University of Zurich^UZH

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

Informatik I
Introduction to Programming

Midterm Fall 2018

General Guidelines:

- It is possible to achieve **90 points**, achievable through completing all tasks.
- You have **90 minutes** to complete the test.
- Please check that you have received all 11 pages of this exam.
- Use a **black or blue, permanent pen** for this exam. It is **not allowed** to write with **green or red pens** or with a **pencil**. Affected answers will not be considered in the grading.
- Do not remove the stapling of this test.
- Please write down your **last name** and your **student id** at the bottom of **each** page.
- You can use a **hand-written formulary** (DIN-A5, two-sided) that clearly states your name.
- Non-native speakers may use a **dictionary**.
- You must **not** use any additional resources. 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.
- Use Python for your answers, you can freely choose between version 2 and 3 and their according functions. It is not allowed to use predefined functions if the task description asks you to implement them.
- We have included a list of helpful Python functions on the last page.
- You are not allowed to change predefined method signatures or variable names in the exam.

Guidelines for the English Exam

- The English version of this midterm test is a translation service for the students.
- If differences exist between the two translations, the German version is decisive.
- You **must** sign the German version of this test and **give your answers there**.
- You can use English as the language for text answers.
- **Answers you give in this English version of the exam will be ignored!**

Task 1: General Questions 20 Points

This task lists several small Python snippets, each of which has an expression in the last line. Write down the *type* and the *value* of these expressions into the corresponding fields. Leave the *value* field empty if the expression does not return any value. In case of errors, state *NoneType* as the type and write "error" as the value. In case of endless loops, state *NoneType* as the type and write "endless loop" as the value.

Note: You do not need to name modules, just write the exact name of the type, e.g., 'int'.

Note: The snippets are invoked in separation and do not have any side effects on each other.

Note: Read the snippets very carefully. Not all answers are obvious.

a) 2 Points

```
1.2 + 3 + "5"
```

Type:		Value:	

b) 2 Points

```
i = [1, 2, 3]
i + [4,5]
i.append([6, 7])
i.extend([8, 9])
i
```

Type:		Value:	

c) 2 Points

```
b1 = "4321"
b2 = "12343"
b1[3:4] + b2
```

Type:		Value:	

d) 2 Points

```
False or False and True or False
```

Type:		Value:	

e) 2 Points

```
tmp = (lambda x: x//2, 13)
tmp[0](tmp[1])
```

Type:		Value:	

f) 2 Points

```
def e():
   for i in range(0, 10, 2):
     if i == 10:
       return i
e()
```

Type:		Value:	

g) 2 Points

```
name = "John doe"
name[5] = "D"
name
```

Type:		Value:	

h) 2 Points

```
h = {'a': 1, "b": 2.3}
tmp = None
for x in h:
   tmp = x
   break
tmp
```

Type:		Value:	

i) 2 Points

```
i = 0
while i <= 100:
   i+=1
i
```

Type:		Value:	

j) 2 Points

```
j = 0
def fun(n):
   j = n
fun(3)
j
```

Type:		Value:	

Task 2: Control Flow and Data Structures **30 Points**

In this task, you will be asked to write some simple scripts. Each subtask will introduce a small programming problem. The examples illustrate the expected behavior that your implementation must satisfy.

Note: You do not have to check for `None` arguments, but you must handle corner cases of the expected argument type (like negative integers or empty strings).

a) Control Flow 5 Points

Write a function that turns a numeric age into an age description. *Children* younger than three years are called *toddlers*. *Adulthood* starts with 18 years and one becomes *senior* with age 65. You can assume that `age` is always greater than zero.

```
def get_age_desc(age):







assert get_age_desc(1) == "toddler"
assert get_age_desc(3) == "child"
assert get_age_desc(14) == "child"
assert get_age_desc(18) == "adult"
assert get_age_desc(65) == "senior"
```

b) Iteration 5 Points

The function `compute_max_diff` receives a list of numbers as an argument. Implement the function and compute the highest absolute difference between any two consecutive numbers in the list. You can use the predefined function `abs` in your solution.

```
def compute_max_diff(numbers):




assert compute_max_diff([]) == 0
assert compute_max_diff([1]) == 0
assert compute_max_diff([2, -1]) == 3
assert compute_max_diff([2, 7, 5, 14, 12, 14]) == 9
```

c) Even/Odd Sum 5 Points

Write a function that aggregates all numbers in a list of integers. The even numbers should be *added*, the odd numbers should be *subtracted* from the result. Return a tuple that contains both the length of the original list and the calculated value.

```
def sum_even_and_odd(numbers):




assert sum_even_and_odd([]) == (0, 0)
assert sum_even_and_odd([2,4,7]) == (3, -1) # 3 values, 2+4-7=-1
```

d) Functional Programming 5 Points

Implement a function that applies an operation `op` to every element of a list `l`. Return the result in a new list and do not alter the original list.

```
def app(l, op):



l = [1, 2, 3]
def double(x): return x * 2
assert app([], double) == []
assert app(l, double) == [2, 4, 6]
assert app(l, lambda x: x * 3) == [3, 6, 9]
assert l == [1, 2, 3]
```

e) Counting 5 Points

Write a function that counts the frequency of each character that is contained in a string. Return the result in a dictionary.

```
def count_chars(s):



assert count_chars("") == {}
assert count_chars("aA .") == {"a": 1, "A": 1, " ": 1, ".": 1}
assert count_chars("abbCaabb") == {"a": 3, "b": 4, "C": 0}
```

f) Function Calls 5 Points

Write a function that counts the frequency of an arbitrary character in a string. Reuse the function `count_chars` from the previous task.

Note: You can assume that `c` always contains exactly one character.

```
def count_occurrences(s, c):















assert count_occurrences("", "a") == 0
assert count_occurrences("x", "b") == 0
assert count_occurrences("abbccc", "c") == 3
```

Task 3: Recursive Functions 20 Points

Implement the function `nested_sum` that calculates the sum of all elements of a list `l`. These elements are either integers or other nested lists that follow the same pattern. The calculated sum should consider the nesting depth in the list as weight, i.e., values on the first level have a weight of 1, on the second level a weight of 2, and so on. You can find concrete examples at the end of the block.

Note: You can assume that `l` is always valid, i.e., only contains integers and nested list. `l` can also be empty. Make sure that your implementation does not violate these assumptions.

Note: Your solution must use recursion.

```python
def nested_sum(l, depth=1):



assert nested_sum([]) == 0
assert nested_sum([1, 2, 3]) == 6 # 1*(1+2+3)
assert nested_sum([1, [2]]) == 5 # 1*(1+2*(2))
assert nested_sum([1, [2, [3]]]) == 23 # 1*(1+2*(2+3*(3)))
assert nested_sum([[1, 2], 3, [4, 5]]) == 27 # 1*(2*(1+2)+3+2*(4+5))
```

Task 4: Files 20 Points

You are working in the university administration and you would like to analyze the exam results of all current students. The administrator has exported you the exam data into a file, but you still need to pre-process it to make it more accessible in your program.

Your file `results.csv` has the following format. Each line contains one exam result that consists of three fields, which are separated by comma: student name, course, and grade.

```
Hans,Info1,5.5
Hans,Economics,5
Petra,Info1,5.25
Petra,Economics,4.75
Petra,Math,6
Martin,Info1,5.75
```

Implement a function `read` to make the contents of such a file easy to use. To do so, read the file referenced by `path`, split each line into the three components and store them in a dictionary. The dictionary should use the *student's name* as the key and a list of all *exams* as a value; this list should contain the *course* and the resulting *grade* in a *tuple* (see `assert` example).

Note: You can assume that the referenced `path` always points to a valid file that strictly follows the schema introduced before. Also, the grade is always a valid `float` value, use it accordingly.

```
def read(path):



assert read("results.csv") == {"Hans": [("Info1", 5.5),
   ("Economics", 5)], "Petra": [("Info1", 5.25), ("Economics", 4.75),
   ("Math", 6)], "Martin": ["Info1", 5.75]}
```

Useful Python Functions

Strings

str.isupper() / str.islower() Returns `True` if all characters in the non-empty string `str` are uppercase/lowercase, `False` otherwise.

str.split(sep) Returns a list of the words of the string `str`, separated on occurrences of `sep`. If `sep` is absent or None, the string is separated by 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() / str.isdigit() Returns True if all characters of a non-empty string are alphabetic/numeric, `False` otherwise.

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

str.endswith(suffix) Returns `True` if the string ends with `suffix`, otherwise `False`.

string.find(x) Returns the starting index of `x` if it occurs in the string, otherwise -1.

Lists

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

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

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

list.count(x) Counts the occurrences of `x` in a list.

Dictionaries

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

dict.keys() Returns a list of all keys defined in dictionary `dict`.

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

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

dict.get(key, default=None) Returns the value associated with `key` or `default` if key does not exist.

dict.pop(key) Removes `key` from the dictionary and returns its former value.

Files

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

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

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

f.readline() Returns the next line of file handle `f`.

f.readlines() Returns all lines of file handle `f`.

os.path.isfile(file) Returns `True` if `file` is an existing regular file.

Other

isinstance(obj, type) Returns `True` if `obj` has a type compatible to `type`, `False` otherwise.

len(obj) Return the length of an object. `obj` may be a sequence (e.g., string, list, etc) or a collection (e.g., dictionary).

sorted(sequence) Return a new sorted list from the items in sequence.

abs(x) Return the absolute value of a number.