Common Mistakes, Problem Solving

Dr. Carol Alexandru-Funakoshi

University of Zurich, Department of Informatics

print vs. return

- `print` and `return` are completely different things!
- `print`
  - Is a function that outputs a String to the console (usually)
  - Returns None
- `return`
  - Is a keyword
  - Returns a value of any type from a function (and ends the function)

```
>>> def say_my_name(name):
...     print(name)
...
>>> x = say_my_name("Bob")
Bob
>>> type(x)
<class 'NoneType'>
```

returns None

```
>>> def return_my_name(name):
...     return name
...
>>> y = return_my_name("Bob")
>>> type(y)
<class 'str'>
```

returns a String

Variable scoping

- Do NOT declare variables outside of your solution function
- Variables outside the function are in global scope
- If the solution function is called more than once, its behavior will be different!
- Do **not** use `global`!

```
s = 0
def mysum(l):
    global s
    for e in l:
        s += e
    return s

assert(mysum([1,2,3]) == 6) # passes
assert(mysum([1,2,3]) == 6) # fails, 12 != 6
```

Trouble with if/elif/else

- Every `if` starts a new condition expression!
- Use `elif` to continue an expression!

```
def numbername(x):
    res = ""
    if x == 1:
        res = "one"
    if x == 2:
        res = "two"
    if x == 3:
        res = "three"
    else:
        res = "I can only count to three"
    return res


print(numbername(2))
```

Trouble with if/elif/else

- Every `if` starts a new condition expression!
- Use `elif` to continue an expression!
- Or: be smart in how you use return

```
def numbername(x):
    if x == 1:
        return "one"
    if x == 2:
        return "two"
    if x == 3:
        return "three"
    return "I can only count to three"


print(numbername(2))
```

Returning **None** by accident

- If you're supposed to return a value, make sure you actually do in all cases!

```
def index_first_even(x):
    if x != []:
        for i, n in enumerate(x):
            if n % 2 == 0:
                return i
    else:
        return -1


print(index_first_even([1,2,3])) # 1
print(index_first_even([1,3,5])) # None
```

Returning **None** by accident

- If you're supposed to return a value, make sure you actually do in all cases!

- Solutions that *look* better are usually (but not always) better
- Simple solutions are often better than complicated ones
- Think before you implement!

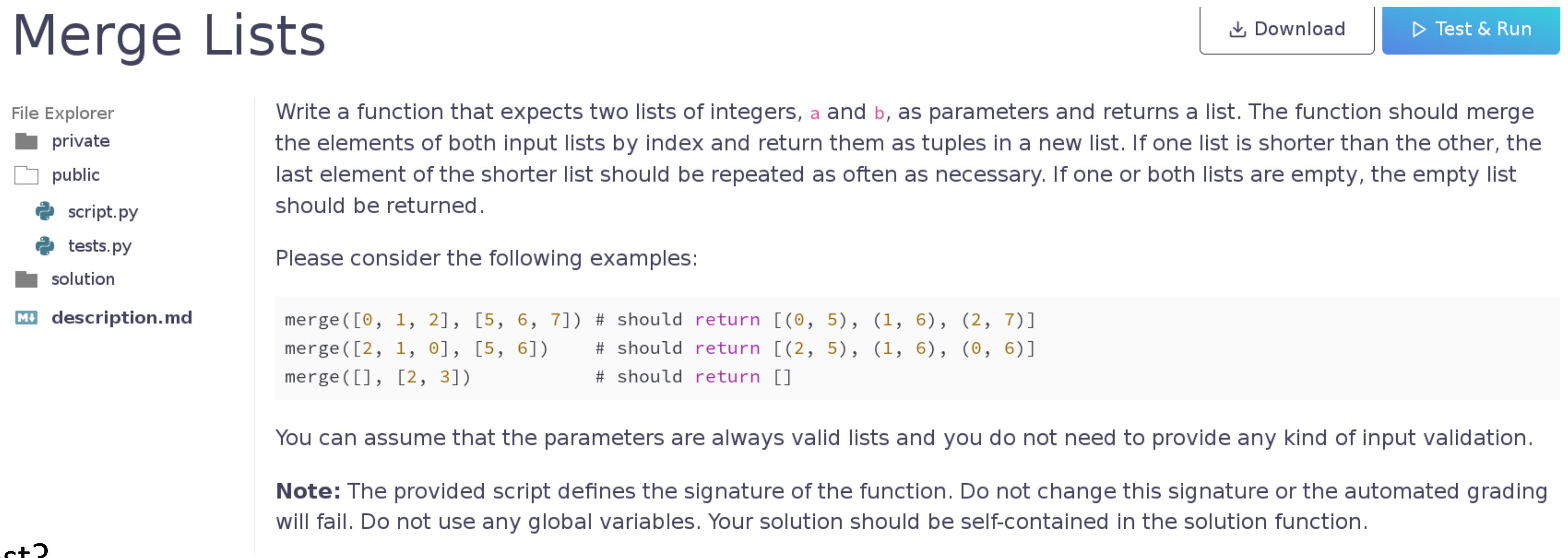
```
def index_first_even(x):
    for i, n in enumerate(x):
        if n % 2 == 0:
            return i
    return -1


print(index_first_even([1,2,3])) # 1
print(index_first_even([1,3,5])) # -1
```

Important! Test all input cases!!!

- A quality assurance engineer walks into a bar.
  - orders a beer
  - orders 2 beers
  - orders 0 beers
  - orders 2^{2049} beers
  - orders -1 beers
  - orders 2.75 beers
  - orders a lizard
  - orders a agh)(!^%@_05"; drop table "account";--
  - orders a `\n\t\x00`

Important! Test all input cases!!!



Merge Lists

Write a function that expects two lists of integers, a and b, as parameters and returns a list. The function should merge the elements of both input lists by index and return them as tuples in a new list. If one list is shorter than the other, the last element of the shorter list should be repeated as often as necessary. If one or both lists are empty, the empty list should be returned.

Please consider the following examples:

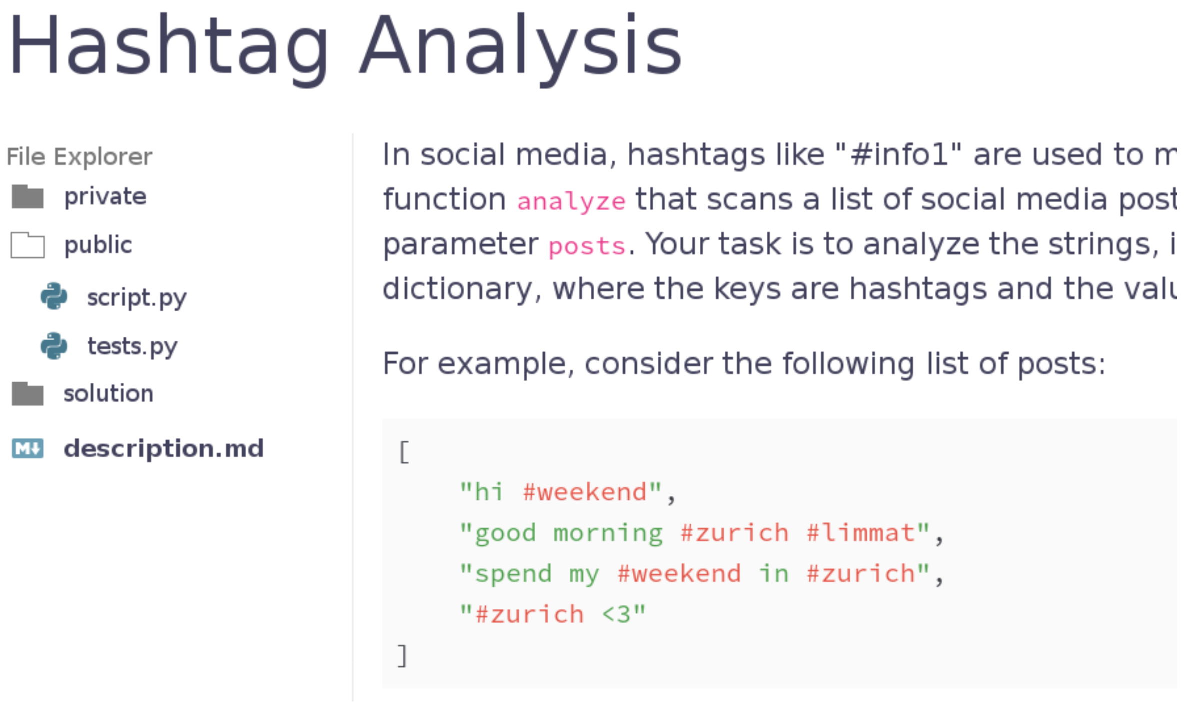
```
merge([0, 1, 2], [5, 6, 7]) # should return [(0, 5), (1, 6), (2, 7)]
merge([2, 1, 0], [5, 6])    # should return [(2, 5), (1, 6), (0, 6)]
merge([], [2, 3])           # should return []
```

You can assume that the parameters are always valid lists and you do not need to provide any kind of input validation.

Note: The provided script defines the signature of the function. Do not change this signature or the automated grading will fail. Do not use any global variables. Your solution should be self-contained in the solution function.

- What to test?
  - a empty, b not empty
  - b empty, a not empty
  - a and b empty
  - a and be same non-zero length
  - a shorter than b
  - b shorter than a

Important! Test all input cases!!!


Hashtag Analysis

File Explorer
- private
- public
  - script.py
  - tests.py
- solution
- description.md

In social media, hashtags like "#info1" are used to m
function `analyze` that scans a list of social media post
parameter `posts`. Your task is to analyze the strings, i
dictionary, where the keys are hashtags and the valu

For example, consider the following list of posts:

```
[
    "hi #weekend",
    "good morning #zurich #limmat",
    "spend my #weekend in #zurich",
    "#zurich <3"
]
```

- What to test?
  - `[ "", "#", "##", "###", "#-", "#1",`
    `"#a", "#b#c", "#d #e", " #f", "g#h", "#ii#jjjj",`
    `".#k", ".#l.", "--#m--", "##n", "-#o ", "#p-", " #q "]`

Review this!

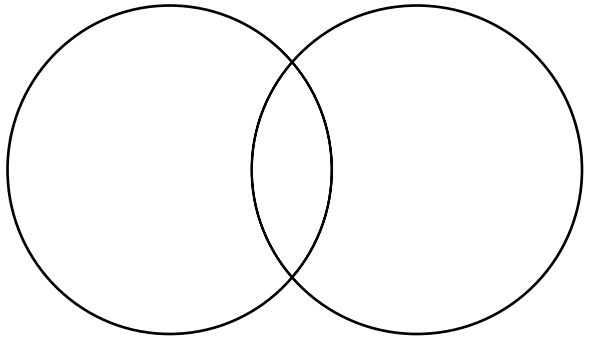
- You must know how functions work in Python!
  - A function has zero or more parameters; some could be optional
  - Know how to call a function
  - Know that functions can be passed as parameters and returned
  - Global variables vs. parameters
- Know how to use lists, tuples, dicts
  - Know about enumerate, .values(), .items(), x in y, etc...
- Know about string manipulation (find(), split(), strip(), join(), etc.)

A few more things...

- Using the `global` keyword is 100% forbidden.
  - You shouldn't be specifying variables outside your solution function anyway
- Read the task carefully
  - "a list", "a tuple" or "a dictionary" implies that these could be empty. Otherwise, the task would explicitly say "a non-empty list", "a non-empty dictionary", etc...
  - Mind terms such as "non-negative", "positive", "integer", "number", etc.
- Prepare your environment!
  - IDE ready? Most important Python documentation API docs open? Lecture slides at hand? Battery full or plugged into wall? Stable internet connection?

Sets can be useful

- A set is an unordered collection of unique values
- You can construct a set from a list or tuple by calling set() on it.
- Supports operations like add(), remove(), &, |, -, ^



```
>>> x = {1, 2, 3, 4, 4}
>>> y = set([3, 4, 5])
>>> x
{1, 2, 3, 4}
>>> y
{3, 4, 5}
>>> x & y
{3, 4}
>>> x | y
{1, 2, 3, 4, 5}
>>> x - y
{1, 2}
>>> x ^ y
{1, 2, 5}
```