UNIVERSITY OF GOTHENBURG

DIT009 - Fundamentals of Programming

H25 Exam 1 - Lindholmen, 2025-10-29

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 answers 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 files.
- All code will be evaluated based on its **correctness, readability, and efficiency**. In addition, your solution should demonstrate the use of **appropriate programming constructs** that align with the problem requirements and the learning goals of each question.

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.
Dict	get(keyname, value)	Returns the value associated with keyname. If the dictionary does not contain keyname, the method returns the specified value instead.
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]: Using your understanding of variables, execution flow, and program debugging, write exactly **what the code below prints** when executed. **Note that** the ordering and formatting of the output are important. Although it is not needed, you can use "_" to indicate a white space in the string. For instance: "Hello World" becomes "Hello_World".

```
def function_a(scores, bonus_threshold):
  for i in range(len(scores)):
    if scores[i] < bonus_threshold:
      scores[i] += 3

  scores[0] += x
  print("Scores: ", scores)

x = 2
def function_b(x, y):
  z = x

  if y == z:
    z = x + y
  elif x % 5 == 0:
    z = y

  a = "5"
  if x > y:
    a = a * 2
  else:
    a = a + "3"

```

```python

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

  return z

def function_c(things, name):
  for product in things:
    if name == product:
      reviews = things.get(name)
      function_a(reviews, 4)
      things[name] = reviews

      return f"Product {name} updated."

  return f"Product {name} not found."

def main():
  products = {
    "Laptop": [7, 3, 4, 8],
    "Headphones": [4, 4, 5],
    "Mouse": [5, 3, 2],
    "Keyboard": [2, 5, 6, 5, 3, 2, 8, 8]
  }

  msg = function_c(products, "Laptop")
  print("Main: ", msg)

  x = products["Mouse"][0]
  y = products["Mouse"][2]
  y = function_b(x, y)

  print("Main: x = ", x)
  print("Main: y = ", y)

  reviews = products["Keyboard"][2:6]
  function_a(reviews, 5)
  print("Main: ", products["Keyboard"])

  msg = function_c(products, "Phone")
  print(msg)

  function_b(4, 7)
  print("Main: x = ", x, " y = ", y)

  for product in products:
    print(f"Main Products: {product}: {products[product]}")

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

Question 2 [30 pts]: Using your knowledge of algorithms, lists, and regular expressions (regex), answer the question below:

You are hired to parse forum discussions about household expenses in Sweden. You can assume that the posts may mention many kinds of numbers (years, temperatures, heights, USD prices, etc.), but users report their expense amounts using the format " X.YY SEK", where:

- ' X' is the integer part of the number with at least one empty space before it.
- '.' is the decimal separator.
- 'YY' is exactly two digits (the cents).
- the format ends with a space and the currency ' SEK'.

Some posts may also use EUR for other amounts, or include other numeric values such as USD prices, $ amounts, years (e.g., 2023), temperatures (e.g., 25.2), or heights (e.g., 1.98 cm). Your program **should ignore** those numbers. Here is an example of a list of posts from a forum webpage and the code used to extract the expenses.

```
media_posts = [
    "Post by Alice85: Just got our electricity bill for the winter - 945.75 SEK - which is actually cheaper than last year s 1050.00 SEK! Also, we bought a used washing machine for 300.00 EUR and sold our old one for 150.00 SEK. I am worried about the costs for heating, because we have a high ceiling of 4.25 cm and the temperature inside the flat is 21.80 C .",

    "Post by Jonas_Swe: Groceries for the week cost about 1200.50 SEK. The bread was 25.20 SEK kg but I only count the total. We spent around $45.99 on imported snacks and 2023.00 SEK (not bad compared to 2024!). Still, I feel bad paying 2.68 SEK per egg in 20 eggs box.",

    "Post by LinaK: Heating costs are insane this month: 2845.00 SEK. Our cat s vet visit was 1150.00 SEK, and coffee beans online were 180.00 USD.",

    "Post by Karen: I just think we should all work harder. I am a hardworker and people complain about not having 1.5 hours to study. Lazy people end up with -1000 SEK in their account."
]

all_matches = []
expense_regex = "" # <-- your regex would go here

for post in media_posts:
    # Note: Each 'expenses' below will be a list of strings.
    expenses = re.findall(expense_regex, post)
    all_matches.append(expenses) # List of lists of strings

print(all_matches)

# Regex for Q2.2:
# a) r"\w{2,}\d{2}\b"
# b) r"\w{10}"
# c) r"^.+?:"
```

Post	Average	Should match	Should not match
Post 1:	715.25	[”945.75 SEK”, ”1050.00 SEK”, ”150.00 SEK”]	[”300.00 EUR”, ”4.25”, ”21.80 ”]
Post 2:	812.845	[”1200.50 SEK”, ”25.20 SEK”, ”2023.00 SEK”, ”2.68 SEK”]	[”$45.99”, ”2024”, ”20”]
Post 3:	1997.5	[”2845.00 SEK”, ”1150.00 SEK”]	[”180.00 USD”]
Post 4:	0.0	[ ]	[”1.5 hours”, ”-1000 SEK”]

Table 2: Average expense per post, matched SEK values, and non-matching numeric examples.

Q2.1 [10 pts]: (i) Write the regex that should be used to extract the individual expenses from the posts. Using the media posts above, Table 2 illustrates all matches and some examples of numbers that should not match by your regex. **Explain** (ii) the different elements in your regex, and (iii) how to make the regex more robust by matching both '.' and ',' as decimal separators.

Q2.2 [10 pts]: For each regex shown below, write the list of matches for each `post` in the list (i.e., each regex yields 4 lists).

Important: 1) Indicate which regex you are answering by using (a-c), and the post: Post 1, Post 2, Post 3, and Post 4. 2) If a regex returns more than three matches for a post, you may list only **three** matches in any order (listing additional incorrect items results in point deductions). 3) If there are no matches for the regex in a post, write an empty list: []

Q2.3 [10 pts]: Write a function that receives a list where each element is a list of string matches from the desired regex (e.g., `all_matches`). The function should return a list of averages, i.e., the average expense of each post. The average of the post is the sum of its extracted expenses divided by the number of expenses for that post.

Table 2 illustrates the expenses per post and all cases that should be handled. For the example above, the function should return the list: `[715.25, 812.845, 1997.5, 0.0]`.

Question 3 [35 pts]: Using your knowledge of data types, debugging, and code quality, answer the questions below. You are hired to maintain the code of a catalog of transaction history. The code below stores users and their past financial transactions. Each transaction has a date and a value.

```
# Example input for extractor:
#  - x = [[("2025-02-14",249.90), ("2025-03-01",310.50), ("2025-03-01",120.00)]]
#  - y = "2025-03-01"
#  - Should return: [310.50, 120.00]
def extractor(x, y):
    z = []
    for i in range(len(x)):
        a = x[i][0]
        if a == y:
            z[i] = x[i][1]

    return z

def main():
```

```
    transaction_history = {
      "C7D": [("2023-11-20",499.99), ("2025-02-14",199.00)],
      "E5F": [("2025-03-01",89.90)],
      "A3B": [("2025-02-14",249.90), ( "2025-03-01",310.50), ("2025-03-01",120.00)],
      "G2H": [],
      "J9K": [("2023-01-20",75.00), ("2024-03-22",980.00), ("2025-04-10",430.00)]
    }

    for id in transaction_history:
        extracted_values = extractor(transaction_history[id], "2025-03-01")
        print(extracted_values)

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

Q3.1 [8 pts]: Running the code above will trigger a bug. You must i) identify the line of code that will trigger the bug, ii) describe why that bug happens, and iii) describe how to fix that bug. **Note:** You can answer the upcoming questions assuming that the bug has been fixed.

Q3.2 [7 pts]: The function `extractor` is difficult to understand. You must: i) describe two code smells that reduce the quality of the function, ii) write a refactored version of the function, and iii) explain **why** the modifications improve the code quality. Tips: 1) write lines of code in your refactored version, and refer to those lines in your justifications. 2) You cannot change the behavior of the function.

Q3.3 [10 pts]: Which data structure(s) is/are used to represent the transaction history in this program? Do you think this is an appropriate choice? Answer Yes or No, and justify your answer by explaining the advantages of the discussed structure(s) for organizing the data.

Q3.4 [10 pts]: Write a function that receives a dictionary with the transaction history and returns a new dictionary where:

- Each key is a year (”YYYY”) extracted from the transaction date.
- Each value is a list containing all transaction values (as floats) that occurred in that year.
- If the input dictionary is empty, your function should return an empty dictionary.

For the `transaction_history` variable in the code above, the function should return:

```
    {
        "2023": [499.99, 75.00],
        "2024": [980.00],
        "2025": [199.00, 89.90, 249.90, 310.50, 120.00, 430.00]
    }
```

Question 4 [15 pts]: Using your knowledge of recursion, answer the questions below:

Q4.1 [10 pts]: Write a recursive function that returns the total sum of a list of integers based on the following rules:

- For numbers greater than or equal to zero, add the number normally.
- For negative numbers, add twice their absolute value. You can use the function `abs(number)` to get the absolute of a number.
- You must use recursion in your solution (loops are not allowed). Except for `len()` and `abs()`, you cannot use helper functions (i.e., max, sum, sort, min, etc.).
- You must write how you would call your recursive function for all the input cases listed in Table 3.

Table 3 shows some examples of input and output. Your code will be evaluated based on correctness, readability, and efficiency.

Input	Breakdown (Equation)	Output
`[2, -3, 4, -1]`	$2 + (2 \times \lvert -3 \rvert) + 4 + (2 \times \lvert -1 \rvert)$	`14`
`[-5, -2, 0, 3]`	$(2 \times \lvert -5 \rvert) + (2 \times \lvert -2 \rvert) + 0 + 3$	`17`
`[1, 5, 3]`	$1 + 5 + 3$	`9`
`[-4]`	$2 \times \lvert -4 \rvert$	`8`
`[]`	No elements to sum.	`0`

Table 3: Examples of recursive summation with double absolute values for negative numbers.

Q4.2 [5 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: [2, -3, 4, -1]`. Your stack trace should include, at each step:

- Function name and parameter values.
- Variables and their values.
- 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.

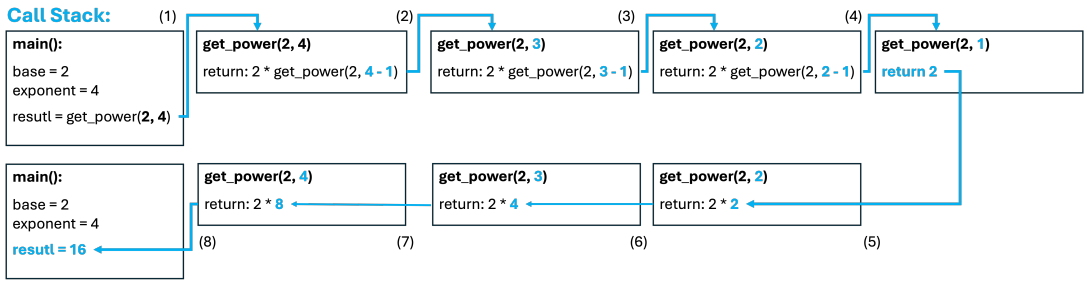

Figure 1: Example of a call stack for a recursive function named `get_power()`.