COMP10001

2023 Sem 1

Foundations of Computing

Tutorial/Workshop ○ ○ ○ ○

Week 8

Today's Tutorial

1. libraries and defaultdict
2. Debugging
3. Types of errors

Libraries

A **group** of methods and/or variables

Gives us more things to use

import <library>

from <library> import <name>

defaultdict

a data type from the **collections** library

from collections **import** defaultdict

same as dictionary but does not throw **KeyError**

if accessing keys that don't exist, will instead initialise them

defaultdict

initialise it with the desired type

will add in a key with the **empty** of that type

useful for counting frequencies

main.py +

```
1  from collections import defaultdict
2
3  i_dict = defaultdict(int)
4  print(i_dict["age"])
5
6  s_dict = defaultdict(str)
7  print(s_dict["Bob"])
8
9  l_dict = defaultdict(list)
10 print(l_dict["flowers"])
```

```
0

[]


** Process exited - Return Code: 0 **
Press Enter to exit terminal
```

Exercise 1

Exercise 1 answer

1. Rewrite the following with a default dictionary

```
my_dict = {}
for i in range(10):
    if i % 3 in my_dict:
        my_dict[i % 3].append(i)
    else:
        my_dict[i % 3] = [i]
```

A:

```
from collections import defaultdict

my_dict = defaultdict(list)
for i in range(10):
    my_dict[i % 3].append(i)
```

Debugging

a "bug" is an error in the code

to fix the bug:

- find sections which might be causing it
- use **print** statements to check the variables

e.g. print the variable before, during, and after the suspected code section

Types of errors

Syntax error

Incorrect syntax; code will not compile

Run-time error

Code will compile, but will crash while running

e.g. index out of bounds

Logic error

Code compiles and runs, but the output is wrong

Exercises 2-4

Exercise 2 answer

2. Find the errors in the following programs, classifying them as (a) syntax, (b) runtime or (c) logic errors. Fix them with a correct line of code.

(a)
```
def disemvowel(text):
    """ Returns string `text` with all vowels removed """
    vowels = ('a', 'e', 'i', 'o', 'u')
    answer = text[0]
    for char in text:
        if char.lower() is not in vowels:
            answer = char + answer
    print(answer)
```

Exercise 2 answer

2. Find the errors in the following programs, classifying them as (a) syntax, (b) runtime or (c) logic errors. Fix them with a correct line of code.

(a)
```
def disemvowel(text):
    """ Returns string `text` with all vowels removed """
    vowels = ('a', 'e', 'i', 'o', 'u')
    answer = text[0]
    for char in text:
        if char.lower() is not in vowels:
            answer = char + answer
    print(answer)
```

- *line 4; logic/run-time (if empty string);* `answer = ''`
- *line 6; syntax;* `if char.lower() not in vowels:`
- *line 7; logic;* `answer = answer + char` *or* `answer += char`
- *line 8; logic;* `return answer`

Exercise 2 answer

```python
def disemvowel(text):
    vowels = ('a','e', 'i', 'o', 'u')
    answer = text[0]
    for char in text:
        if char.lower() not in vowels:
            answer = char + answer
    print(answer)
disemvowel("Test my code")
```

Run | Share | Command Line Arguments

```
dc ym tsTT
```

Exercise 2 answer

(b)
```
1  def big-ratio(nums, n):
2      """ Calculates and returns the ratio of numbers
3      in list `nums` which are larger than `n` """
4      n = 0
5      greater_n = 0
6      for number in nums:
7          if number > n:
8              greater_n += 1
9              total += 1
10     return greater_n / total
11
12 nums = [4, 5, 6]
13     low = 4
14 print(f"{100*big_ratio(nums,␣low)}%␣of␣numbers␣are␣greater␣than␣{low}")
```

A:

- *line 1; syntax;* `def big_ratio(nums,n):`
- *line 4; logic/(run-time as well since it would cause error as* `total` *is undefinedl);* `total = 0`
- *line 9; logic; remove one level of indentation (outside* `if` *block)*
- *line 13; syntax; remove indentation*

Exercise 3 answer

A: *Test cases to consider include: The empty list* `[]`*, a list with no negative numbers* `[0, 1, 2]` *and a list with only negative numbers* `[-1, -2, -3]`.
The debugging process will be different for everyone, but here is an example:
Begin by observing the function's failure to process the following test case.

```
lst = [-1, -2, 3]
remove_negative(lst)
print(lst)
```

Include a print statement to observe the values of the variables within the for loop.

```
def remove_negative(nums):
    for num in nums:
        print(num, nums)
        if num < 0:
            nums.remove(num)
```

Repeating the above test case, one will find that `num` *takes the value* -1 *and 3, but skips* -2 *entirely. With any luck, this will lead to the recollection/realisation that it is dangerous to remove elements from list whilst iterating over them, as Python can skip elements. Instead, the following solution may be attained:*

Exercise 3 answer

```
def remove_negative(nums):
    to_remove = []
    for num in nums:
        if num < 0:
            to_remove.append(num)

    for num in to_remove:
        nums.remove(num)
```

This question is based on a previous exam question. The code below is intended to validate a data entry, being a list of the following string elements.

(a) a *staff ID*, valid if it is a 5 digit number (e.g. `"00520"` or `"19471"`)

(b) a *first name*, valid if non-empty and only containing alphabetical letters

(c) a *password*, valid if including at least one lower-case letter, one upper-case letter, and one punctuation mark from the following `[',', '.', '!', '?']`

The function should return `True` if the data entry contains entirely valid values (according to the above rules) and `False` if any of the fields are invalid. A valid data example is: `['10001', 'Chris', 'Comp!']`

```
STAFFID_LEN = 5

def validate(data):
    staffid = data.pop(0)
    if not 10**(STAFFID_LEN-1) <= int(staffid) < 10**STAFFID_LEN:
        return False

    first_name = data.pop(0)
    if not first_name and first_name.isalpha():
        return False

    password = data.pop(0)
    contains_lower = contains_upper = contains_punct = False
    for letter in password:
        if letter.islower():
            contains_lower = True
        elif letter.isupper():
            contains_upper = True
        elif not letter.strip(',.!?␣'):
            contains_punct = True

    if not contains_lower and contains_upper and contains_punct:
        return False

    return True
```

```
STAFFID_LEN = 5

def validate(data):
    staffid = data.pop(0)
    if not 10**(STAFFID_LEN-1) <= int(staffid) < 10**STAFFID_LEN:
        return False

    first_name = data.pop(0)
    if not first_name and first_name.isalpha():
        return False

    password = data.pop(0)
    contains_lower = contains_upper = contains_punct = False
    for letter in password:
        if letter.islower():
            contains_lower = True
        elif letter.isupper():
            contains_upper = True
        elif not letter.strip(',.!?␣'):
            contains_punct = True

    if not contains_lower and contains_upper and contains_punct:
        return False

    return True
```

(a) *Valid data that is correctly classified:* `['12345', 'Kim', 'Ron!']` *(Function returns* `True` *correctly - input is valid)*

(b) *Invalid data that is correctly classified:* `['12345', 'Kim', 'RON!']` *(Function returns* `False` *correctly - password has no lowercase)*

(c) *Invalid data that is *in*correctly classified:* `['12345', '', 'Ron!']` *(Function returns* `True` *incorrectly since empty name is invalid but check is incorrect)*

(d) *Valid data that is *in*correctly classified:* `['00117', 'Kim', 'Ron!']` *(Function returns* `False` *incorrectly since staff ID is correct but check is not)*

Testing our code

Want our code to run in **all** situations

Write **test cases**

Think of all possible types of inputs

Such as large input, small input, empty input, divide by 0

Make sure the code checks and handles them

WORKSHOP

Grok, problems from sheet, ask me questions :)

See you next week!