

Session 4 Handout Solutions

0. Debugging Illustration (Case 6a from last session)

```
[18]: import math
      def numberMonths(total,monthly,interest=0.0425,downpay=0):
          T=total
          M=monthly
          I=interest
          D=downpay
          return math.ceil(-math.log(1-I(T-D)/M)/math.log(1+I))
      numberMonths(500000,4000)/12
```

```
-----

TypeError                                Traceback (most recent call last)

<ipython-input-18-41abbd54549f> in <module>()
      6     D=downpay
      7     return math.ceil(-math.log(1-I(T-D)/M)/math.log(1+I))
----> 8 numberMonths(500000,4000)/12

<ipython-input-18-41abbd54549f> in numberMonths(total, monthly, interest, downpay)
      5     I=interest
      6     D=downpay
----> 7     return math.ceil(-math.log(1-I(T-D)/M)/math.log(1+I))
      8 numberMonths(500000,4000)/12

TypeError: 'float' object is not callable
```

A. Recreating the error outside of the function

```
[19]: T=500000
      M=4000
      I=0.0425
      D=0
      math.ceil(-math.log(1-I(T-D)/M)/math.log(1+I))
```

```
-----

TypeError                                Traceback (most recent call last)

<ipython-input-19-d77348fe2d7e> in <module>()
      3 I=0.0425
      4 D=0
----> 5 math.ceil(-math.log(1-I(T-D)/M)/math.log(1+I))
```

```
TypeError: 'float' object is not callable
```

B. Dissecting the line containing the error

```
[20]: T-D
```

```
500000
```

```
[21]: I(T-D)
```

```
-----  
TypeError                                Traceback (most recent call last)  
  
  <ipython-input-21-1a7a9b0b7ec8> in <module>()  
----> 1 I(T-D)
```

```
TypeError: 'float' object is not callable
```

```
[22]: I*(T-D)/M
```

```
5.3125
```

```
[23]: 1-I*(T-D)/M
```

```
-4.3125
```

```
[24]: math.log(-4.3125)
```

```
-----  
ValueError                                Traceback (most recent call last)  
  
  <ipython-input-24-4e9cf63ad0b5> in <module>()  
----> 1 math.log(-4.3125)
```

```
ValueError: math domain error
```

C. Correcting the logic (this time building up one component at a time)

```
[25]: T=500000
```

```
      M=4000
```

```
      I=0.0425
```

```
      D=0
```

```
      i=I/12
```

```
      A=i*(T-D)/M
```

```

top=-math.log(1-A)
bottom=math.log(1+i)
N=math.ceil(top/bottom)
N

```

166

D. Putting correct logic back into function

```

[26]: import math
def numberMonths(total,monthly,interest=0.0425,downpay=0):
    T=total
    M=monthly
    I=interest
    D=downpay
    i=I/12
    A=i*(T-D)/M
    top=-math.log(1-A)
    bottom=math.log(1+i)
    N=math.ceil(top/bottom)
    return N
numberMonths(500000,4000)/12

```

13.833333333333334

(Optional: Shortening the code to work with original named variables directly.)

```

[27]: import math
def numberMonths(total,monthly,interest=0.0425,downpay=0):
    i=interest/12
    A=i*(total-downpay)/monthly
    top=-math.log(1-A)
    bottom=math.log(1+i)
    return math.ceil(top/bottom)
numberMonths(500000,4000)/12

```

13.833333333333334

1. Using while loops

The following code assumes that you have a correct orderQuantity function from last session in a session3.py file in the current directory.

```

[1]: from session3 import orderQuantity
while True:
    userInput=input('Enter inventory (or done): ')
    if userInput=='done':
        break
    elif userInput=='skip':
        continue
    inventory=int(userInput)
    print ('Order',orderQuantity(inventory),'units.')

```

```

Enter inventory (or done): skip
Enter inventory (or done): 30
Order 70 units.
Enter inventory (or done): 25
Order 75 units.
Enter inventory (or done): done

```

Alternative implementation without using break or continue.

```

[ ]: ## from session3 import orderQuantity
    userInput=input('Enter inventory (or done): ')
    while userInput!='done':
        if userInput!='skip':
            inventory=int(userInput)
            print ('Order',orderQuantity(inventory),'units.')
            userInput=input('Enter inventory (or done): ')

```

Q1: Write a program to repeatedly ask the user to input the number of hours worked, and display the total pay, assuming that the rate for first 40 hours is 10/hour, and the rate for additional hours is 15/hour. The program should terminate whenever the user inputs done.

```

[2]: from session3 import calculateWage
    while True:
        userInput=input('Enter hours worked (or done): ')
        if userInput=='done':
            break
        hours=float(userInput)
        print('Pay is',calculateWage(hours))

```

```

Enter hours worked (or done): 38
Pay is 380.0
Enter hours worked (or done): 42
Pay is 430.0
Enter hours worked (or done): done

```

(optional) Q2: Rewrite the code in Q1 but do not use break.

```

[ ]: from session3 import calculateWage
    prompt='Enter hours worked (or done): '
    userInput=input(prompt)

    while userInput!='done':
        hours=float(userInput)
        print('Pay is',calculateWage(hours))
        userInput=input(prompt)

```

The following function uses try and except (see PY4E Chapter 3) for checking whether a certain value is convertible to a float.

```

[3]: def isNumber(x):
    try:

```

```

        float(x)
        return True
    except:
        return False

print(isNumber(3))
print(isNumber('3'))
print(isNumber('three'))

```

True
True
False

Q3: Modify the first example of this handout so that if the user does not input done nor an integer, then the program prints Invalid input. and asks for another input. (Hint: first write an isInteger(x) function by modifying the above, then use an if statement to decide whether to convert the input to an integer, or display Invalid input.)

```

[4]: from session3 import orderQuantity
def isInteger(x):
    try:
        if int(x)==float(x):
            return True
        else:
            return False
    except:
        return False

while True:
    userInput=input('Enter inventory (or done): ')
    if userInput=='done':
        break
    if isInteger(userInput):
        inventory=int(userInput)
        print ('Order',orderQuantity(inventory),'units.')
    else:
        print('Invalid input.')

```

```

Enter inventory (or done): 2.5
Invalid input.
Enter inventory (or done): 30
Order 70 units.
Enter inventory (or done): thirty
Invalid input.
Enter inventory (or done): done

```

2. Using for loops

```

[5]: for i in [0,3,5,2]:
    print(i,end=' ')

```

0 3 5 2

```
[6]: for i in range(5):  
      print(i,end=' ')
```

0 1 2 3 4

Q4: Modify the first example of the handout to use a for loop instead of a while loop, and limit the number of iterations to at most 5.

The following example illustrates reading and writing to a file using for loops. The generated file will be used as an input to case 8c.

2.1 Using for loops to read files

Type the following code example in your Jupyter notebook, as it will create a data file that we will use later.

```
[7]: import random  
      file=open('session4_data.txt','w')  
      random.seed(0)  
      for t in range(10):  
          value=random.randint(10,60)  
          print(value,file=file)  
      file.close()  
  
[8]: file=open('session4_data.txt','r')  
      for line in file:  
          print(int(line),end=' ')  
      file.close()
```

34 58 36 12 26 42 41 35 60 29

2.2 Computations Using Loops

```
[9]: l=[0,3,5,2,-2]  
      total=0  
      largest=-1e100  
      smallest=1e100  
      for num in l:  
          total=total+num  
          if num>largest:  
              largest=num  
          if num<smallest:  
              smallest=num  
      print('Total:',total)  
      print('Average:',total/len(l))  
      print('Largest:',largest)  
      print('Smallest:',smallest)
```

Total: 8
Average: 1.6
Largest: 5
Smallest: -2

Q5: Modify the above code to apply to the numbers in the file `session4_data.txt` created by the previous code example.

```
[1]: file=open('session4_data.txt','r')
    total=0
    largest=-1e100
    smallest=1e100
    count=0
    for line in file:
        num=float(line)
        total=total+num
        count+=1
        if num>largest:
            largest=num
        if num<smallest:
            smallest=num
    print('Total:',total)
    print('Average:',total/count)
    print('Largest:',largest)
    print('Smallest:',smallest)
```

```
Total: 373.0
Average: 37.3
Largest: 60.0
Smallest: 12.0
```