DIT009 - Fundamentals of Programming

H24 Exam 2 - Lindholmen, 2025-01-09

Teachers:	Francisco Gomes and Ranim Khojah
Questions:	+46 31 772 6951 (Francisco)
	Ranim 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 $.

Table 1: List of useful methods in Python, for Lists, Stringsm 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: 15 pts]: This question verifies your knowledge of algorithms, variables, execution flow, and debugging programs. Write **exactly** what is printed when running the Python program below. **Important:** The **ordering and the formatting is important**.

```python
counter = 2
def function_a(fruits, new_fruit):
    del fruits[1]
    result = "Result: "
    for i in range(1,len(fruits)-1):
        result += fruits[i] + " "

    fruits[-1] = new_fruit
    print("A: ",result)

def function_b(x, y):
    if x % 2 == 0:
        temp = x
        x = y
        y = temp
    if x < y:
        x += y
    else:
        y += x

    print("B: x =", x)
    print("B: y =", y)
    return x
```

```python

def function_c(numbers):
    counter = len(numbers) / 2
    total = counter
    for i in range(len(numbers)):
        total = total + numbers[i]
        for j in range(i+1, len(numbers)):
            total = total + numbers[j]

    print("C: total =", total)

x = 3
y = 4
z = function_b(y, x)
print("Main: x =", x, "y = ", y)
print("Main: z =", z)

fruits = ["Apple", "Banana", "Pear", "Kiwi", "Orange"]
function_a(fruits, "Lemon")
print("Main: fruits =", fruits)

numbers = [5, 6, 7, 8]
function_c(numbers)
print("Main: counter =", counter)

z = function_b(len(fruits), z)
print("Main: z =", z)

function_a(fruits, "Strawberry")
print("Main: fruits =", fruits)

counter += 10
numbers = numbers[0:2]
function_c(numbers)
print("Main: counter =", counter)
```

Question 2 [25 pts]: Using your knowledge of regular expressions, answer the questions below:

```python
# A list of 10 social media posts

posts = [
 "Just finished a killer w@rkout! #gymday - by user: lift_master",
 "Meal prepping for the week ahead. #healthychoices - by user: fitfoodie",
 "Ran my first 5k @gym today! So proud! #runnerslife - by user: runner_jane2023",
 "Achieved a new personal best on squats! #strength - by user: strong_mike",
 "What's your go-to post-workout snack? #fitnesschat - by user: fit_guru2024",
 "New yoga routine from an AI agent. Relaxed out @nature. #zen - by user: yoga_88",
 "Hydration is key! Drink water, folks. #stayhealthy - by user: water_warrior",
 "Love the progress in my fitness journey. #keepgoing - by user: health_fanatic",
 "Check this new tracker by user: lift_master.Game changer! - by user: gadget_geek",
 "Who else loves a good outdoor workout? #naturefit - by user: trail_runner"
]

```

```python
# REGEX used in the question
# a) r"\b\w{3}\b",
# b) r"@\w+",
# c) r"\b[aeiou]{2}\w*\b",
# d) r"@\w+\b",
# e) r"\s\w+\d{3}\b"
```

Q2.1 [15 pts]: For each regex, write the result of doing `re.findall(expression, post)`.

Important: 1) Indicate which regex you are answering to by using a, b, c, d, or e. For regex that return more than 3 matches, you may write only three matches in any order you prefer (note that writing more than three and including incorrect ones can risk deducting points).

Q2.2 [10 pts]: Write two regex patterns to:

1. Extract the first sentence of each post (a sentence ends with a period, exclamation point, or question mark).
2. Extract the part that states the post's user (that part is always indicated by the string "- by user: ..." at the end of each post).

In your answer, you must: 1) Write the two regex patterns; 2) Explain how each regex works, breaking it down into its own components.

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

3.1 [10 pts]: Using your knowledge of variables and types: (i) define and explain how Python handles types, and (ii) explain one advantage and one disadvantage of that type system.

3.2 [5 pts]: You were tasked to work with the code below for a Pokemon app. You should: (i) explain in natural language what is the purpose of the function `process_data`, and (ii) write what will be printed when running the code.

```python
def process_data(data, filter1, filter2, format_type, format_length, data_type):

    filtered_data1 = []
    for pokemon in data.keys():
        types = data[pokemon]
        if filter1 in types:
            filtered_data1.append(pokemon)
    print(f"Filtered pokemon with {filter1}: {filtered_data1}")

    filtered_data2 = []
    for element in data.keys():
        items = data[element]
        if filter2 in items:
            filtered_data2.append(element)
    print(f"Filtered pokemon with {filter2}: {filtered_data2}")

```

```
17      formatted_data = []
18      for item in filtered_data1:
19          if format_type == "uppercase":
20              formatted_data.append(item.upper())
21          elif format_type == "lowercase":
22              formatted_data.append(item.lower())
23
24      for item in filtered_data2:
25          if format_type == "uppercase":
26              formatted_data.append(item.upper())
27          elif format_type == "lowercase":
28              formatted_data.append(item.lower())
29
30      return formatted_data
31
32  def main():
33      my_pokemon = {
34          "Pikachu"  : ["Electric"],
35          "Charizard": ["Fire", "Flying"],
36          "Bulbasaur": ["Grass", "Poison"],
37          "Chansey"  : ["Normal"],
38          "Zapdos"   : ["Electric", "Flying"],
39          "Vileplume": ["Grass", "Poison"]
40      }
41      data = process_data(my_pokemon, "Electric", "Grass", "uppercase", 5, "string")
42      print(f"List of formatted pokemon: {data}")
43
44  if __name__ == "__main__":
45      main()
```

3.3 [15 pts]: The code above contains several code smells. Your task is to: (i) Identify and explain the code smells present in the code. For each code smell, specify the exact line(s) where the issue occurs. (ii) Describe the refactoring steps needed to improve the maintainability of the code.

3.4 [10 pts]: Write a function that receives a dictionary of Pokemon and a single character. The dictionary should have the Pokemon name as a key and a corresponding list of strings indicating its type(s) (e.g., similar to the `my_pokemon` dictionary in the code above).
The function should return a new dictionary containing only the Pokemon whose names start with the specified character as keys, along with their corresponding types as values. Consider the following cases:

- If the user has specified a character that is longer than 1, then raise a value error with the message *"Letter must be a single character."*
- If no Pokemon with the specified letter is found, return an empty dictionary.
- You can assume that all Pokemon names will start with uppercase (i.e., no need to handle upper or lowercase letters when matching the letters to Pokemon name).

For instance, for the `my_pokemon` dictionary in the main method above:

- If given `"C"`, the function returns: `{"Charizard": ["Fire", "Flying"], "Chansey": ["Normal"]}`

- If given `"B"`, the function returns: `{"Bulbasaur": ["Grass", "Poison"]}`
- If given `"I"`, the function returns: `{}`
- If given `"Pi"`, the function raises: `ValueError("Letter must be a single character")`

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 a divisor integer as inputs. The function should return the number of elements in the list that are divisible by the specified divisor. If the target integer does not appear, the function should return 0. **Important:**

- You **must** use recursion in your solution.
- You can assume that the list contains only integer. The divisor is always greater than zero.
- The elements in the list can be in any order (i.e., the list is not sorted).
- When writing your algorithm, we recommend including lines of code to help with the call stack question.

```
Examples:
Case 1:................................ | Case 3:................................
Input: [30, 12, -5, 7, 25], divisor: 5  | Input: [2, 8, 10], divisor: 7
Output: 3                               | Output: 0

Case 2:................................ | Case 4:................................
Input: [3, 6, 9, 12, 15], divisor: 3    | Input: [], divisor: 3
Output: 5                               | Output: 0
```

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: [30, 12, -5, 7, 25], divisor: 5`.

Your stack trace should include, at each step: (i) function name and parameter values, (ii) variables and corresponding values, (iii) lines of code when calling (or returning to) the recursive step. As a reminder of how to illustrate the call stack, Figure 1 includes an example of a call stack we did for the recursion lecture.

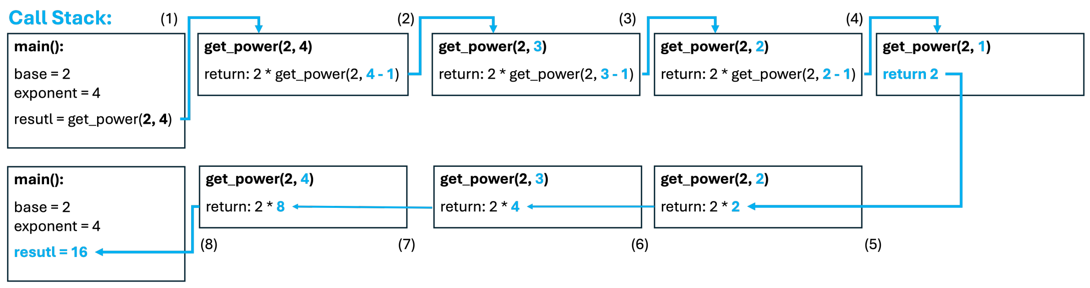

Figure 1: Overview of the call stack for the recursive function `get_power()`.