UNIVERSITY OF GOTHENBURG

DIT009 - Fundamentals of Programming

H24 Exam 3 - Lindholmen, 2025-08-20

Teachers:	Francisco Gomes and Ranim Khojah
Questions:	+46 31 772 6951 (Francisco) Francisco will visit the exam hall at ca 15:00 and ca 16:30
Results:	Will be posted within 15 working days.
Grades:	00–49 points: U (Fail) 50–69 points: 3 (Pass) 70–84 points: 4 (Pass with merit) 85–100 points: 5 (Pass with distinction)
Allowed aids:	No aids (books or calculators) are allowed.

Read the instructions below. Not following these instructions will result in the point deductions.

- Write clearly and in legible English (illegible translates to "no points"!). Motivate your answers, and clearly state any assumptions made. Your own contribution is required.
- Before handing in your exam, **number and sort the sheets in task order!** Write your anonymous code and page number on every page! The exam is anonymous, **do not** leave any information that would reveal your name on your exam sheets.
- Answers that require code must be written using the Python programming language. When answering what is being printed by the program, make sure to write as if the corresponding answer were being printed in the console.
- You can assume that all python files in the code presented in this exam are in the **same folder**. Similarly, **you can assume** that all libraries used in the code (math, regex, files, json, etc.) are properly imported in their respective file.

Cheatography

Regular Expressions Cheat Sheet

by Dave Child (DaveChild) via cheatography.com/1/cs/5/

Anchors

^	Start of string, or start of line in multi-line pattern
\A	Start of string
$	End of string, or end of line in multi-line pattern
\Z	End of string
\b	Word boundary
\B	Not word boundary
\<	Start of word
\>	End of word

Character Classes

\c	Control character
\s	White space
\S	Not white space
\d	Digit
\D	Not digit
\w	Word
\W	Not word
\x	Hexadecimal digit
\O	Octal digit

POSIX

[:upper:]	Upper case letters
[:lower:]	Lower case letters
[:alpha:]	All letters
[:alnum:]	Digits and letters
[:digit:]	Digits
[:xdigit:]	Hexadecimal digits
[:punct:]	Punctuation
[:blank:]	Space and tab
[:space:]	Blank characters
[:cntrl:]	Control characters
[:graph:]	Printed characters
[:print:]	Printed characters and spaces
[:word:]	Digits, letters and underscore

Assertions

?=	Lookahead assertion
?!	Negative lookahead
?<=	Lookbehind assertion
?!= or ?<!	Negative lookbehind
?>	Once-only Subexpression
?()	Condition [if then]
?()|	Condition [if then else]
?#	Comment

Quantifiers

*	0 or more	{3}	Exactly 3
+	1 or more	{3,}	3 or more
?	0 or 1	{3,5}	3, 4 or 5

Add a ? to a quantifier to make it ungreedy.

Escape Sequences

\	Escape following character
\Q	Begin literal sequence
\E	End literal sequence

"Escaping" is a way of treating characters which have a special meaning in regular expressions literally, rather than as special characters.

Common Metacharacters

^	[	.	$
{	*	(	\
+	)	|	?
<	>		

The escape character is usually \

Special Characters

\n	New line
\r	Carriage return
\t	Tab
\v	Vertical tab
\f	Form feed
\xxx	Octal character xxx
\xhh	Hex character hh

Groups and Ranges

.	Any character except new line (\n)
(a|b)	a or b
(...)	Group
(?:...)	Passive (non-capturing) group
[abc]	Range (a or b or c)
[^abc]	Not (a or b or c)
[a-q]	Lower case letter from a to q
[A-Q]	Upper case letter from A to Q
[0-7]	Digit from 0 to 7
\x	Group/subpattern number "x"

Ranges are inclusive.

Pattern Modifiers

g	Global match
i *	Case-insensitive
m *	Multiple lines
s *	Treat string as single line
x *	Allow comments and whitespace in pattern
e *	Evaluate replacement
U *	Ungreedy pattern

* PCRE modifier

String Replacement

$n	nth non-passive group
$2	"xyz" in /^(abc(xyz))$/
$1	"xyz" in /^(?:abc)(xyz)$/
$`	Before matched string
$'	After matched string
$+	Last matched string
$&	Entire matched string

Some regex implementations use \ instead of $.

By **Dave Child** (DaveChild)
cheatography.com/davechild/
aloneonahill.com

Published 19th October, 2011.
Last updated 12th March, 2020.

Table 1: List of useful methods in Python, for Lists, Strings and Dictionaries.

Used in	Method specification	Description
—	int(value)	Converts the specified value into integer.
—	str(value)	Converts the specified value into a string.
—	float(value)	Converts the specified value into a float.
List	len(list)	Returns the number of elements in a list.
List	append(element)	Appends the specified element to the end of the list.
List	pop()	Removes and returns the element at the end of the list.
List	remove(element)	Remove the specified element from the lsit.
Dict	keys()	Returns a list with all the keys in a dictionary.
Dict	values()	Returns a list with all the values in a dictionary.
Dict	items()	Returns a list containing a tuple for each key value pair.
String	split(delimiter)	Split a string into a list of strings separated by the delimiter.
String	strip()	Remove empty spaces at the beginning and end of a string.
String	replace(pattern, replacement)	Returns a string where the pattern is replaced by the specified value.
String	lower()/upper()	Returns a new string with only lower or upper case characters.

Question 1 [Total: 20 pts]: This question verifies your knowledge of algorithms, variables, execution flow, and debugging programs.

Q1.1 [5 pts]: Running the code below will trigger an error. You must (i) identify the line of code of the error, (ii) explain the cause of the error, (iii) explain how to fix the error.

Q1.2 [15 pts]: Assuming that the error has been fixed, write **exactly** what is printed when running the Python program below.

Important: The **ordering and the formatting of your printing is important**. Although it is not needed, you can use "_" to indicate a white space in the string. For instance: "Hello World" becomes "Hello_World".

```python
def function_a(heroes, hero_a, hero_b):
    for i in range(len(heroes)):
        hero_name = heroes[i]
        if hero_name == hero_a:
            heroes[i] = ""

    heroes[-1] = hero_b
    print("A has ", len(heroes), " heroes: ", heroes)

def function_b(x, y):
    z = 10

    if y % 2 != 0:
        z = x
        x = y
        y = z
    if x > y:
        x += y
    else:
        y += x

    function_c(z)
    print("B: x =", x)
    print("B: y =", y)
    print("B: z =", z)

counter = 4
def function_c(counter):
    counter = counter + 5
    result = ""
    for i in range(counter, 1, -1):
        result = result + i + " "

    print("C: result =", result)
    print("C: counter =", counter)

def main():
    names = ["Storm", "Batman", "Wolverine", "Batman", "Spider-Man"]
    function_a(names, "Batman", "Rogue")

    x = int(len(names) / 2)
    y = len(names[0])
    function_b(x, y)

    print("Main: names =", names)
    print("Main: counter =", counter)
    function_a(names, "Cyclops", "Hawkeye")

    x = 3
    y = 3
    function_b(x, y)
    print("Main: ", x+y+counter)

if __name__ == "__main__":
    main()
```

Question 2 [20 pts]: Using your knowledge of regular expressions, answer the questions below:

```
# A list of 10 social media posts
posts = [
  "Movie night at 21:30 with @cine_lover! #popdiva #Friday - by user: filmFan2024",
  "'Neon Skies' was epic. Trailer at https://example.com/trailer #movies",
  "Coffee in Gothenburg at 07:05 before #Keynote with @Prof_Ana - see you there!",
  "2023 recap: learned Python,built a bot, ran 10k. #goals",
  "Who remembers the 9:45 show? Nope, only 8:59 was free. #throwback",
  "Streaming live now at http://stream.me/live #gaming @player_two",
  "Project demo 4:00 today in Room Pink. @team_lead #SEM",
  "Can't wait for PopCon 2025!! Tickets: https://popcon.org #events",
  "Just saw Alex at the CEE hackathon - legendary! #devlife",
  "Ops meeting cancelled? Send new date soon. #updates"
]

# REGEX used in Q2.1
# a) r"\b\d{2}:\d{2}\b",
# b) r"\b[A-Z][ae]\w+\b",
# c) r",\s+\w+\b",
```

Q2.1 [10 pts]: For each regex `ae` shown in the code, write the result of `re.findall(expression, post)` for each `post` in the list.

Important: 1) Indicate which regex you are answering by using `a`, `b`, or `c`. 2) If a regex returns more than three matches for a post, you may list only **three** matches in any order (listing extra items that are incorrect may lose points).

Q2.2 [10 pts]: Write two regular expressions:

1. Extract all **sentences that end with a question mark** (`?`). You may assume sentences are separated by `.`, `!`, or `?`.
2. Extract all **URLs** that begin with `https://` and continue until the next whitespace character.

For each regex you write, briefly **explain** how it works (what each key part matches and why).

Question 3 [Total: 40 pts]: This question verifies your knowledge of code quality, refactoring, data structures, and algorithms.

Q3.1 [10 pts]: Write a function that asks the user for their favourite animal and stores each answer in a list. The program should follow these rules:

- Keep asking for animal names **until** the user types the word `exit`. Only unique animal names are allowed.
- You do not need to handle different capitalisations. So, you can assume that the user will only type words using lowercase.

- If the user enters an **empty string** (`""`), the program should print the message: `Animal name cannot be empty.`. The program should then continue asking for the next animal name (the program should not stop).
- At the end, the function should return a list containing all the animal names entered (excluding `exit`).

Example interaction:

```
Enter your favourite animal: cat
Enter your favourite animal: dog
Enter your favourite animal:
Animal name cannot be empty
Enter your favourite animal: cat
Enter your favourite animal: horse
Enter your favourite animal: exit
```

Returned list:

```
["cat", "dog", "horse"]
```

Q3.2 [10 pts]: Write a function that receives two lists: one containing **animal names** and the other containing **adjectives**. The function should return a list with all possible combinations of an adjective followed by an animal name (e.g., `"smart turtle"`, `"needy dog"`).

- If either of the provided lists is empty, the function should raise a `ValueError` with the message: `Lists cannot be empty`.
- You may assume that all words are in lowercase.
- Pay attention to the **order** in which the combinations are generated: all combinations for the first adjective should appear before those for the second adjective, and so on.

Example:

- Input:
  ```
  animals = ["turtle", "dog"],
  adjectives = ["smart", "needy", "playful"]
  ```
- Output:
  ```
  ["smart turtle", "smart dog", "needy turtle",
  "needy dog", "playful turtle", "playful dog"]
  ```

Answer the questions below based on the following code. Although the code produces the correct output, it contains several code smells.

```python
def doStuff(x):
    d = {}
    for i in x:
        l = len(i)
        if l < 5:
            if "short" not in d:
                d["short"] = []
            d["short"].append(i)
        elif l >= 5 and l <= 7:
            if "medium" not in d:
                d["medium"] = []
            d["medium"].append(i)
        elif l > 7:
            if "long" not in d:
                d["long"] = []
            d["long"].append(i)
    return d

def main():
    animals1 = ["cat", "tiger", "elephant", "dog", "alligator", "cow"]
    result1 = doStuff(animals1)
    print(result1)

    animals2 = ["bat", "whale", "hippopotamus", "ox", "emu"]
    result2 = doStuff(animals2)
    print(result2)

    animals3 = []
    result3 = doStuff(animals3)
    print(result3)

if __name__ == "__main__":
    main()
```

Q3.3 [10 pts]: You should (i) explain in natural language what is the purpose of the function `doStuff`, and (ii) write what will be printed when running the code.

Q3.4 [10 pts]: The code contains several code smells. Identify and explain each code smell. For each code smell, you must (i) specify the line number(s) where the problem occurs, (ii) suggest and describe specific refactoring steps to improve the code.

Question 4 [Total: 20 pts]: Using your knowledge of recursion, answer the questions below:

Q4.1 [10 pts]: Write a recursive function that takes a list of integers and returns the **average of all even numbers greater than zero** in the list. If there are no positive even numbers, the function should return `0`. Your solution will be evaluated in terms of its correctness, readability, and efficiency.

Important:

- You **must** use recursion in your solution (loops are not allowed).
- You can assume the list contains only integers (positive, negative, or zero).
- The elements in the list can be in any order (i.e., the list is not sorted).
- When writing your algorithm, we recommend adding lines of code to help with the call stack question.
- **Include a call** to your function for all cases below, to illustrate how the parameters are initialised.

```
Examples:
Case 1:..................................... | Case 4:.............................
Input: [4, -3, 7, 10, 2], Output: 5.33      | Input: [-2, -4, 3, 5, 0], Output: 0
# Ex: (4 + 10 + 2) / 3

Case 2:..................................... | Case 5:.............................
Input: [8, 12, 15, 20], Output: 13.33       | Input: [], Output: 0

Case 3:..................................... | Case 6:.............................
Input: [10, 2, 0, -15, -2], Output: 6.0     | Input: [10], Output: 10
```

Q4.2 [10 pts]: Illustrate the step-by-step execution of your function by showing the call stack when executing your function with the **first case** in the examples above: `Input: [4, -3, 7, 10, 2]`.

Your stack trace should include, at each step:

1. Function name and parameter values.
2. Variables and their values (sum so far, count so far, etc.).
3. The line of code being executed when calling (or returning from) the recursive step.

As a reminder of how to illustrate the call stack, Figure 1 includes an example from the recursion lecture.

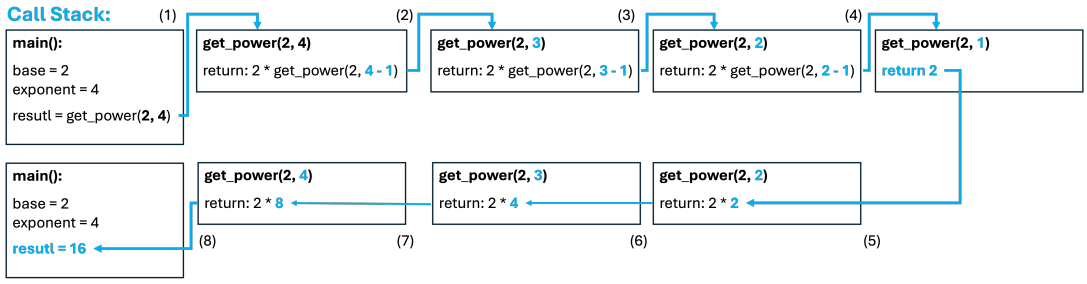

Figure 1: Example of a call stack for a recursive function named `get_power()`.