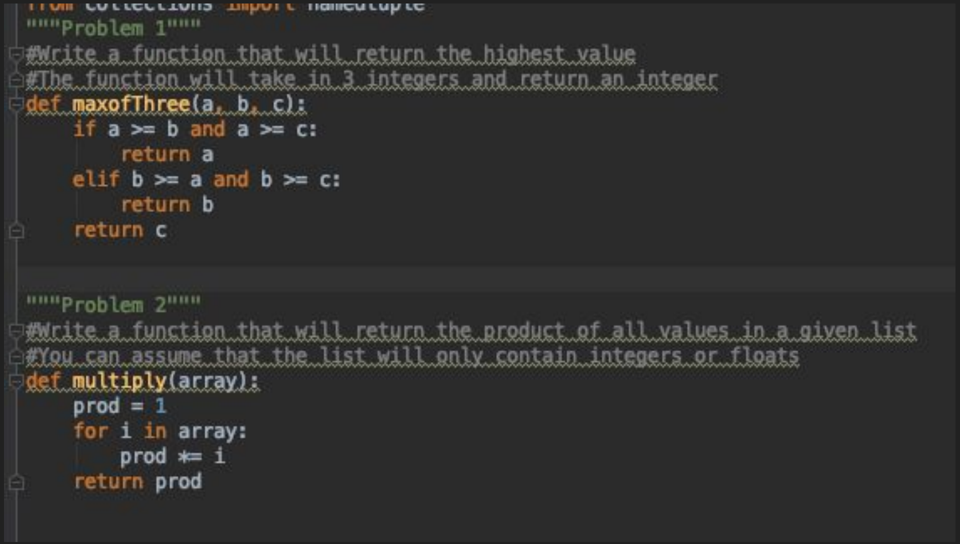
Lecture 6

Functions & Debugging

Announcements

1. There will be a practice midterm on **Tuesday** where you will be given 1 hr to complete it
2. Midterm should be easier than the homework assignments
3. Midterm review assignment will be up tonight

Assignment 6 Solutions



```python
from collections import namedtuple
"""Problem 1"""
#Write a function that will return the highest value
#The function will take in 3 integers and return an integer
def maxofThree(a, b, c):
    if a >= b and a >= c:
        return a
    elif b >= a and b >= c:
        return b
    return c


"""Problem 2"""
#Write a function that will return the product of all values in a given list
#You can assume that the list will only contain integers or floats
def multiply(array):
    prod = 1
    for i in array:
        prod *= i
    return prod
```

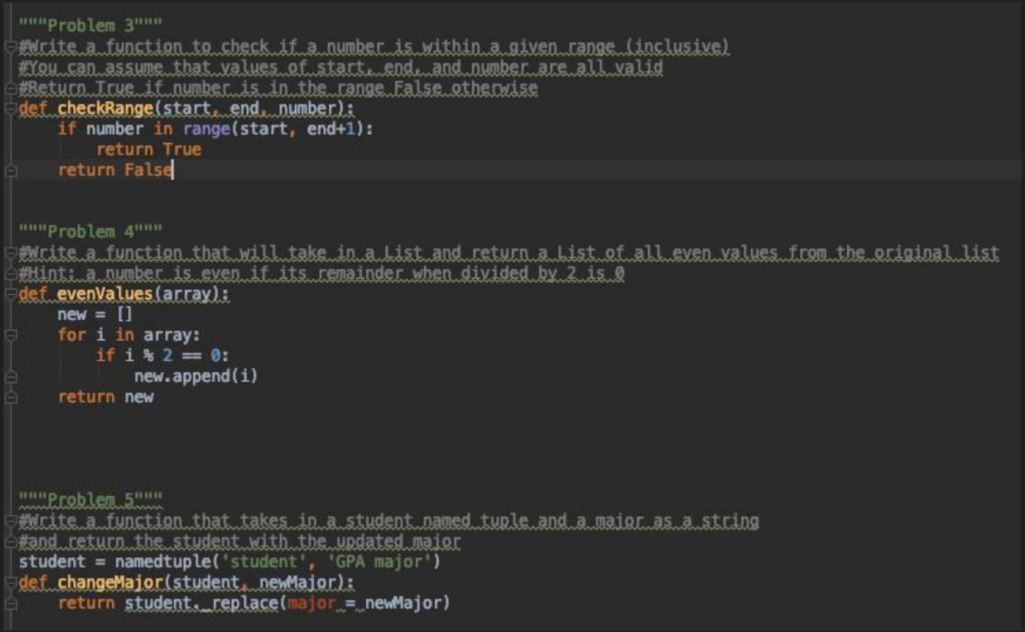
Problem 1:

Check if a is the greatest, check if the b is the greatest, its c otherwise

Problem 2:

Loop through the array while keeping track of the current product at each iteration

Assignment 6 Solutions



```
"""Problem 3"""
#Write a function to check if a number is within a given range (inclusive)
#You can assume that values of start, end, and number are all valid
#Return True if number is in the range False otherwise
def checkRange(start, end, number):
    if number in range(start, end+1):
        return True
    return False


"""Problem 4"""
#Write a function that will take in a List and return a List of all even values from the original list
#Hint: a number is even if its remainder when divided by 2 is 0
def evenValues(array):
    new = []
    for i in array:
        if i % 2 == 0:
            new.append(i)
    return new



"""Problem 5"""
#Write a function that takes in a student named tuple and a major as a string
#and return the student with the updated major
student = namedtuple('student', 'GPA major')
def changeMajor(student, newMajor):
    return student._replace(major = newMajor)
```

Problem 3

Checks if the number exists within a given range

Problem 4

Loops through the array and if the remainder is 0 then its even

Problem 5

Replace the major with the new one

Do it yourself: Function Practice #1

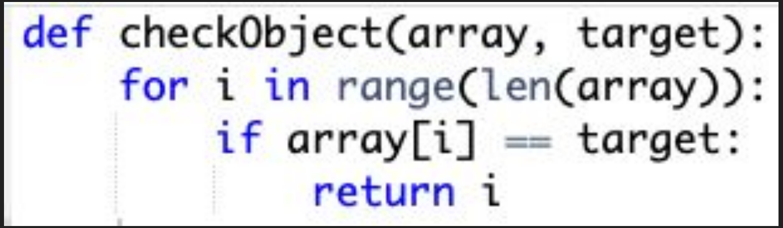
Write a function checkObject() which takes in a list and an object and returns the position/index of the object in the list if the object exists and None if the object does not exist in the list.

Solution:

Do it yourself: Function Practice #1 - Solution

Write a function checkObject() which takes in a list and an object and returns the position/index of the object in the list if the object exists and None if the object does not exist in the list.

Solution:



```python
def checkObject(array, target):
    for i in range(len(array)):
        if array[i] == target:
            return i
```

Do it yourself: Function Practice #2

Write a function stepsToZero() which takes in an integer and returns the number of steps it will take to reduce the integer to 0. If the integer is even, divide it by 2, if the integer is odd, subtract it by 1:

Input: num = 8

Output: 4

Explanation:

Step 1) 8 is even; divide by 2 and obtain 4.

Step 2) 4 is even; divide by 2 and obtain 2.

Step 3) 2 is even; divide by 2 and obtain 1.

Step 4) 1 is odd; subtract 1 and obtain 0.

Solution:

Do it yourself: Function Practice #2 - Solution

Write a function stepsToZero() which takes in an integer and returns the number of steps it will take to reduce the integer to 0. If the integer is even, divide it by 2, if the integer is odd, subtract it by 1:

Input: num = 8

Output: 4

Explanation:

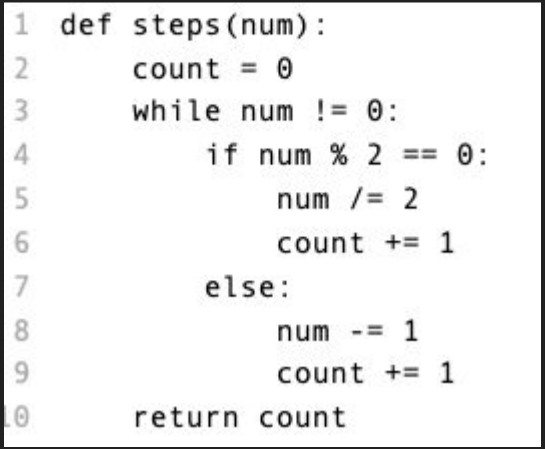
Step 1) 8 is even; divide by 2 and obtain 4.

Step 2) 4 is even; divide by 2 and obtain 2.

Step 3) 2 is even; divide by 2 and obtain 1.

Step 4) 1 is odd; subtract 1 and obtain 0.

Solution:



```python
def steps(num):
    count = 0
    while num != 0:
        if num % 2 == 0:
            num /= 2
            count += 1
        else:
            num -= 1
            count += 1
    return count
```

Do it yourself: Function Practice #3

Make a function runningSum() which takes in an array and returns an array of the running sum.

Input: nums = [1,2,3,4]

Output: [1,3,6,10]

Explanation: Running sum is obtained as follows:

[1, 1+2, 1+2+3, 1+2+3+4].

Solution:

Do it yourself: Function Practice #3 - Solution

Make a function runningSum() which takes in an array and returns an array of the running sum.

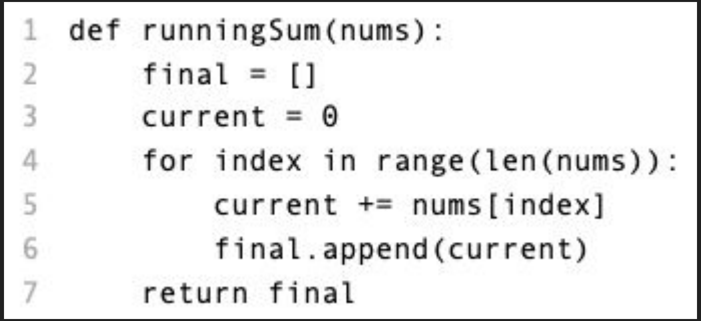
Input: nums = [1,2,3,4]

Output: [1,3,6,10]

Explanation: Running sum is obtained as follows:

[1, 1+2, 1+2+3, 1+2+3+4]

Solution:



```python
def runningSum(nums):
    final = []
    current = 0
    for index in range(len(nums)):
        current += nums[index]
        final.append(current)
    return final
```

Do it yourself: Function Practice #4

Write a function subtractProandSum() which takes in an integer and returns the difference between the product and sum of it’s digits

Input: n = 234

Output: 15

Explanation:

Product of digits = 2 * 3 * 4 = 24

Sum of digits = 2 + 3 + 4 = 9

Result = 24 - 9 = 15

Solution:

Do it yourself: Function Practice #4

Write a function subtractProandSum() which takes in an integer and returns the difference between the product and sum of it's digits

Input: n = 234

Output: 15

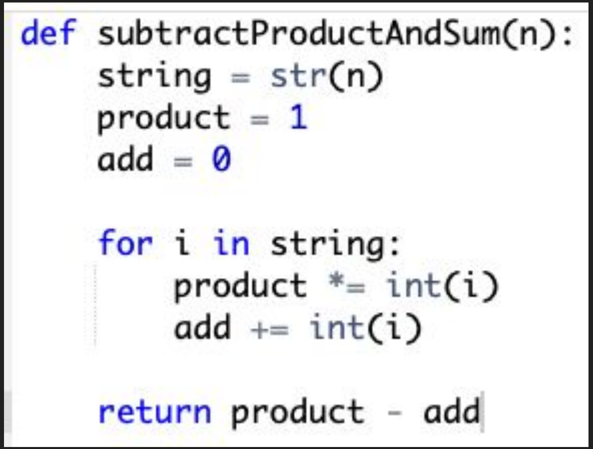
Explanation:

Product of digits = 2 * 3 * 4 = 24

Sum of digits = 2 + 3 + 4 = 9

Result = 24 - 9 = 15

Solution:



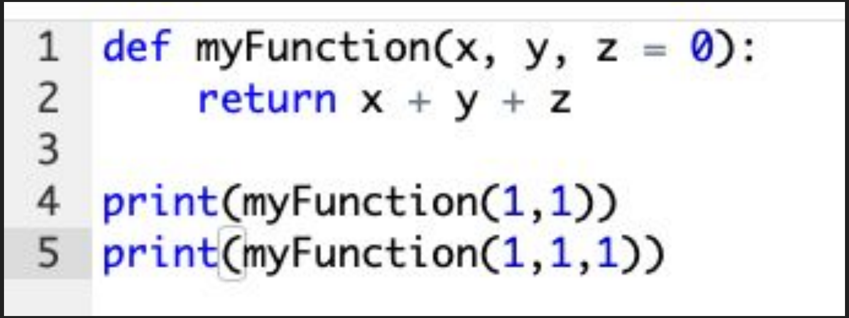
```
def subtractProductAndSum(n):
    string = str(n)
    product = 1
    add = 0

    for i in string:
        product *= int(i)
        add += int(i)

    return product - add
```

Setting Default Parameter Values

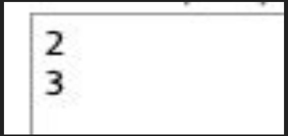
This isn't used very often but it's just something that you should know is possible to do.

If a function has a default value(s) for any of its parameters then it will take on those values if it is not given in the function call.

To assign a default value you would put = default value after a parameter.
Note: parameters with default values have to go last

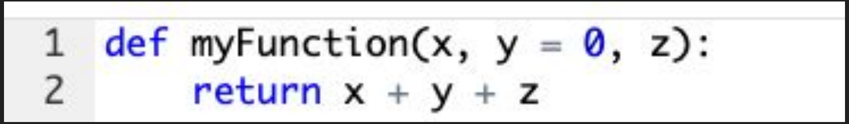


```python
def myFunction(x, y, z = 0):
    return x + y + z

print(myFunction(1,1))
print(myFunction(1,1,1))
```



```
2
3
```

Note: this wont work, parameters with default values have to go last



```python
def myFunction(x, y = 0, z):
    return x + y + z
```

Testing Our Program

In my experience I have easily spent 10x more time debugging and testing my program to make sure it works for all cases than I have coding it.

Most of the time the code that you write won’t work perfectly the first time you run it.

We need to learn skills to test our code and build the habit of constantly thinking about all possible cases when we are writing our programs to help reduce our time debugging and running into issues.

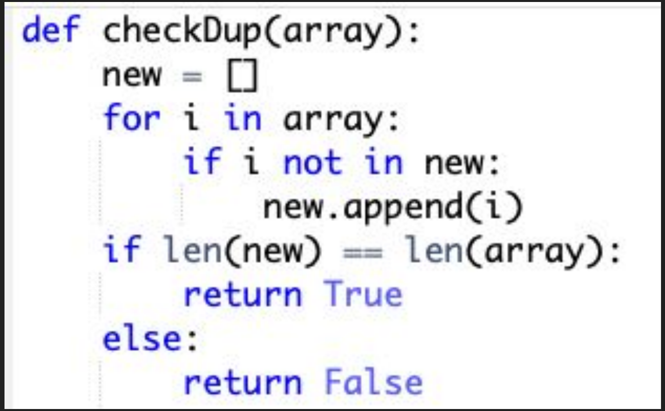
How do I come up with tests?

There is no right or direct approach to this. As you are most likely aware by now programming is very abstract and conceptual so there is no one right way or procedure to take to get the answer.

So how do we come up with proper test cases for our program? It depends and we can only learn to see and do so through practice.

Making Test Cases - Example #1

Write a function checkDup() that takes in a list and returns True if the list has no duplicates and False if it does.

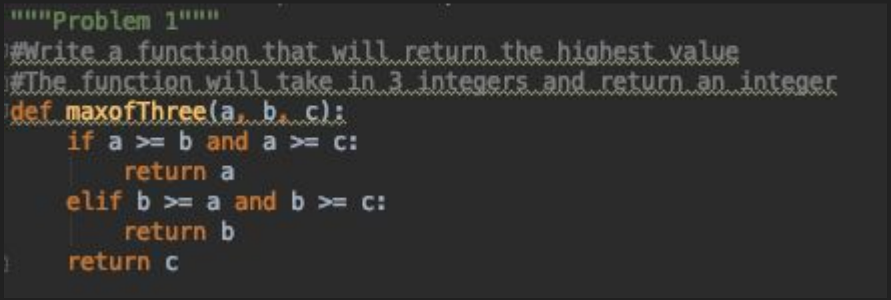


```
def checkDup(array):
    new = []
    for i in array:
        if i not in new:
            new.append(i)
    if len(new) == len(array):
        return True
    else:
        return False
```

Since we are checking if an array has duplicates there are only two possible cases that we need to cover

1. There are duplicates
2. There are no duplicates

Making Test Cases - Example #2



```
"""Problem 1"""
#Write a function that will return the highest value
#The function will take in 3 integers and return an integer
def maxofThree(a, b, c):
    if a >= b and a >= c:
        return a
    elif b >= a and b >= c:
        return b
    return c
```

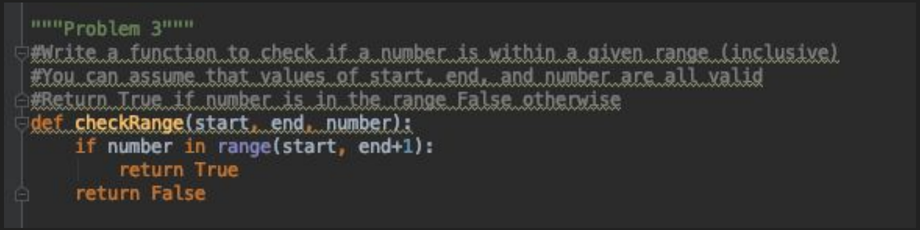
In this scenario we have three possible cases right off the bat.

1. A is the greatest
2. B is the greatest
3. C is the greatest

Other possible test cases that we could consider but we can also consider the cases

A < b < c or A < c < b or B < a < c or B < c < a

Try it yourself - Test Case #1

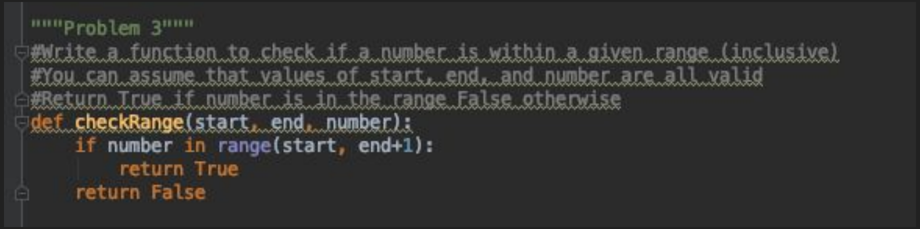


```
"""Problem 3"""
#Write a function to check if a number is within a given range (inclusive)
#You can assume that values of start, end, and number are all valid
#Return True if number is in the range False otherwise
def checkRange(start, end, number):
    if number in range(start, end+1):
        return True
    return False
```

What are the possible test cases that you would want to cover for this function?

Solution:

Try it yourself - Test Case #1 - Solution



```
"""Problem 3"""
#Write a function to check if a number is within a given range (inclusive)
#You can assume that values of start, end, and number are all valid
#Return True if number is in the range False otherwise
def checkRange(start, end, number):
    if number in range(start, end+1):
        return True
    return False
```

What are the possible test cases that you would want to cover for this function?

Solution:

(Ignore the fact that I said we can assume the values of start, end, and number are valid)

1. Start < number < end
2. Start < end < number
3. End < Start < number
4. End < number < start
5. Number < start < end
6. Number < end < start

Try it yourself - Test Case #2

I want to make a function that allows a user to create an account with a username and password.

The username can be anything but the password must have at least 8 characters, 1 number, 1 letter.

What do possible test cases do we want to make?

Solution:

Try it yourself - Test Case #2 - Solution

I want to make a function that allows a user to create an account with a username and password.

The username can be anything but the password must have at least 8 characters, 1 number and 1 letter.

What do possible test cases do we want to make?

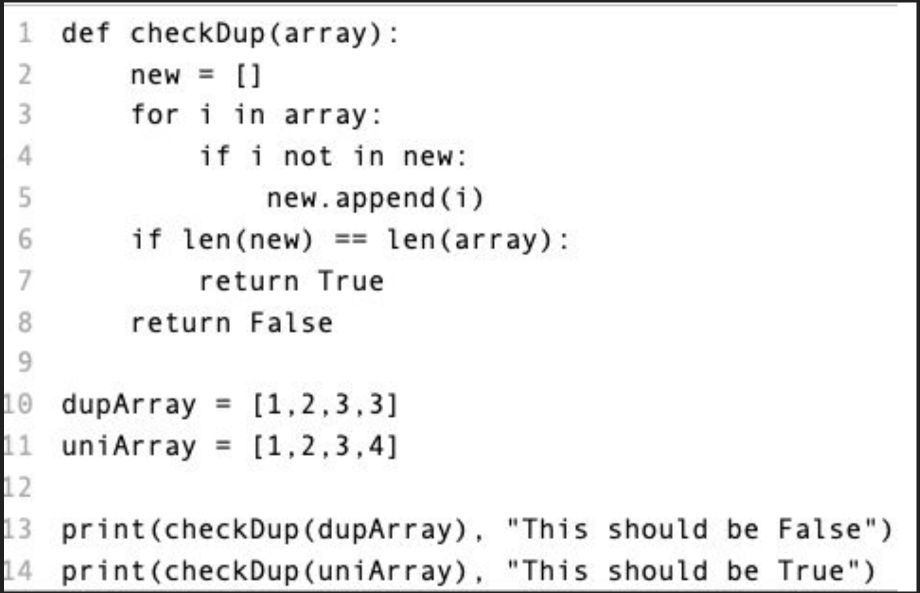
Solution:

1. 8 character long string with numbers and letters
2. 8 character long string with only numbers
3. 8 character long string with only letters
4. 7 character long string
5. 9 character long string
6. 0 character long string
7. 20 character long string

Checking our cases with print statements

This is the easiest way and it’s how I do my test cases when I am trying to test just a couple of functions and cases.

This isn’t good for large programs and more comprehensive tests as they can be harder to track through your program and take longer to type multiple times



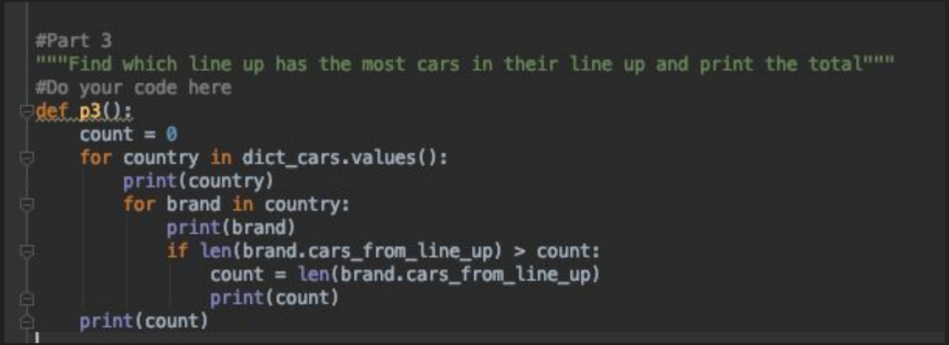
```
def checkDup(array):
    new = []
    for i in array:
        if i not in new:
            new.append(i)
    if len(new) == len(array):
        return True
    return False

dupArray = [1,2,3,3]
uniArray = [1,2,3,4]

print(checkDup(dupArray), "This should be False")
print(checkDup(uniArray), "This should be True")
```

Using print to debug (print debugging)

Print statements could be used anywhere in your functions and program to help you keep track of what your program is doing to your variables at any given time.

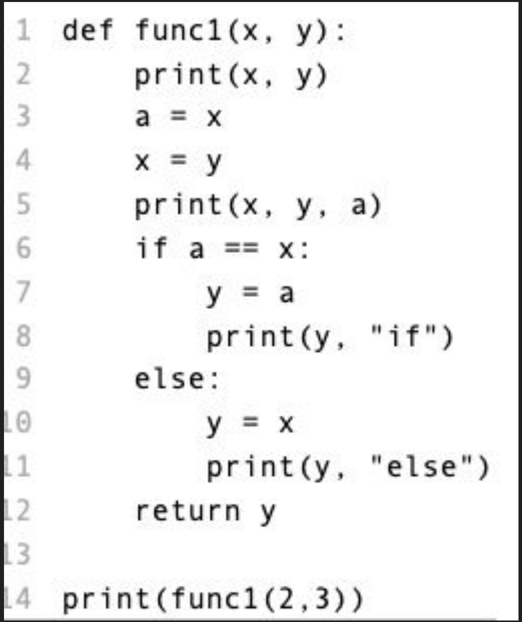
This type of debugging is very powerful as you can get a lot of specific information to find small bugs that could lead to a larger issue.

Since print statements don't end the program you can use them freely to track everything from start to finish.

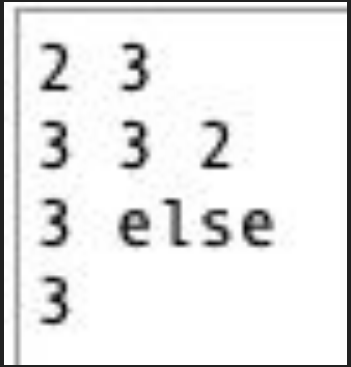


```python
#Part 3
"""Find which line up has the most cars in their line up and print the total"""
#Do your code here
def p3():
    count = 0
    for country in dict_cars.values():
        print(country)
        for brand in country:
            print(brand)
            if len(brand.cars_from_line_up) > count:
                count = len(brand.cars_from_line_up)
                print(count)
    print(count)
```

Print Debug in practice



```
1  def func1(x, y):
2      print(x, y)
3      a = x
4      x = y
5      print(x, y, a)
6      if a == x:
7          y = a
8          print(y, "if")
9      else:
10         y = x
11         print(y, "else")
12     return y
13
14 print(func1(2,3))
```



```
2 3
3 3 2
3 else
3
```

By using print debugging we were able to see that x and y were 2 and 3 respectively.

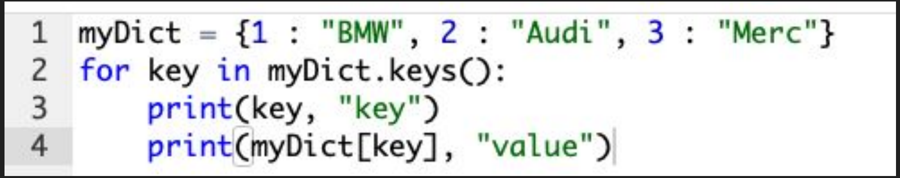
At like 5 x was changed to 3.

Then at line 11, 3 else was printed

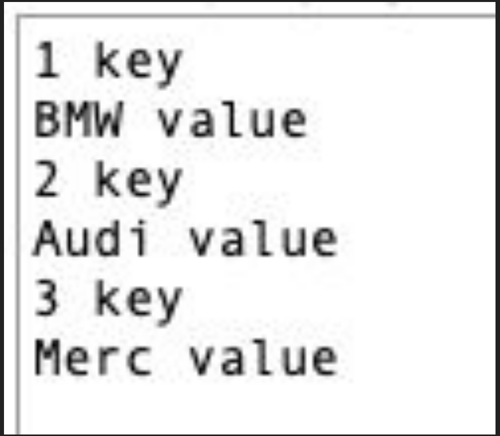
Then line 14 ran and printed the returned value of 3.

Notice how the print statement on line 8 never printed. From this we can conclude that that if conditional statement was never met.

Print Debug in practice



```python
myDict = {1 : "BMW", 2 : "Audi", 3 : "Merc"}
for key in myDict.keys():
    print(key, "key")
    print(myDict[key], "value")
```



```
1 key
BMW value
2 key
Audi value
3 key
Merc value
```

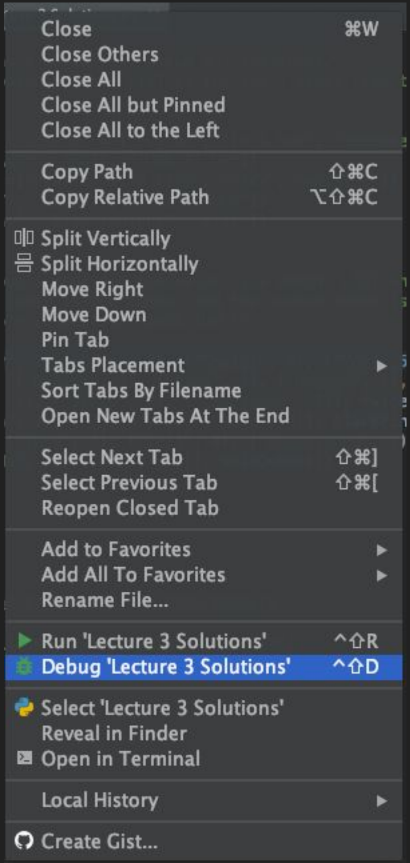
If you had tried assignment 4 then you must have noticed this already but print debugging can be really useful for helping you keep track of what value you are getting from a data structure at each iteration.

Built in Debugger

Within every interpreter there is a built in debugger which is very powerful. Most of the time you won’t have to resort to using this cause print debugging when used properly can be just as effective but.

When trying to debug more complicated code the debugger is more “fool proof” as it shows you everything regardless.

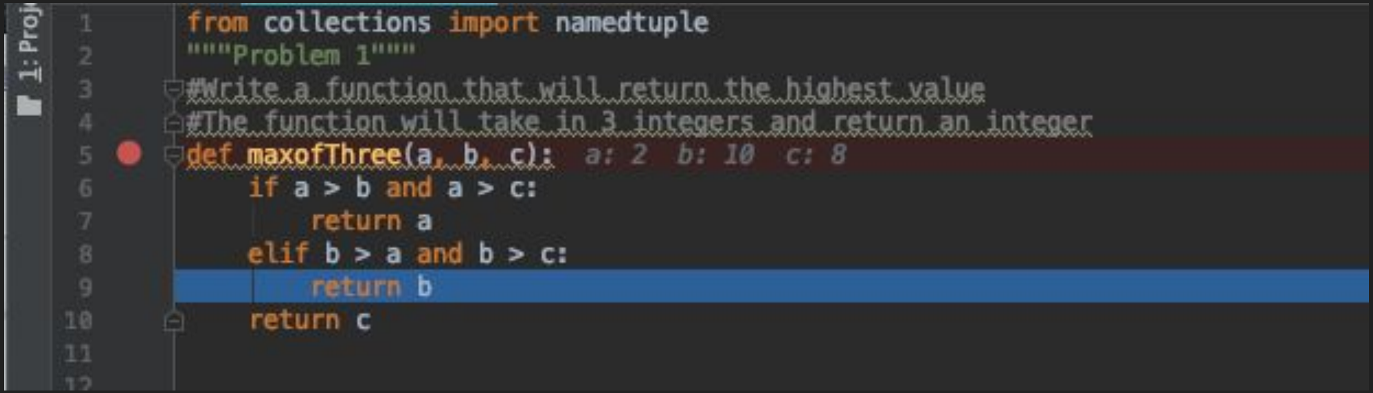
In my experience I don’t really use the built in debugger as print statements can be very strong when used properly and that is something you learn from experience.



How to use the Debugger

Every interpreter is different but it should be the same process and idea.

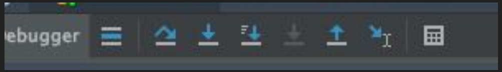
1. Set a breakpoint by clicking in the margin at the line where you want the program to stop running or where you want to start "debugging"



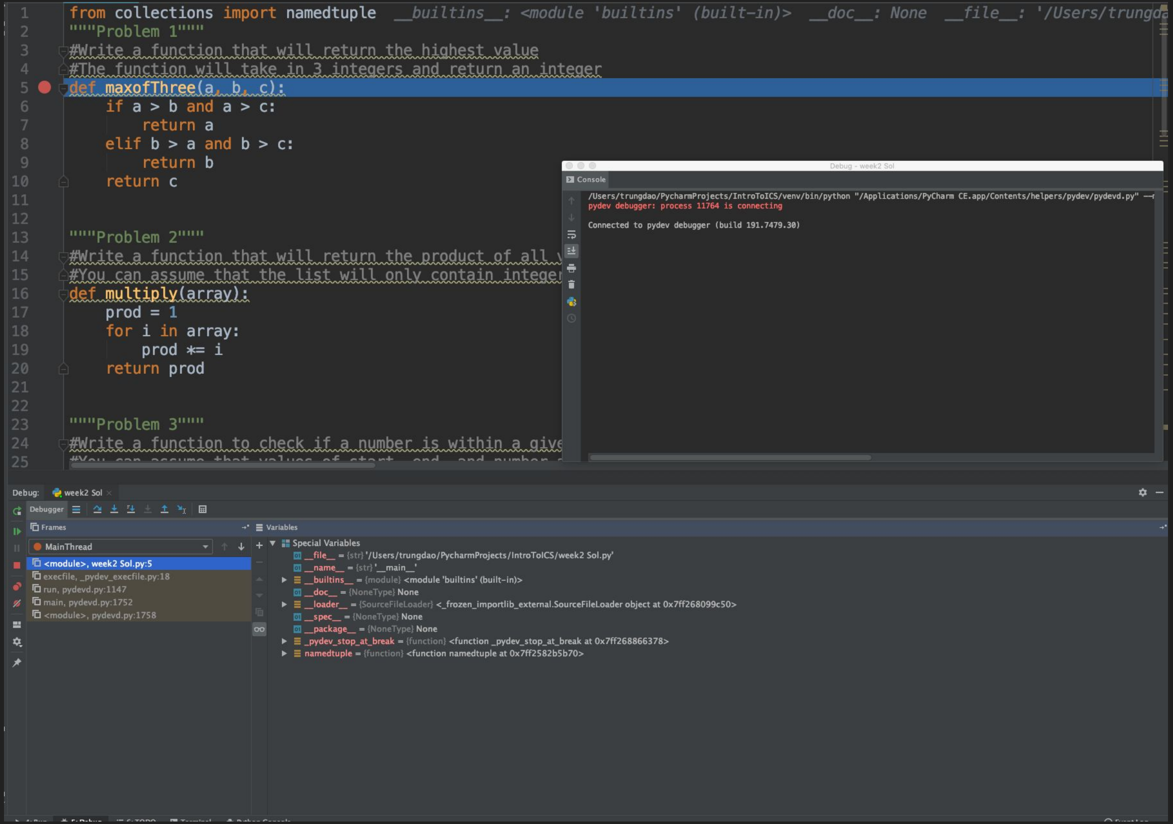
```
from collections import namedtuple
"""Problem 1"""
#Write a function that will return the highest value
#The function will take in 3 integers and return an integer
def maxofThree(a, b, c):  a: 2  b: 10  c: 8
    if a > b and a > c:
        return a
    elif b > a and b > c:
        return b
    return c
```

How to use the Debugger

Click on the run debug instead of regular run and something like this should pop up.



Now you can use these buttons to move through the program line by line and see how each line of code affects the variables



```
from collections import namedtuple   __builtins__: <module 'builtins' (built-in)>   __doc__: None   __file__: '/Users/trungda
"""Problem 1"""
#Write a function that will return the highest value
#The function will take in 3 integers and return an integer
def maxofThree(a, b, c):
    if a > b and a > c:
        return a
    elif b > a and b > c:
        return b
    return c


"""Problem 2"""
#Write a function that will return the product of all v
#You can assume that the list will only contain integer
def multiply(array):
    prod = 1
    for i in array:
        prod *= i
    return prod


"""Problem 3"""
#Write a function to check if a number is within a give
```

Debug - week2 Sol

Console

```
/Users/trungdao/PycharmProjects/IntroToICS/venv/bin/python "/Applications/PyCharm CE.app/Contents/helpers/pydev/pydevd.py" --
pydev debugger: process 11764 is connecting

Connected to pydev debugger (build 191.7479.30)
```

Debug: week2 Sol

```
<module>, week2 Sol.py:5
execfile, _pydev_execfile.py:18
run, pydevd.py:1147
main, pydevd.py:1752
<module>, pydevd.py:1758
```

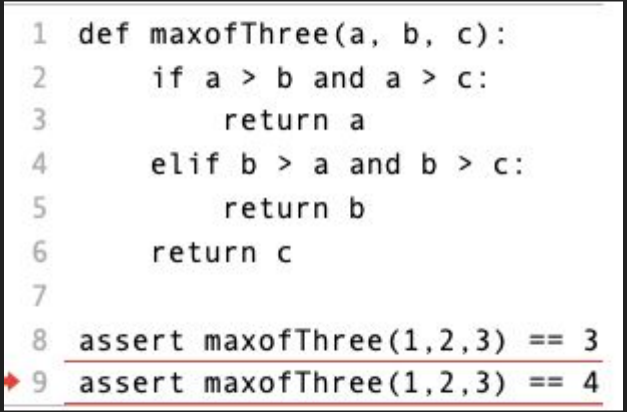
```
Special Variables
  __file__ = {str} '/Users/trungdao/PycharmProjects/IntroToICS/week2 Sol.py'
  __name__ = {str} '__main__'
▶ __builtins__ = {module} <module 'builtins' (built-in)>
  __doc__ = {NoneType} None
▶ __loader__ = {SourceFileLoader} <_frozen_importlib_external.SourceFileLoader object at 0x7ff268099c50>
  __spec__ = {NoneType} None
  __package__ = {NoneType} None
▶ _pydev_stop_at_break = {function} <function _pydev_stop_at_break at 0x7ff268866378>
▶ namedtuple = {function} <function namedtuple at 0x7ff2582b5b70>
```

Assert

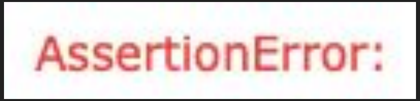
The assert statement is used for testing and code.

This allows you to test a conditional and does nothing if it’s True and will raise an AssertionError otherwise.

I never use these cause when it raises an AssertionError the program stops and I am not able to check what happens after



```
def maxofThree(a, b, c):
    if a > b and a > c:
        return a
    elif b > a and b > c:
        return b
    return c

assert maxofThree(1,2,3) == 3
assert maxofThree(1,2,3) == 4
```



```
AssertionError:
```