University of Zurich[UZH]

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

Informatik I
Introduction to Programming

Midterm Fall 2019

General Guidelines:

- It is possible to achieve **60 points**, achievable through completing all tasks.
- You have **60 minutes** to complete the test.
- Please check that you have received all 9 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 3.7** and its corresponding functions for your answers. 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.
- **You acknowledge the following points by returning your exam:**
  - I have read and understood these guidelines.
  - I am mentally and physically fit to solve the exam.
  - The room is adequate and I can work on the exam undisturbed.
- Any disturbance during the exam has to be reported to the supervisory staff immediately.

Provide the following information in **capital letters** and with **clear** hand-writing:

Last Name: | | | | | | | | | | | | | | | | | |

First Name: | | | | | | | | | | | | | | | | | |

Matrikelnummer: | | | - | | | | - | | | |

Task 1	Task 2	Task 3	Task 4	Task 5	Total
20	10	10	10	10	60

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. 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: It is enough to name the simple type without mentioning the module, e.g., *int* or *integer*.

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

```
c = []
c.extend("abc")
c
```

Type: Value:

b) 2 Points

```
8 + 3.0 + "9.1"
```

Type: Value:

c) 2 Points

```
c1 = c2 = []
c1.append(1)
c2.append(2)
c1+c2
```

Type: Value:

d) 2 Points

```
d = [(1), (2,3), (4,5,6)]
d[0]
```

Type: Value:

e) 2 Points

```
names = [
   ["adrian", "alina"],
   ["ben", "beat"],
   ["cornelius", "christoph"]
]
names[-1][-2][2:3]
```

Type: Value:

f) 2 Points

```
f = ['a', 'h', 'b']
f[3] = 'g'
f
```

Type: Value:

g) 2 Points

```
g = 0
for n in [1, 2, 3.0, 4.0]:
  if n%2:
    g += n
  break
g
```

Type: Value:

h) 2 Points

```
def fun_h():
   h = 0
   for i in range(10):
      h += i
fun_h()
h
```

Type: Value:

i) 2 Points

```
i = (lambda x: 0.5 * x**2, 4)
i[1]
```

Type: Value:

j) 2 Points

```
a = [1,2,3]
b = [1,2,3]
c = b
if a == c:
   j = a is c
else:
   j = b == c
j
```

Type: Value:

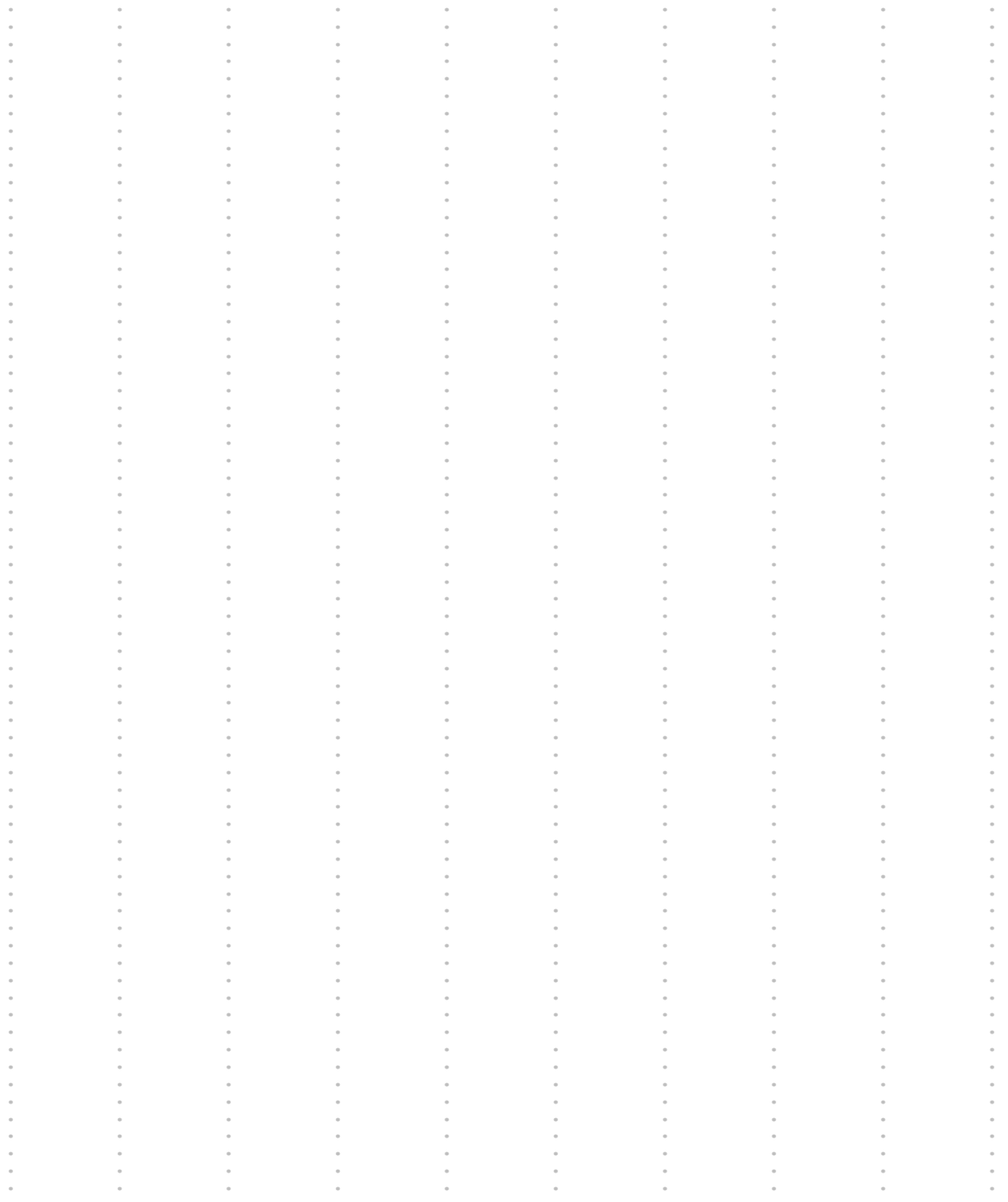
Task 2: Iteration — 10 Points

Write a function with two arguments: `l` is a list of integers and `target` is an integer value. The function should return a tuple of the *distinct* indices of the *first* two numbers that add up to `target`.

Note: You can assume that `l` is always a list and that `target` is always an integer.

Note: You cannot use the same element in `l` twice.

```
def find_sum(l, target):
```



```
assert find_sum([], 9) == None
assert find_sum([1], 2) == None
assert find_sum([1, 1], 2) == (0, 1)
assert find_sum([1, 7, 2, 11], 9) == (1, 2)
assert find_sum([1, 2, 3, 4], 5) == (0, 3)
```

Task 3: Working With Files — 10 Points

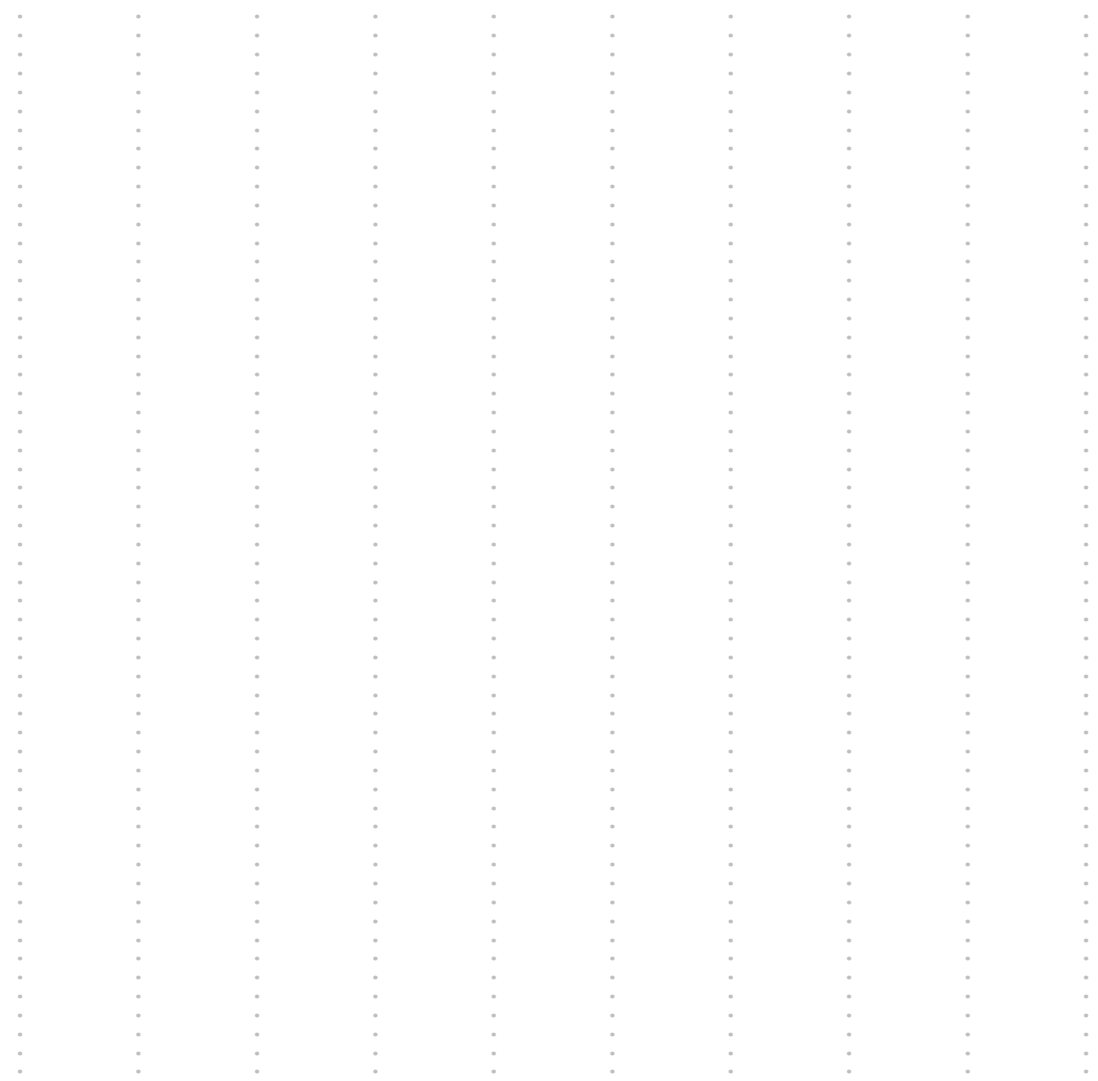
Write a function `count_keywords` that takes two input arguments, a `path` to a file and a list of `keywords`. The function should count how often the provided `keywords` occur in the file. You can assume that the file always exists and that it only contains letters, spaces, and new lines. For example:

```
Midway upon the journey of our life I found myself within a FOREST dark
for the straightforward pathway had been lost

Ah me How hard a thing it is to say what was this forest savage rough and
stern which in the very thought renews the fear So bitter is it death is
little more but of the good to treat, which there I found speak will I of
```

Count *case-insensitive* and return the results in a dictionary (word as keys, count as value). Make sure that you do not accidentally count subwords, i.e., `the` and `there` are two distinct words.

```
def count_keywords(path, keywords):
```



```
# file.txt contains the example above
assert count_keywords('file.txt', ['forest', 'the', 'found'])
         == {'the': 5, 'found': 2, 'forest': 2}
assert count_keywords('file.txt', ['black']) == {}
assert count_keywords('file.txt', []) == {}
```

Task 4: Binary Conversion — 10 Points

Write a function `to_binary` that converts a number `n` to a string of its binary representation. The most significant bit (= the highest value) should be on the left side. You can assume that `n` is always a positive integer. In contrast to other programming languages, in which `int` typically has a maximum value, a Python 3 `int` is unbounded. Your solution must work for every possible $n \geq 0$.

```
def to_binary(n):
```

```
assert to_binary(0) == "0"
assert to_binary(1) == "1"
assert to_binary(2) == "10"
assert to_binary(19) == "10011"
```

Task 5: Recursive Functions 10 Points

Implement the function `find_max` that searches for the maximum value in a list `l`. The list can contain integers or nested list that should be searched recursively. If no actual numbers are contained in the lists, the result should be `None`. Your solution *must* use recursion.

```
def find_max(l):
```

```
assert find_max([]) == None
assert find_max([1, 12, -3]) == 12
assert find_max([2, [1]]) == 2
assert find_max([1, [-7, [13, [4]], [[[[5]]]] ]]) == 13
```

Useful Python Functions

Strings

str.upper() / str.lower() Returns a new string, in which all letters are converted to *uppercase/lowercase*.
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.
string.replace(old, new) Returns a copy of the string where all the occurrences of the substring `old` are replaced with the substring `new`.

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

key in dict 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.

TestCase

assertEqual(a, b) Test that `a` and `b` are equal or fails the test, otherwise.
assertTrue(a) / assertFalse(a) Test that `a` is `True` / `False`.
assertRaise(Type) Can be used in a `with` statement to make sure that the enclosed code `raises` the given error type. The test fails, if not.