COMP10001

Week 7

Why is it important to write comments for the code we write? Wouldn’t it save time, storage space and processing time to write code without comments?

Why is it important to write comments for the code we write? Wouldn’t it save time, storage space and processing time to write code without comments?

- People look at code in the future (near or distant!) need to understand what it does and why we made certain choices when we wrote it

Why is it important to write comments for the code we write? Wouldn’t it save time, storage space and processing time to write code without comments?

- People look at code in the future (near or distant!) need to understand what it does and why we made certain choices when we wrote it
- Code is read more often than it is written!

Why is it important to write comments for the code we write? Wouldn’t it save time, storage space and processing time to write code without comments?

- People look at code in the future (near or distant!) need to understand what it does and why we made certain choices when we wrote it
- Code is read more often than it is written!
- It might be quicker to write code if we didn’t write comments, but we would lose time trying to debug it later!

Why is it important to write comments for the code we write? Wouldn’t it save time, storage space and processing time to write code without comments?

- People look at code in the future (near or distant!) need to understand what it does and why we made certain choices when we wrote it
- Code is read more often than it is written!
- It might be quicker to write code if we didn’t write comments, but we would lose time trying to debug it later!
- Comments are discarded before code is run, so don’t affect run-time

What is a “docstring”? What is its purpose?

What is a “docstring”? What is its purpose?

- A big comment at the top of a function that describes what it does

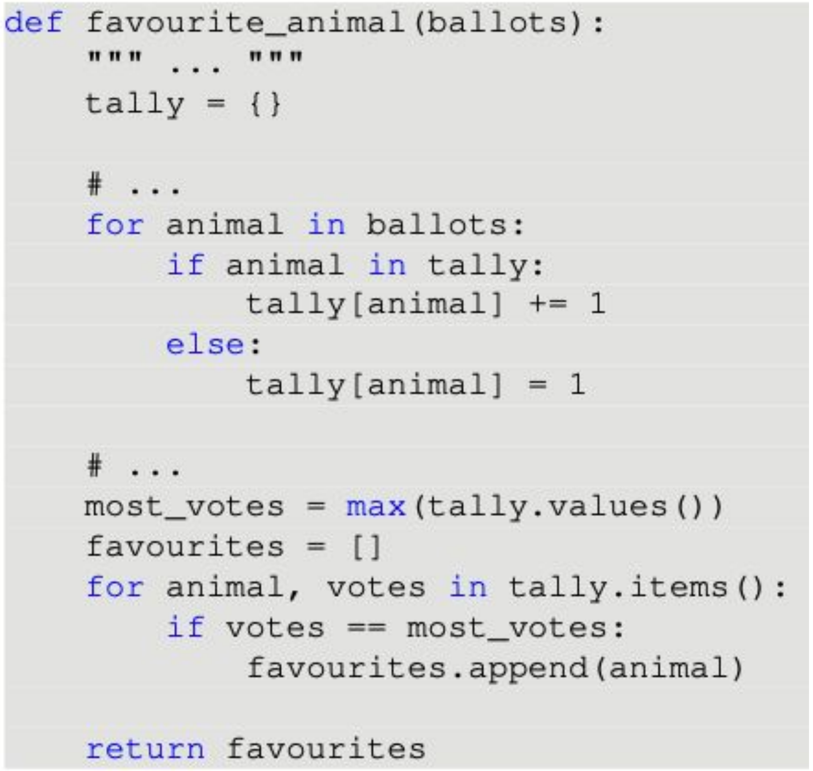
What is a “docstring”? What is its purpose?

- A big comment at the top of a function that describes what it does
- Should include:

What is a “docstring”? What is its purpose?

- A big comment at the top of a function that describes what it does
- Should include:
    - Inputs/arguments
    - Outputs
    - Briefly, what the function does

Exercise! Fill in the blanks with comments and a docstring



```python
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
```

An example for ballots is['dog', 'pig', 'cat', 'pig', 'dog'], in which case the function returns ['dog', 'pig'].

What makes a good variable name? Why are good variable names important?

What makes a good variable name? Why are good variable names important?

- Should accurately describe the thing they’re storing
- Should allow code to be readable, so the reader can follow what the code is doing

What makes a good variable name? Why are good variable names important?

- Should accurately describe the thing they’re storing
- Should allow code to be readable, so the reader can follow what the code is doing
- Bad variable names are arbitrary: a, b, c, lst, lst1, lst2, lsta, lstb

What makes a good variable name? Why are good variable names important?

- Should accurately describe the thing they’re storing
- Should allow code to be readable, so the reader can follow what the code is doing
- Bad variable names are arbitrary: a, b, c, lst, lst1, lst2, lsta, lstb
- Try to describe what the variable is storing, e.g. **name** instead of **str**

What are “magic numbers”? How do we write code without them using global constraints?

What are “magic numbers”? How do we write code without them using global constraints?

- Constants written into our code as literals

What are “magic numbers”? How do we write code without them using global constraints?

- Constants written into our code as literals
- E.g. a threshold for a passing mark

```
if mark > 0.5:
```

What are “magic numbers”? How do we write code without them using global constraints?

- Constants written into our code as literals
- E.g. a threshold for a passing mark
- Use global constants to make the purpose of magic numbers clear!

```
if mark > 0.5:
```

What are “magic numbers”? How do we write code without them using global constraints?

- Constants written into our code as literals
- E.g. a threshold for a passing mark
- Use global constants to make the purpose of magic numbers clear!

```
if mark > 0.5:
```

$\longrightarrow$

```
PASS_MARK = 0.5

if mark > PASS_MARK:
```

What are “magic numbers”? How do we write code without them using global constraints?

- Constants written into our code as literals
- E.g. a threshold for a passing mark
- Use global constants to make the purpose of magic numbers clear!

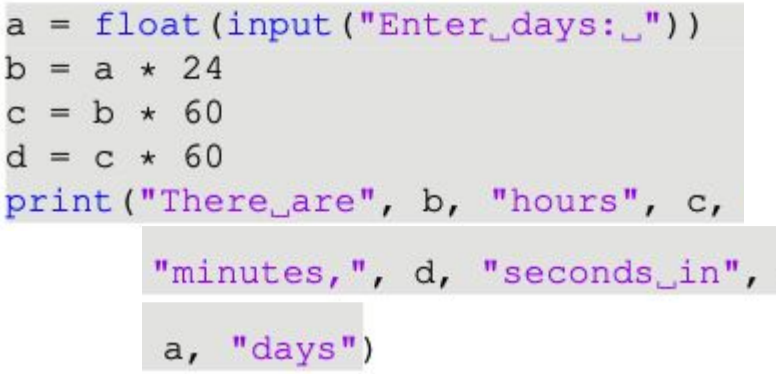
```
if mark > 0.5:
```

$\longrightarrow$

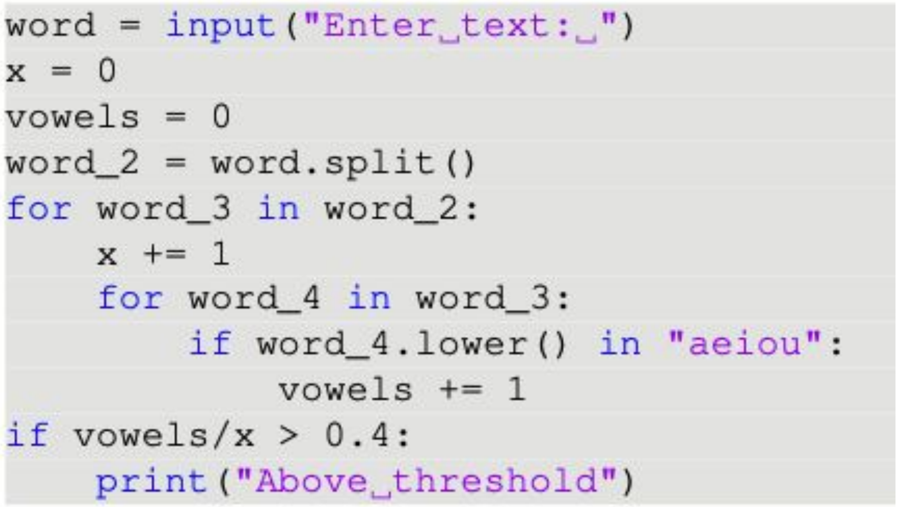
```
PASS_MARK = 0.5

if mark > PASS_MARK:
```

(Global constants are defined in the global namespace of our code, up the top, in capital letters. Their value doesn’t change.)

Consider the following programs. What are the problematic aspects of their variable name/use of magic numbers? What improvements would you make?



```
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")
```



```
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")
```

Consider the following programs. What are the problematic aspects of their variable name/use of magic numbers? What improvements would you make?

```
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")
```

Consider the following programs. What are the problematic aspects of their variable name/use of magic numbers? What improvements would you make?

```python
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")
```

```python
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")
```

What do we mean by “mutability”? Which data types are mutable out of the ones we’ve seen?

What do we mean by “mutability”? Which data types are mutable out of the ones we’ve seen?

- Mutable objects can be changed after they’re created

What do we mean by “mutability”? Which data types are mutable out of the ones we’ve seen?

- Mutable objects can be changed after they’re created
- Lists, dictionaries and set are mutable

What do we mean by “mutability”? Which data types are mutable out of the ones we’ve seen?

- Mutable objects can be changed after they’re created
- Lists, dictionaries and set are mutable
- Immutable objects can’t be changed short of creating a new object
    - e.g. a += 1

What do we mean by “mutability”? Which data types are mutable out of the ones we’ve seen?

- Mutable objects can be changed after they’re created
- Lists, dictionaries and set are mutable
- Immutable objects can’t be changed short of creating a new object
  - e.g. a += 1
- Integers, floats, strings and tuples are immutable.

What is a “namespace”?

What is a “namespace”?

- A mapping from names (of variables/functions) to objects.
- Defines the collection of variables that can be used in a certain part of your program

What do we mean by “local” and “global” namespace? What is “scope”?

- Global namespace is collection of all variable/function names available outside of any functions in a program

What do we mean by “local” and “global” namespace? What is “scope”?

- Global namespace is collection of all variable/function names available outside of any functions in a program
- When a function is called, it has a local namespace
    - This local namespace is unique to its execution and forgotten once it returns

What do we mean by “local” and “global” namespace? What is “scope”?

- Global namespace is collection of all variable/function names available outside of any functions in a program
- When a function is called, it has a local namespace
  - This local namespace is unique to its execution and forgotten once it returns
- When a variable is referred to, Python looks in the **most local namespace** first, then checks less local namespaces, then finally the global namespace

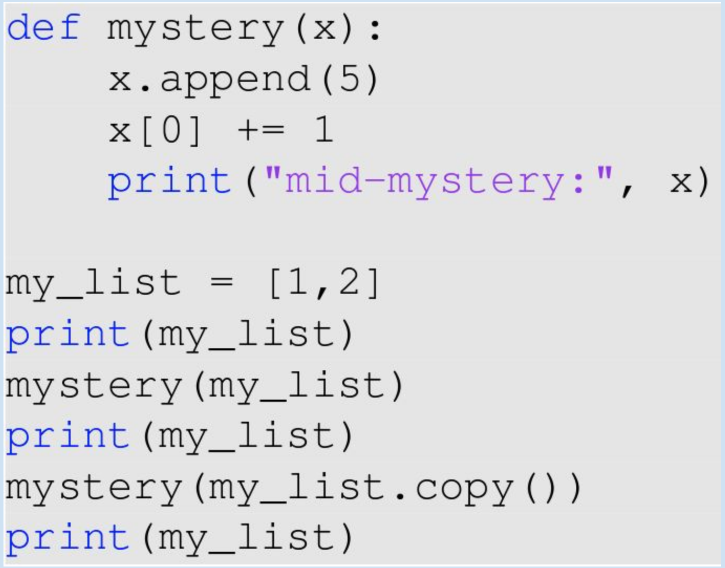
What do we mean by “local” and “global” namespace? What is “scope”?

- Global namespace is collection of all variable/function names available outside of any functions in a program
- When a function is called, it has a local namespace
    - This local namespace is unique to its execution and forgotten once it returns
- When a variable is referred to, Python looks in the most local namespace first, then checks less local namespaces, then finally the global namespace
- We discourage the editing of global variables inside a function - it’s safer to return something from your function

What do we mean by “local” and “global” namespace? What is “scope”?

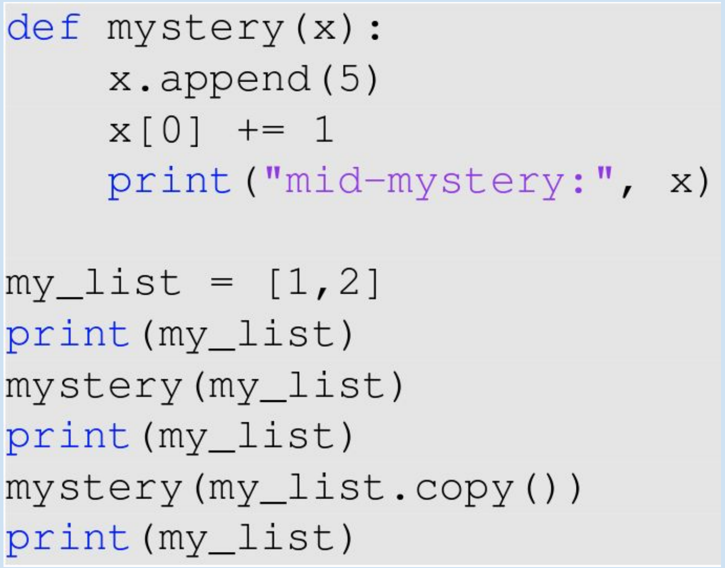
- Global namespace is collection of all variable/function names available outside of any functions in a program
- When a function is called, it has a local namespace
    - This local namespace is unique to its execution and forgotten once it returns
- When a variable is referred to, Python looks in the most local namespace first, then checks less local namespaces, then finally the global namespace
- We discourage the editing of global variables inside a function - it’s safer to return something from your function
- **Scope** is the area of a program where a namespace is used. E.g. variables in a function’s local namespace are said to be in that function’s scope.

What is the output of this code?



```
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)
```

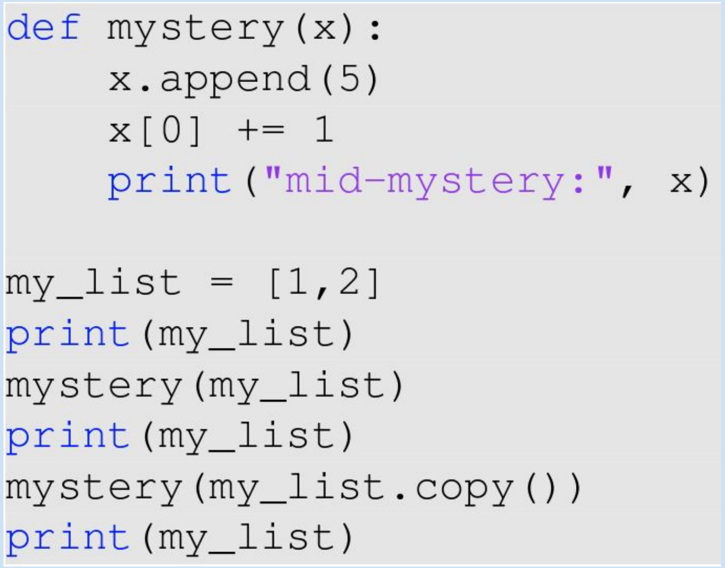
What is the output of this code?



```python
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]

What is the output of this code?

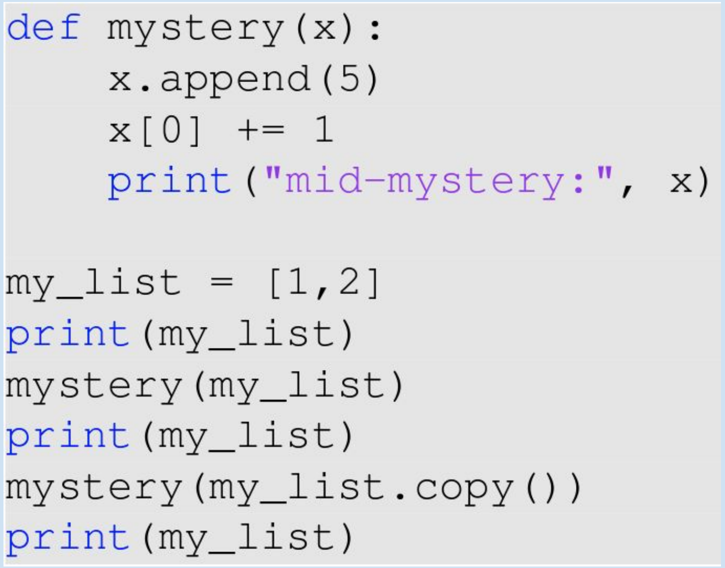


```python
def mystery(x):
    x.append(5)
    x[0] += 1
    print("mid-mystery:", x)

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]

What is the output of this code?



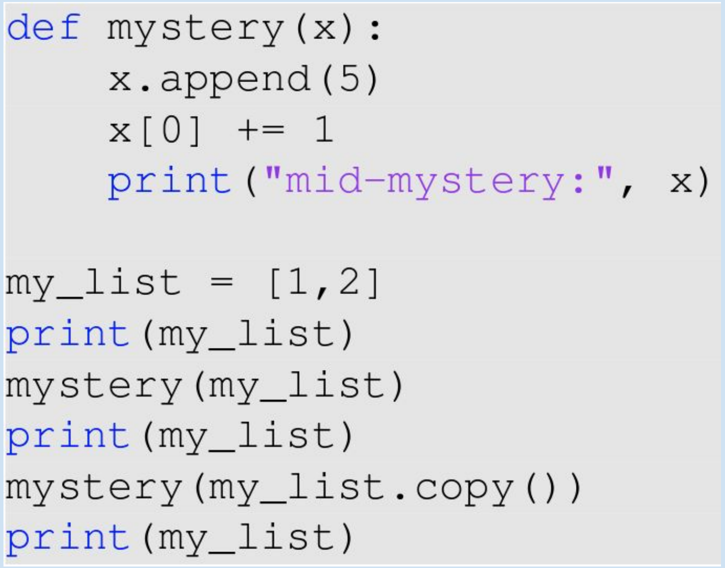
```python
def mystery(x):
    x.append(5)
    x[0] += 1
    print("mid-mystery:", x)

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]

What is the output of this code?



```python
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)
```

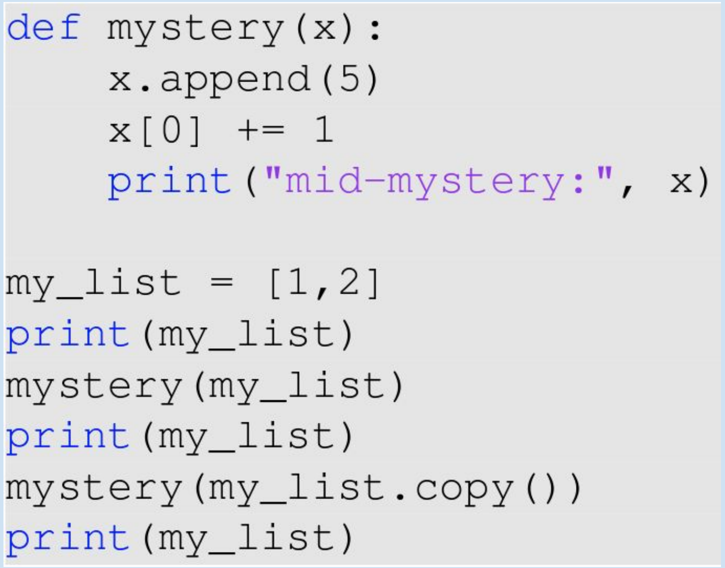
[1,2]

Mid-mystery: [2,2,5]

[2,2,5]

Mid-mystery: [3,2,5,5]

What is the output of this code?



```
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]

Something is wrong with this code. What will its output be? How could we fix it?

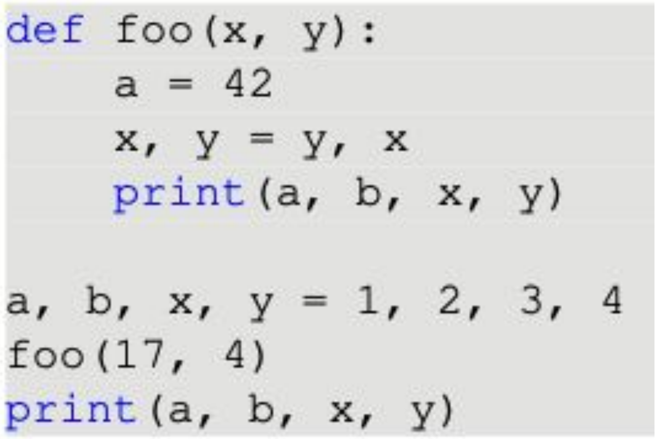
```python
def invert_grid(grid):
    new_grid = [[0] * len(grid[0])] * len(grid)
    for i in range(len(grid)):
        for j in range(len(grid[0])):
            new_grid[i][j] = 1 - grid[i][j]
    return new_grid

grid = [[1, 0, 0], [0, 1, 0], [0, 0, 1]]
print(invert_grid(grid))
```

Something is wrong with this code. What will its output be? How could we fix it?

```python
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))
```

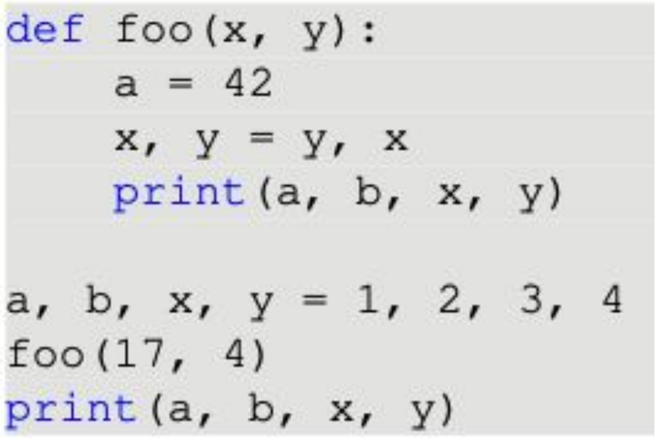
```python
def invert_grid(grid):
    new_grid = []
    for i in range(len(grid)):
        new_grid.append([0] * len(grid[0]))
        for j in range(len(grid[0])):
            new_grid[i][j] = 1 - grid[i][j]
    return new_grid
```

What is the output of the following code? Classify the variables by which namespace they belong in

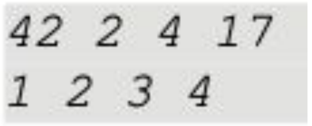


```python
def foo(x, y):
    a = 42
    x, y = y, x
    print(a, b, x, y)

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

What is the output of the following code? Classify the variables by which namespace they belong in



```python
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
```

When is it useful to “return early”? How can it make out code safer and more efficient?

When is it useful to “return early”? How can it make out code safer and more efficient?

- The answer is returned as soon as it is known, rather than waiting for a process to complete

When is it useful to “return early”? How can it make out code safer and more efficient?

- The answer is returned as soon as it is known, rather than waiting for a process to complete
    - E.g. a searching function can return as soon as the thing it’s searching for is found, rather than checking things it hasn’t yet checked.

When is it useful to “return early”? How can it make out code safer and more efficient?

- The answer is returned as soon as it is known, rather than waiting for a process to complete
    - E.g. a searching function can return as soon as the thing it’s searching for is found, rather than checking things it hasn’t yet checked.
- This increases the efficiency of our programs!

What are helper functions? How do they make our code more reusable and readable?

What are helper functions? How do they make our code more reusable and readable?

- Function that performs some small part of the computation of another function
- Allow us to give descriptive names to bits of computation!
- We can reuse a helper function if we need to do the same computation more than once!

Compare the two functions. Are they equivalent?

```
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))
```