4 and 5, and using a suitable method from NumPy or SciPy, find the polynomial parameters (A-E below) that gives the smallest least-squares error for the given x-y data.

For example, a fifth order polynomial can be written as:

$$y = C_0 + C_1x + C_2x^2 + C_3x^3 + C_4x^4 + C_5x^5$$

For the four polynomials, print the resulting parameters (for example C_0 - C_5 for the fifth order polynomial above), with labels.

As for the splines, plot the data and the resulting polynomials in the lower subplot. Once again labels etc. are needed for full points.

What polynomial order is sufficient? (Answer with a comment in the code!)

Question 3: OneHotVector (6p)

In some applications there is a need for a special vector where only a single element is set to 1, the rest being zero. The is typically called a `OneHotVector`.

Part A (1p)

Create the class `OneHotVector` that takes a length and an index of the nonzero element.

Part B (0.5p)

Add a nice looking `repr` implementation.

Part C (0.5p)

Implement support for `len(x)` on your `OneHotVector`.

Part D (1p)

Implement support for scalar indexing (just getting the items, not assignment). Also add support so that negative indexes work (starting from the end).

Part E (1p)

Add the method `toarray` which returns the a NumPy array version of this vector. Have it take an optional argument of what datatype to use in the NumPy array, with a default value of `int`.

Part F (1p)

Implement support for a dot product between the vector and a matrix (NumPy array) by adding a `dot` method that computes $v \cdot M$ for a vector v and matrix M. Since where is just 1 nonzero value, this simply means extracting the corresponding row which can be done very efficiently.

Part G (1p)

Add bounds checking to Part A, D, F and raise an `IndexError` with appropriate messages when indices or dimensions don't match.

Example

```
import numpy as np

a = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])
b = np.array([[1,2,3], [4,5,6], [7,8,9], [10,11,12]])

v = OneHotVector(length=3, index=2)
w = OneHotVector(length=4, index=1)
u = OneHotVector(length=1000000000000, index=0)

print(u)

assert len(v) == 3

assert w[0] == 0
assert w[1] == 1
assert w[2] == 0

assert w[-3] == 1
assert w[-2] == 0

print(f'v as np.array: {v.toarray()}')
print(f'w as float np.array: {w.toarray(dtype=float)}')

print('v . a =', v.dot(a))
print('w . b =', w.dot(b))
```

Output

```
OneHotVector(length=1000000000000, index=0)
v as np.array: [0 0 1]
w as float np.array: [0. 1. 0. 0.]
v . a = [ 9 10 11 12]
w . b = [4 5 6]
```

Question 4: Text user interface (7p)

Similar to GUI widgets, one can quite easily create Text User Interfaces (TUI) as well. We can borrow the concept of a "Widget" from GUIs and create a small hierarchy of text widgets.

Part A (1p)

Create an abstract base class `TUIWidget` that stores the number of `columns` and `rows` of a text widget.

It should have a abstract method `render` that takes no arguments and returns a list of rendered strings (one for each row). Also add a (normal) method `show` that simply prints out the rendered lines.

Part B (1p)

Implement a subclass, `ProgressBar`, that only takes the number of columns, and an optional `progress` value with a default of zero. (The ProgressBar always has a height of 1).

Also add a method `set_progress` for updating the progress.

Implement rendering of strings by showing a row filling up with `#` proportional to the progress made, delimited by brackets, e.g with progress at 0.5:

```
[##########          ]
```

Part C (2p)

Implement another subclass, `TextField`, that takes the number of columns and rows, and an optional string (empty by default). When rendering, it should wrap the text around the given number of columns up until the number of rows. If the text isn't long enough, it should pad out the rows with empty lines. If the text is to long, the overflowing characters can simply be discarded.

Also add a method `set_text` for updating the text.

Part D (2p)

Implement yet another subclass, `Box`, that takes a reference to another widget and adds a nice border around it.

```
+--------------+
|Another widget|
|inside        |
+--------------+
```

Part E (1p)

Implement a subclass to `Box`, `GroupBox`, that still takes a widget but also a `name` as a string. It should reuse what `Box` already renders, only replacing the top section with a name inside brackets:

```
+-[Example]----+
|Another widget|
|inside        |
+--------------+
```

Example

```
p = ProgressBar(22)
p.show()
p = ProgressBar(22, 0.2)  # 10% progress
p.show()
assert p.render() == ['[####                  ]']

print()

t = TextField(20, 3, 'Learning Python sure is fun. Even the exam is fun! Not all text fits')
assert t.render()[0] == 'Learning Python sure'
b = Box(t)
b.show()

print()

g = GroupBox("Download", p)
p.set_progress(0.75)
g.show()

print()

g.name = 'Box in a box'
bg = Box(g)
bg.show()
```

Output

```
[                    ]
[####                ]

+--------------------+
|Learning Python sure|
| is fun. Even the ex|
|am is fun! Not all t|
+--------------------+

+--[Download]----------+
|[###############     ]|
+----------------------+

+------------------------+
|+--[Box in a box]------+|
||[################     ]||
|+----------------------+|
+------------------------+
```