University of Zurich[UZH]

s.e.a.l. software evolution & architecture lab

Informatik I – EProg HS17

Midterm

General guidelines:

- It is possible to achieve **105 points**, achievable through completing all tasks.
- You have **75 minutes** to complete the test.
- You must **not** use any additional resources. Non-native speakers may use a dictionary.
- Please check that you have received all 14 pages of this exam.
- Please write down your **name** and your **student number** at the bottom of **each** page.
- If you use any unfair or unauthorized resources or if you copy from a fellow student, you have to hand in your test immediately and it will be considered as failed. Additionally, there will be a disciplinary enquiry.
- Do not remove the stapling of this test.
- **Please note that we included a list of helpful functions at the end of the test.**

- The English version of this midterm test is a service for the students. If there are any differences to the German version of this test, the German version is decisive.
- You **must** sign the confirmation on the German version of this test and write down all your answers there. Answers on the English version will be ignored.

Hereby, I confirm that:

- I have read and understood these guidelines.
- I have written the exam in reasonable conditions.

Signature: _____ Date: _____

Task 1	Task 2	Task 3	Task 4	Task 5	Task 6	Task 7	Total
10	5	5	5	20	30	30	105

1 Task: Data Types, Variables and Expressions (10 Points)

a) General expressions: Write down the output for each of the following snippets, if you would run them with a Python interpreter. Write "error" in case an error occurs.

1 Point

```
1  sentence = "The quick brown fox jumps over the lazy dog"
2  print(sentence[0:3])
```

---

1 Point

```
1  sentence = "The quick brown fox jumps over the lazy dog"
2  print(sentence[-3:-1])
```

---

1 Point

```
1  sentence = "The quick brown fox jumps over the lazy dog"
2  print(sentence[-8:])
```

---

1 Point

```
1  print((9 ** 2 - 4 ** 3 + 8) ** 0.5)
```

---

1 Point

```
1  account_stm = "The balance of account no. %s is %.4f"
2  print(account_stm % ("CH123 888 222", 123.225))
```

---

b) Types: Assume that the following function is defined in your environment.

```
1  def fun(n):
2      if n > 0:
3          return "x"
```

What are the types of the following expressions that are passed to the `type` operator?

1 Point

```
1  type(17)
```

---

1 Point

```
1  type(3.1415)
```

---

1 Point

```
1  type(fun)
```

---

1 Point

```
1  type(fun(5))
```

---

1 Point

```
1  type(fun(-2))
```

---

2 Task: Control Flow and Iteration (5 points)

1. Write a Python script that asks the user to enter a number. Print the message "Positive" if the number is greater or equal to 0, otherwise print "Negative". (2.5 Points)

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
```

2. Write a Python script that computes the sum of all numbers between 55 and 110, including both, and that prints the calculated result. (2.5 Points)

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
```

3 Task: Functions (5 Points)

1. Write a Python function that adds two numbers: `a` and `b`. You should return `a + 1`, if no value is provided for `b`. (2.5 Points)

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
```

2. Write a Python function `is_odd` that accepts a number as an argument. The function should returns `True` if this number is odd and `False` otherwise. (2.5 Points)

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
```

4 Task: Data Structures (5 points)

1. Implement a Python function that receives a list of numbers `lst` as an argument and that prints the maximum value. You can use other helper functions if you consider it necessary, but you are not allowed to use the Python function `max`. (3 Points)

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
```

2. Implement a Python script: create a dictionary `a`, in which you store your `first_name`, `last_name`, and `student_id`. Create a similar dictionary `b` for "Max Mustermann" (Id: 12345). Add both dictionaries to a list and print all first names using a loop. (2 Points)

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
```

5 Task: Validate Username (20 points)

Write a Python function `validate_username` to validate a `username` argument by checking that it satisfies the following conditions:

- Has at least 3 characters.
- Has at most 16 characters.
- Contains only letters (A-Z, a-z) or the special characters "_" and ".".
- Starts with an uppercase letter.

The function should return a *tuple* that contains the two values `True` and `None` if the username is valid. Should the username be invalid, then the tuple should contain `False` together with an appropriate error message.
Please review the following examples that illustrate the expected behavior of the function.

```
 # prints (False, "Username too short!")
print(validate_username("aa"))
# prints (False, 'Username too long!')
print(validate_username("aaaaaaaaaaaaaaaaaaaa"))
# prints (False, 'Username contains invalid character!')
print(validate_username("aaaaaaa_aaaa2"))
# prints (False, 'Username does not start with an uppercase letter!')
print(validate_username("_jjjjjjjj"))
# prints (False, 'Username does not start with an uppercase letter!')
print(validate_username("valid_username"))
# prints (True, None)
print(validate_username("Valid_username"))
# prints (True, None)
print(validate_username("Az_."))
```

Now, implement the function `validate_username` that checks these rules.

```
```

6 Task: Find Last Occurrence (30 points)

Write a function that, given a string `line` and a character `ch` as arguments, returns the last position of `ch` in `line`. The function should return -1, if `line` is either `None` or empty, or if `line` does not contain the character `ch`. You are not allowed to reuse predefined Python functions like `str.index` or `str.find` in your solution.

Please review the following examples that illustrate the expected behavior of the function.

```
1  # returns 0
2  find_last("The quick brown fox jumps over the lazy dog", "T")
3  # returns 42
4  find_last("The quick brown fox jumps over the lazy dog", "g")
5  # returns 4
6  find_last("The quick brown fox jumps over the lazy dog", "q")
7  # returns -1
8  find_last("The quick brown fox jumps over the lazy dog", "X")
9  # returns -1
10 find_last("", "a")
11 # returns -1
12 find_last(None, "a")
```

a) Implement the `find_last` function using a *recursive* solution. (20 points)

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
```

b) Implement the `find_last` function using an *iterative* solution. (10 points)

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
```

7 Task: Build Words Dictionary From File (30 points)

Implement a function `build_words_dict` that receives the `filename` of a text file as an argument and a value `min_occurrences`. The function should build and return a dictionary containing pairs of words and the number of their occurrences in the `filename`. The dictionary should also contain a special word `UNK` that counts the total occurrences of rare words (words that appear less than `min_occurrences` times), these words should not be included in the returned dictionary. You can assume that the file has already been preprocessed and does not contain any punctuation and that all words are lower case.

Please consider the following file contents as an example.

```
1 the quick brown fox jumps over the lazy dog
2 the slow red fox jumps over the lazy dog
```

If the expected function would read the file with `min_occurrences=2`, the resulting dictionary should contain the following values:

```
1 jumps = 2
2 UNK = 4
3 fox = 2
4 lazy = 2
5 dog = 2
6 the = 4
7 over = 2
```

Now, implement the function that can build such a dictionary from a file.

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
```

8 Useful Python Functions

8.1 Strings

str.isupper() Returns True if all letter characters in the string `str` are uppercase and there is at least one uppercase letter, False otherwise (e.g., `'A'.isupper()` is `True`, `'Ab'.isupper()` is `False`).

str.islower() Returns True if all letter characters in the string `str` are lowercase and there is at least one lowercase letter, False otherwise.

str.split(sep) Returns a list of the words of the string `str` separated by `sep`. If the optional argument `sep` is absent or None, the words are separated by arbitrary strings of whitespace characters (space, tab, newline, return, formfeed).

str.join(words) Returns a string by concatenating the list of words with intervening occurrences of `str`.

str.isalpha() Returns True if all characters in the string are alphabetic and there is at least one character, False otherwise (e.g. `'A'.isalpha()` returns `True`).

str.isdigit() Returns True if all characters in the string are digits, False otherwise.

str.startswith(prefix) Returns True if string starts with the prefix, False otherwise.

len(str) Returns the length of the string `str`.

8.2 Lists

list.append(x) Add an item to the end of the list; equivalent to `a[len(a):] = [x]`.

list.remove(x) Remove the first item from the list whose value is `x`. It is an error if there is no such item.

list.index(x) Return the index in the list of the first item whose value is equal to `x`. It is an error if there is no such item.

list.count(x) Return the number of times `x` appears in the list.

8.3 Dictionaries

dict.has_key(key) Returns `True` if the dictionary has key `key`, `False` otherwise.

dict.keys() Returns a list of the dictionary's `dict` keys.

dict.items() Returns a list of dictionary's (key, value) tuple pairs.

dict.values() Returns a list of dictionary's values.

dict.get(key, default=None) For key `key` it returns the value of `default` if the key is not in the dictionary.

dict.pop(key) Removes the element from the dictionary with key `key` and returns the value.

8.4 Files

open(filename, 'r') Opens the file named `filename` for reading and returns a file object.

open(filename, 'w') Opens the file named `filename` for writing and returns a file object.

f.close() Closes the file object `f`.

f.readline() Returns a line from the file object `f`.

f.readlines() Returns all lines from the file object `f`.