Python for Text Analysis
Programming for Humanities and Social Sciences
Exam 2018-2019

Teachers:
Pia Sommerauer Chantal van Son

December 17, 2018

This exam marks the end of *Programming for Humanities and Social Sciences* (6 ECTS) and the first part of *Python for Text Analysis* (9 ECTS). It consists of five parts:

1. Booleans (15 points)
2. General knowledge about Python (24 points)
3. Spot the error (16 points)
4. Tracing (20 points)
5. Writing code (15 points)

The total number of points you can earn is 90. You will get 10 points for free. Your grade will be computed by dividing the total number of points by 10. For example: 99 points corresponds to a 9.9 out of 10.

1 Booleans

Please indicate for each of the cases whether they print `True` or `False`. Each question is worth 1 point.

1. ```
   print("hello" != "HELLO")
   ```

2. ```
   print(10 == "10")
   ```

3. ```
   print(type("animal") == str)
   ```

4. ```
   my_list1 = [1, 2, "c", "e", "c", 2]
   my_list2 = [1, 1, 2, "c", "e", "e"]
   print(len(my_list1) > len(my_list2))
   ```

5. ```
   my_list1 = [1 ,2, "c", "e", "c", 2]
   my_list2 = [1, 1, 2, "c", "e", "e"]
   print(set(my_list1) == set(my_list2))
   ```

6. ```
   shopping = {"chocolate": 3, "oranges": 10, "bananas": 3}
   print(3 in shopping or 21 in shopping)
   ```

7. ```
   names1 = {"Harry", "Ron", "Hermione"}
   names2 = {"Tom", "Bellatrix", "Peter"}
   print("Bella" not in names1 and "Bella" in names2)
   ```
   ```

8.
```
filepath1 = "../../Data/HuckFinn.txt"
filepath2 = "../Data/books/Mackbeth.txt"
print(filepath1.split("/")[2] == filepath2.split("/")[2])
```

9.
```
favorite_colors = ["blue", "green"]
new_favorite_colors = favorite_colors.append("purple")
print(new_favorite_colors == ["blue", "green", "purple"])
```

10.
```
a_name = "Tom Riddle"
a_name.replace("d", "g")
print(a_name == "Tom Riddle")
```

11.
```
food_dict = {"pizza_margherita": 5, "brownie": 10, "ice_cream": 4}
food_dict["lemon_meringue"] = 3
food_dict["brownie"] = 20
print(food_dict["brownie"] == 10)
```

12.
```
food_dict = {"pizza_margherita": 5, "brownie": 10, "ice_cream": 4}
food_dict["lemon_meringue"] = 3
food_dict["brownie"] = 20
print(len(food_dict) == 4)
```

13.
```
animals1 = ["dog", "cat", "hamster"]
animals2 = {"spider", "dragon_fly", "lady_bug", "spider"}
print(len(animals1) == len(animals2))
```

14.
```
word1 = "hello"
word2 = "WORLD"
print((3 * 5 + 4) < 20 or "ok" not in "book" or word1.upper() == word2)
```

15.
```
a_list = []
a_set = set()
a_dict = dict()
print(len(a_list) == len(a_set) == len(a_dict))
```

2 General knowledge about Python

Please answer the questions below. Each question is worth 2 points (total 24).

16. Explain the difference between the two operators == and =.

17. Can you loop over a string? If not, explain why not. If so, explain what happens if you loop over a string (provide an example).

18. What is the difference between local and global variables? Explain where they are defined and where they can be accessed.

19. Briefly explain the difference between a function definition and a function call.

20. What is the difference between using the `print()` function and a `return` statement in a function? Name 2 differences.

21. What properties apply to sets but not to lists? Name 3 properties.

22. What is the difference between using a combination of 'if-if' statements and using 'if-elif'? Explain using the example below.

```
# if-if
x = 5
if x > 1:
    print("Larger than 1")
if x > 3:
    print("Larger than 3")

# if-elif
x = 5
if x > 1:
    print("Larger than 1")
elif x > 3:
    print("Larger than 3")
```

23. Which Python containers (or parts of Python containers) do **not** accept mutable objects as elements? Name 2.

24. What does the list method `append` do? What does it return? You can use an example to illustrate your answer.

25. Which Python containers should you use to represent a single row of a CSV/TSV file? Name 2 containers.

26. What method from the `lxml.etree` module can you use to get a specific attribute value of an XML element? For example, how can you access the string "23/06/1912" from the 'person' element below (shown in the example XML element)?

27. What method from the `lxml.etree` module can you use to get all child elements with a specific tag of an XML element? For example, how can you access all 'profession' child elements from the 'person' element below?

```
# Example XML-element
<person born="23/06/1912" died="07/06/1954">
    <name>
        <first>Alan</first>
        <last>Turing</last>
    </name>
    <profession>computer scientists</profession>
    <profession>mathematician</profession>
    <profession>cryptographer</profession>
</person>
```

3 Spot the error

Below are eight pieces of code with a mistake in them. Please indicate the errors, and explain why they cause the code to break (you don't need to know the exact error message). Each question is worth 2 points (one for showing where the code breaks, one for the explanation), with a total of 16 points. For your convenience, we also printed line numbers next to each line of code. Please always **point to the exact location (i.e. not just to the line number).** An explanation of about one sentence is enough.

Example question: Where is the mistake and why does it break the code?

```
1 my_list = [1,2,3]
2 2nd_item = my_list[1]
```

Answer: The mistake is in the variable name `2nd_item` (line 2). Python doesn't allow for variable names to start with numbers.

28. Where is the mistake and why does it break the code?

```
1 fruit = {"bananas": 5, "apples": 10, "strawberries": 15}
2 bananas = fruit[5]
3 print(bananas)
```

29. Where is the mistake and why does it break the code?

```
1 fruit = ["orange", "grapefruit", "papaya", "raspberry"]
2 fruit.add("peach")
3 for item in fruit:
4     print(fruit)
```

30. Where is the mistake and why does it break the code?

```
1 fruit = "pineapple"
2 if "apple" in fruit
3     print("yummy")
```

31. Where is the mistake and why does it break the code?

```
1 fruit = "grapefruit"
2 fruit.replace("grape", "passion")
3 print(fruit[11])
```

32. Where is the mistake and why does it break the code?

```
1 fruit = {"cranberries": 10, {"coconuts", "pomegranates"}: 1}
2 for item, number in fruit.items():
3     print(item, number)
```

33. Where is the mistake and why does it break the code?

```
1 fruit = ["banana", "banana", "apple", "banana", "lemon"]
2 count_fruit = {}
3 for item in fruit:
4     count_fruit[item] += 1
5 print(count_fruit)
```

34. Where is the mistake and why does it break the code?

```
fruit = {"cranberries": 10, "lychees": 8, "pears": 5}
for item in fruit:
    amount = int(item)
    print(item, amount)
```

35. Where is the mistake and why does it break the code?

```
def count_all_fruit(list_fruit):
    total = len(list_fruit)
    return total

fruit = ["apricot", "mango", "passionfruit"]
count_all_fruit(fruit)
print(total)
```

4 Tracing

Context

Whenever computers try to make sense of natural language, it is helpful to have some knowledge about the properties of a word. For instance, we could say that a property of *cat* is *furry*. Each year, the natural language processing community has a number of contests in which computers have to solve different tasks about word meaning. One of the 2018 tasks was to find out whether a property can distinguish two words. For instance, the property *red* can distinguish *apple* from *banana*, because apples can be red, but bananas are usually not red. One of the systems we submitted uses definitions of all senses of a word to find properties. A simplified version is included in Appendix A, B and C. The goal of the task was to label two given words and a property as discriminative or not.

The idea behind our approach is that important properties of words should be mentioned in dictionary definitions. If a property is mentioned in only one of the definitions, we say it is discriminative (labeled as 1). If it is found in both definitions, it applies to both words and cannot be used to discriminate between words (labeled as 0). In cases where the property does not appear in any of the definitions, we do not want to make a decision yet (labeled as `None`).

We used the computational lexicon WordNet to get definitions of word senses. In this resource, senses are called synsets.

Appendices: Code & WordNet definitions

Please have a look at the following appendices:

- Appendix A: `utils.py` contains a number of functions to decide whether two words and a property should be classified as discriminative (1) or not discriminative (0) or undecided (None). It contains 2 helper functions:
  - `get_definitions()`: given a word as input, this function collects all definitions of the senses of the word in WordNet.
  - `check_def()`: given a list of definitions and a property, this function checks whether the property is in any of the definitions.
- Appendix B: `system.py`:
  - `definition_checker()`: given two words and a property, this function checks whether the property can distinguish between the words. It uses the `get_definitions()` and `check_def()` from `utils.py` to do this.
  - In addition, this script contains three test instances, each of which consists of a word pair and a property. Each test instance is processed by the `definition_checker()`. The answers are collected for further processing (e.g. we could write them to a file - this is not shown here).
- Appendix C: This appendix contains all WordNet definitions of the words in the test pair *bus* and *taxi* in the same sequence as they appear in WordNet. This is additional information you need to trace the code. In the Python scripts, this information will be accessed via the WordNet module included in nltk.

Questions about the code

Please read the code in Appendix A and B and answer the questions below. For some, we ask you to consider the test instance. By test instance, we mean the first test instance shown in `system.py` consisting of *taxi*, *bus*, *passengers*.

36. What is the **type** of the variable `def_tokenized` on line 13 in `utils.py`? What is the **type** of `definitions` returned on line 15 in `utils.py`?

37. Why do you think we tokenize the definition instead of simply checking whether the property is in the definition represented as a string? Please provide an example. You can use the example definitions for inspiration, use an example from the lectures or come up with your own example.

38. Please consider the first test instance (i.e. the first iteration of the loop on line 31 in `system.py`). What will be the **lengths** of `definitions1` and `definitions2` (lines 8 and 9) in `system.py`?

39. What will be the **values** of `def1_answer` and `def2_answer` on lines 11 and 12 in `system.py` for the first test instance?

40. Will the loop starting on line 23 of `utils.py` be broken when the function is called on lines 11 and 12 in `system.py`? If so, at which iteration? Hint: Have a close look at the definitions in Appendix C (they appear in the same sequence as in WordNet).

41. What will the function `definition_checker` return when considering the first test instance? Please provide a short explanation. Hint: Have a close look at the definitions in `definitions.txt`.

42. Would the code still run if line 1 of `system.py` were changed to `import utils`? If so, why? If not, please explain what other change we could make to fix the code.

43. What is the **type** of the variable `test_instance` on line 32 of `system.py`? What is the **type** of the variable `test_instances` defined on line 24 of `system.py`)?

44. What will be the **length** of the variable `answers` (first defined on line 30 of `system.py`) after iterating over all test instances? Does this depend on the outcome of the definition check performed in the function `definition_checker()` in `system.py`?

45. What are the **possible types** of the variable `answer` on line 35 of `system.py`?

5 Writing code

Below, you will find some code descriptions. Please write code according to these descriptions. This exercise is worth 15 points in total (1 point per question).

General information

The JSON file `potter.json` contains information about the characters of the Harry Potter books. The JSON file is formatted as a 'list of dictionaries'. Part of the file is shown below. For each of the characters, we have information about their name, their role, their bloodstatus, their species, etc. We also have information about the wood, the length and the core of their wands. In this part of the exam, we will write code to read this JSON, find all wizards that have a wand with a specific core and write the result to a text file.

```
[
    {'_id': '5a12292a0f5ae10021650d7e',
      'name': 'Harry Potter',
      'role': 'student',
      'house': 'Gryffindor',
      'school': 'Hogwarts School of Witchcraft and Wizardry',
      'wand': 'Holly, 11", phoenix feather',
      'ministryOfMagic': False,
      'orderOfThePhoenix': True,
      'dumbledoresArmy': True,
      'deathEater': False,
      'bloodStatus': 'half-blood',
      'species': 'human'},
    {'_id': '5a109f8b3dc2080021cd8795',
      'name': 'Draco Malfoy',
      'role': 'student',
      'house': 'Slytherin',
      'school': 'Hogwarts School of Witchcraft and Wizardry',
      'wand': 'Hawthorn, 10", unicorn hair',
      'ministryOfMagic': False,
      'orderOfThePhoenix': False,
      'dumbledoresArmy': False,
      'deathEater': True,
      'bloodStatus': 'pure-blood',
      'species': 'human'},
    {'_id': '5a109af13dc2080021cd877a',
      'name': 'Hermione Granger',
      'role': 'student',
      'house': 'Gryffindor',
      'school': 'Hogwarts School of Witchcraft and Wizardry',
      'wand': 'Vine wood, 10 3/4", dragon heartstring',
      'ministryOfMagic': False,
      'orderOfThePhoenix': True,
      'dumbledoresArmy': True,
      'deathEater': False,
      'bloodStatus': 'muggle-born',
      'species': 'human'}
]
```

Part I: Load the JSON file (3 points)

46. Load the module/package that allows us to work with JSON files.

47. Open the file `"potter.json"` for reading (assume that the file is in the current working directory). You can either use a context manager, or make sure to close the file again at the end of this part.

48. Load the JSON file as a Python object (remember: you have to choose between `load()` and `loads()`) and store it as `list_characters`.

Part II: Create a function to find all wizards with a certain wand (8 points)

49. Define a function that follows these specifications:

    (a) The function is called `find_wizards_by_wand`.

    (b) It should have one positional parameter: `list_characters`.

    (c) It should have one keyword parameter: `core` with default value `"dragon heartstring"` (string).

50. At the start of the function, create an empty set called `wizards`.

51. Loop over each `char` in the list of characters.

52. Get the `wand` of each character (you can safely assume that each character has information about their wand). For example, the character with the name `Harry Potter` has the wand `'Holly, 11", phoenix feather'`.

53. Split the `wand` into `wand_wood`, `wand_length` and `wand_core`. To do this, first make use of the string method `split()` and store the result as `split_wand`. Then assign the variable name `wand_wood` to the first element, `wand_length` to the second element and `wand_core` to the third element of `split_wand`. Make sure that none of the elements have leading/trailing spaces.

54. Check if `wand_core` is equal to the input parameter `core`.

55. If so, get the `name` of the character and add it to the set `wizards` (you can safely assume that each character has information about their name).

56. At the end of your function, return the set of `wizards`.

Part III: Calling the function and writing the result to a file (4 points)

57. Call the function `find_wizards_by_wand()` and set the keyword argument `core` to `"unicorn hair"` (string). Assign the result of the function to a variable called `unicorn_hair_wizards`.

58. Open a file called `"unicorn_hair_wizards.txt"` for writing (assume the file is in the current working directory).

59. Create one string of `unicorn_hair_wizards` by using the string method `join()` to join the elements in the set `unicorn_hair_wizards` by newline characters. Store the result as a variable called `lines`.

60. Write the string `lines` to the file `"unicorn_hair_wizards.txt"`.

Page intentionally left blank. Appendix A on the next page.

Appendix A `utils.py`

```
from nltk.corpus import wordnet as wn
from nltk import word_tokenize

def get_definitions(word):
    """
    Get the definitions of all senses (synsts) of a
    word and return a list of all definitions.
    """
    definitions = []
    synsets = wn.synsets(word)
    for syn in synsets:
        definition = syn.definition()
        def_tokenized = word_tokenize(definition)
        definitions.append(def_tokenized)
    return definitions

def check_def(definitions, property):
    """
    Check if the property prop is in any of
    the definitions
    """
    def_answer = 0
    for definition in definitions:
        if property in definition:
            def_answer = 1
            break
    return def_answer
```

Page intentionally left blank. Appendix B on the next page.

Appendix B `system.py`

```
from utils import check_def
from utils import get_definitions

def definition_checker(word1, word2, property):
    """
    Checks whether the property can distinguish between word1 and word2
    """
    definitions1 = get_definitions(word1)
    definitions2 = get_definitions(word2)

    def1_answer = check_def(definitions1, property)
    def2_answer = check_def(definitions2, property)

    if (def1_answer == 1) and (def2_answer == 0):
        answer = 1
    elif (def1_answer == 1) and (def2_answer == 1):
        answer = 0
    elif (def1_answer == 0) and (def2_answer == 1):
        answer = 1
    else:
        answer = None
    return answer

test_instances = [
{'word1' : 'bus', 'word2': 'taxi', 'property': 'passengers'},
{'word1' : 'shrimp', 'word2': 'spinach', 'property': 'pink'},
{'word1' : 'apple', 'word2': 'banana', 'property': 'mammal'}
]

answers = []
for test_instance in test_instances:
    word1 = test_instance['word1']
    word2 = test_instance['word2']
    property = test_instance['property']
    answer = definition_checker(word1, word2, property)
    answers.append([word1, word2, property, str(answer)])
```

Page intentionally left blank. Appendix C on the next page.

Appendix C WordNet definitions

```
WordNet definitions of "bus":

1 a vehicle carrying many passengers; used for public transport
2 the topology of a network whose components are connected by a busbar
3 an electrical conductor that makes a common connection between several circuits
4 a car that is old and unreliable
5 send or move around by bus
6 ride in a bus
7 remove used dishes from the table in restaurants

WordNet definitions of "taxi":

1 a car driven by a person whose job is to take passengers where they want
to go in exchange for money
2 travel slowly
3 ride in a taxicab
```