

Foundations of Computing

Tutorial/Workshop ○ ○ ○ ○

Week 7

Today's Tutorial

○ ○ ○ ○

1

Comments, docstrings, variable names

2

Namespaces

3

Timely returns

Commenting



Explain what code does

Easier for others to **read and understand**

Also helps us reading our own code in future

Do excessive comments slow down a program?

No! They are discarded before the program runs

Docstrings



Written per function

Inputs + outputs of function

```
def ave_len(words):  
    """ Accepts a sequence of strings `words` as input and returns  
    the average length of strings in that sequence as a float.  
    """  
    count = 0  
    for word in words:  
        count += len(word)  
    return count/len(words)
```

Exercise 1



Exercise 1 answer

```
def favourite_animal(ballots):  
    """ ... """  
    tally = {}  
  
    # ...  
    for animal in ballots:  
        if animal in tally:  
            tally[animal] += 1  
        else:  
            tally[animal] = 1  
  
    # ...  
    most_votes = max(tally.values())  
    favourites = []  
    for animal, votes in tally.items():  
        if votes == most_votes:  
            favourites.append(animal)  
  
    return favourites
```

Takes a list 'ballots' as input. Counts the frequency of each animal in 'ballots', and returns a list of the most frequently voted animals.

Counts frequencies of each animal in the ballots.

Find and store the animals that received the highest number of votes.

Variable names

Picking **good variable** names makes code **readable**

Arbitrary variables like a,b,c,i,j are fine for loops but not good otherwise


Abbreviations like "temp" "avg" are common

Magic numbers


Numbers written **without being defined**

Appear **arbitrary**, with no context or meaning

```
public void setAge(int age)
{
    if(age>21)
    {
        //do something important
    }
}
```



```
final int legalDrinkingAge=21
public void setAge(int age)
{
    if(age>legalDrinkingAge)
    {
        //do something important
    }
}
```



Exercise 2



Exercise 2 answer

```
a = float(input("Enter_days:_"))
b = a * 24
c = b * 60
d = c * 60
print("There_are", b, "hours", c, "minutes", d, "seconds_in", a, "days")
```

```
HOUR_DAY = 24
MINUTE_HOUR = 60
SECOND_MINUTE = 60

days = float(input("Enter days: "))
hours = days * HOUR_DAY
minutes = hours * MINUTE_HOUR
seconds = minutes * SECOND_MINUTE
print("There are", hours, "hours", minutes,
      "minutes", seconds, "seconds in", days, "days")
```

Exercise 2 answer

```
word = input("Enter_text:_")
x = 0
vowels = 0
word_2 = word.split()
for word_3 in word_2:
    x += 1
    for word_4 in word_3:
        if word_4.lower() in "aeiou":
            vowels += 1
if vowels/x > 0.4:
    print("Above_threshold")
```

```
THRESHOLD = 0.4
```

```
text = input("Enter text: ")
n_words = 0
n_vowels = 0
words = text.split()
for word in words:
    n_words += 1
    for letter in word:
        if letter.lower() in "aeiou":
            n_vowels += 1
if n_vowels/n_words > THRESHOLD:
    print("Above threshold")
```

Mutability



An object is mutable if it can be **changed** after being created

Mutable types: lists, dictionaries, sets

Immutable types: ints, floats, strings, tuples

(not a complete list)

Namespaces



The **collection** of variables which can be used in certain parts of a program

Includes **local** and **global** variables

Namespaces

```
x = 1
```

global namespace
variable

```
def foo():
```

```
    x = 10
```

```
    y = 20
```

local namespace
variables

```
for i in range(1, 2):
```

```
    x = 100
```

```
    y = 200
```

local namespace
variables

Namespaces

```
x = 1
```

global namespace
variable

```
def foo():
```

```
    x = 10
```

```
    y = 20
```

local namespace
variables

```
for i in range(1, 2):
```

```
    x = 100
```

```
    y = 200
```

local namespace
variables

Namespaces



Will return local variables **over** global variables

A function can use it's local variables or global variables, **but not other function** variables

Namespaces



```
1  x = 1
2
3  def func_1(x):
4      x = 2
5      return x
6
7  def func_2(x):
8      y = 5
9      return y
```

```
11  print(x)
12  x = func_1(x)
13  print(x)
14  x = func_2(x)
15  print(x)
```

```
1
2
5
>
```

Namespaces

```
1 x = 1
2
3 def func_1(x):
4     x = 2
5     return x
6
7 def func_2(x):
8     y = 5
9     return y
```

```
11 print(y)
```

```
Traceback (most recent call last):
  File "<string>", line 11, in <module>
ERROR!
NameError: name 'y' is not defined
> |
```

Exercises

3-5

○ ○ ○ ○

Exercise 3 answer



```
def mystery(x):  
    x.append(5)  
    x[0] += 1  
    print("mid-mystery:", x)
```

```
my_list = [1,2]  
print(my_list)  
mystery(my_list)  
print(my_list)  
mystery(my_list.copy())  
print(my_list)
```

```
[1, 2]  
mid-mystery: [2, 2, 5]  
[2, 2, 5]  
mid-mystery: [3, 2, 5, 5]  
[2, 2, 5]
```

Exercise 4 answer



```
def invert_grid(grid):  
    new_grid = [[0] * len(grid[0])] * len(grid)  
    for i in range(len(grid)):  
        for j in range(len(grid[0])):  
            new_grid[i][j] = 1 - grid[i][j]  
    return new_grid  
  
grid = [[1, 0, 0], [0, 1, 0], [0, 0, 1]]  
print(invert_grid(grid))
```

```
def invert_grid(grid):  
    new_grid = []  
    for i in range(len(grid)):  
        new_grid.append([0] * len(grid[0]))  
        for j in range(len(grid[0])):  
            new_grid[i][j] = 1 - grid[i][j]  
    return new_grid
```

Exercise 5 answer



```
def foo(x, y):  
    a = 42  
    x, y = y, x  
    print(a, b, x, y)
```

```
a, b, x, y = 1, 2, 3, 4  
foo(17, 4)  
print(a, b, x, y)
```

```
42 2 4 17  
1 2 3 4
```

Early returns



Returning the answer **as soon as** the program has it, rather than continuing

e.g. if iterating over "Hello World" checking if 'l' is present, stop as soon as it is found **instead of checking every letter**

Early returns

main.py



```
1 sentence = "Hello World"
2
3 def check_letter(sentence, letter):
4     for i in range(len(sentence)):
5         if sentence[i] == letter:
6             return True
7     return False
8
9 print(check_letter(sentence, 'l'))
10
```


"Short circuiting"



Similar idea

Used in boolean tests

e.g. if `num != 0 and 4/num == 2`:

Protects against accidentally dividing by 0

Helper functions



A function that performs some part of the computation of another function

Makes the code easier read

Able to re-use those helper functions in other areas of the code

Exercise 6



Exercise 6 answer



```
def noletter_1(words, letter='z'):  
    for word in words:  
        if letter in word:  
            return False  
    return True
```

```
def noletter_2(words, letter='z'):  
    no_z = True  
    for word in words:  
        if letter in word:  
            no_z = False  
    return no_z
```

```
wordlist = ['zizzer'] + ['aardvark'] * 10_000_000  
print(noletter_1(wordlist))  
print(noletter_2(wordlist))
```

WORKSHOP

○ ○ ○ ○

Grok, problems from sheet, ask me questions :)

**See you
next week!**

