Session 3 Handout (Solutions Only)

Q1: (Modification of case 2 from last session) Write a function named `orderQuantity` that takes two input arguments, `inventory` and `basestock`. If `inventory` is at least equal to `basestock`, then return 0. Otherwise, return the difference between `basestock` and `inventory`. Set the default value for `inventory` to be 0 and for `basestock` to be 100. Include an appropriate docstring to explain what the function does.

```
[2]: def orderQuantity(inventory=0,basestock=100):
         ''' Calculates order quantity given inventory level and basestock level'''
         if inventory>=basestock:
             return 0
         else:
             return basestock-inventory
```

```
[3]: # Code to test your function
     help(orderQuantity)
     print(orderQuantity())
     print(orderQuantity(25))
     print(orderQuantity(51,50))
     print(orderQuantity(basestock=200))
     print(orderQuantity(inventory=80))
```

```
Help on function orderQuantity in module __main__:

orderQuantity(inventory=0, basestock=100)
    Calculates order quantity given inventory level and basestock level

100
75
0
200
20
```

Q2: Walk through the code to explain each line of the above output.

Q3: Run the above line and out of the items in all lowercase, choose five that look interesting to you, and use `type` and `help` and trial and error to find out what each of these built-in objects are and what you can do with them. Explain to your neighbor.

```
[9]: help(abs)
```

```
Help on built-in function abs in module builtins:

abs(x, /)
    Return the absolute value of the argument.
```

```
[10]: help(max)
```

```
Help on built-in function max in module builtins:
```

```
max(...)
    max(iterable, *[, default=obj, key=func]) -> value
    max(arg1, arg2, *args, *[, key=func]) -> value

    With a single iterable argument, return its biggest item. The
    default keyword-only argument specifies an object to return if
    the provided iterable is empty.
    With two or more arguments, return the largest argument.
```

```
[11]: help(min)
```

```
Help on built-in function min in module builtins:

min(...)
    min(iterable, *[, default=obj, key=func]) -> value
    min(arg1, arg2, *args, *[, key=func]) -> value

    With a single iterable argument, return its smallest item. The
    default keyword-only argument specifies an object to return if
    the provided iterable is empty.
    With two or more arguments, return the smallest argument.
```

```
[12]: help(sum)
```

```
Help on built-in function sum in module builtins:

sum(iterable, start=0, /)
    Return the sum of a 'start' value (default: 0) plus an iterable of numbers

    When the iterable is empty, return the start value.
    This function is intended specifically for use with numeric values and may
    reject non-numeric types.
```

Q4: Import the `math` module and print the list of variables and functions within this module using `dir`. Choose five functions from this list and use `help` and trial and error to figure out how to use them. Explain to your neighbor.

```
[13]: import math
      print(dir(math))
```

```
['__doc__', '__loader__', '__name__', '__package__', '__spec__', 'acos', 'acosh', 'asin', 'as
```

```
[14]: help(math.cos)
```

```
Help on built-in function cos in module math:
```

```
cos(...)
    cos(x)

    Return the cosine of x (measured in radians).
```

```
[15]: help(math.log)
```

```
Help on built-in function log in module math:

log(...)
    log(x[, base])

    Return the logarithm of x to the given base.
    If the base not specified, returns the natural logarithm (base e) of x.
```

Q5: Use `dir` on the string object `"Hi"`. Choose five functions from this list and use `help` and trial and error to figure out how to use these functions built in to every string object. Explain to your neighbor.

```
[16]: print(dir('Hi'))
```

```
['__add__', '__class__', '__contains__', '__delattr__', '__dir__', '__doc__', '__eq__', '__fo
```

```
[17]: help('Hi'.lower)
```

```
Help on built-in function lower:

lower(...) method of builtins.str instance
    S.lower() -> str

    Return a copy of the string S converted to lowercase.
```

```
[18]: help(str.lower)
```

```
Help on method_descriptor:

lower(...)
    S.lower() -> str

    Return a copy of the string S converted to lowercase.
```

```
[19]: help(str.find)
```

```
Help on method_descriptor:

find(...)
    S.find(sub[, start[, end]]) -> int

    Return the lowest index in S where substring sub is found,
    such that sub is contained within S[start:end].  Optional
    arguments start and end are interpreted as in slice notation.

    Return -1 on failure.
```

Case 6a. Mortgage Calculator I

Write a function `numberMonths` in module `session3` that calculates how many months it would take to pay off a mortgage given the monthly payment. The function has four input arguments: `total`, `monthly`, `interest`, and `downpay`. Let the default values for `interest` be 0.0425 and for `downpay` be 0. Label the four arguments T, M, I, D respectively. The number of months needed N is given by the formula

$$N = ceil\left(\frac{-\log(1-\frac{i(T-D)}{M})}{\log(1+i)}\right),$$

where $i = I/12$ is the monthly interest rate and *ceil* is the `math.ceil` function. (Note, after modifying the `session3.py`, you will have to restart the kernel using the toolbar above to reload the latest version.)

```
[20]: import session3 as s3
      s3.numberMonths(500000,4000)/12
```

```
13.833333333333334
```

```
[21]: s3.numberMonths(500000,4000,interest=0.05)/12
```

```
14.75
```

Case 6b. Mortgage Calculator II

Write a function `monthlyPayment` in module `session3` that calculates the monthly payment needed to pay off a mortgage in a given number of months. The function has four input arguments: `total`, `months`, `interest`, and `downpay`. Let the default values for `interest` be 0.0425 and for `downpay` be 0. Label the four arguments T, N, I, D respectively. The monthly payment M is given by the formula

$$M = \frac{(1+i)^N}{(1+i)^N - 1} i(T-D),$$

where $i = I/12$ is the monthly interest rate. Round the answer to two decimal places using the `round` function.

```
[22]: s3.monthlyPayment(500000,12*30)
```

```
2459.7
```

```
[23]: s3.monthlyPayment(500000,12*30,interest=0.05)
```

```
2684.11
```

The two functions are below.

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

     def monthlyPayment(total,months,interest=0.0425,downpay=0):
         i=interest/12
         return round((1+i)**months*(total-downpay)*i/((1+i)**months-1),2)
```

Note that it is easier to debug if you disected the formula into small chuncks, as below

```
[ ]: 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)

     def monthlyPayment(total,months,interest=0.0425,downpay=0):
         i=interest/12
         top=(1+i)**months*i*(total-downpay)
         bottom=(1+i)**months-1
         return round(top/bottom,2)
```