Session 3 Handout

1. Defining and Calling your Own Functions

```
[1]: def calculateWage(hours,base=10,bonus=.5):
         ''' Calculates weekly wage '''
         if hours<=40:
             pay=hours*base
         else:
             pay=hours*base+(hours-40)*base*bonus
         return pay

     help(calculateWage)
     print('Pay for 42 hours with default base and bonus:',calculateWage(42))
     print('Pay for 42 hours with base 12/hour and default bonus:',calculateWage(42,12))
     print('Pay for 42 hours with base 12/hour and bonus 60%:', calculateWage(42,12,.6))
     print('Pay for 42 hours with default base and bonus 50%:',calculateWage(42,bonus=0.6))
```

```
Help on function calculateWage in module __main__:

calculateWage(hours, base=10, bonus=0.5)
    Calculates weekly wage

Pay for 42 hours with default base and bonus: 430.0
Pay for 42 hours with base 12/hour and default bonus: 516.0
Pay for 42 hours with base 12/hour and bonus 60%: 518.4
Pay for 42 hours with default base and bonus 50%: 432.0
```

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.

Packaging functions within a module

Open Spyder and create a new Python script, and copy paste the two functions `calculateWage` and `orderQuantity` into the script. Save the script into the same folder as this notebook, as `session3.py`. If everything is correct, you should be able to run the following.

```
[4]: import session3
     print(session3.calculateWage(40))
     print(session3.orderQuantity(30))
     help(session3.orderQuantity)
```

```
400
70
Help on function orderQuantity in module session3:

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

```
[5]: print('Module contains the following variables and functions:', dir(session3))
```

```
Module contains the following variables and functions: ['__builtins__', '__cached__', '__doc_
'__file__', '__loader__', '__name__', '__package__', '__spec__',
'calculateWage', 'math', 'monthlyPayment', 'numberMonths', 'orderQuantity']
```

```
[6]: help(session3)
```

```
Help on module session3:

NAME
    session3 - Created on Fri Jan 11 14:52:23 2019

DESCRIPTION
    @author: pengshi

FUNCTIONS
    calculateWage(hours, base=10, bonus=0.5)
        Calculates weekly wage

    monthlyPayment(total, months, interest=0.0425, downpay=0)
    numberMonths(total, monthly, interest=0.0425, downpay=0)

    obtainNumeric(prompt='')
        Elicits user to input a number, and continues to ask if input is invalid.

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

FILE
    c:\users\pengshi\dropbox\teaching\2019-spring-dso599\course material\handouts and notes\s
```

2. Exploring Existing Functions

```
[7]: help(print)
```

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

print(...)
    print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

    Prints the values to a stream, or to sys.stdout by default.
    Optional keyword arguments:
    file:  a file-like object (stream); defaults to the current sys.stdout.
    sep:   string inserted between values, default a space.
    end:   string appended after the last value, default a newline.
    flush: whether to forcibly flush the stream.
```

```
[8]: print(dir(__builtins__))
```

```
['ArithmeticError', 'AssertionError', 'AttributeError', 'BaseException', 'BlockingIOError', '
'BufferError', 'BytesWarning', 'ChildProcessError', 'ConnectionAbortedError', 'ConnectionErro
'ConnectionRefusedError', 'ConnectionResetError', 'DeprecationWarning', 'EOFError', 'Ellipsis
'Exception', 'False', 'FileExistsError', 'FileNotFoundError', 'FloatingPointError', 'FutureWa
'IOError', 'ImportError', 'ImportWarning', 'IndentationError', 'IndexError', 'InterruptedErro
'KeyError', 'KeyboardInterrupt', 'LookupError', 'MemoryError', 'ModuleNotFoundError', 'NameEr
'NotADirectoryError', 'NotImplemented', 'NotImplementedError', 'OSError', 'OverflowError',
'PendingDeprecationWarning', 'PermissionError', 'ProcessLookupError', 'RecursionError', 'Refe
'ResourceWarning', 'RuntimeError', 'RuntimeWarning', 'StopAsyncIteration', 'StopIteration', '
'SyntaxWarning', 'SystemError', 'SystemExit', 'TabError', 'TimeoutError', 'True', 'TypeError'
'UnicodeDecodeError', 'UnicodeEncodeError', 'UnicodeError', 'UnicodeTranslateError', 'Unicode
'ValueError', 'Warning', 'WindowsError', 'ZeroDivisionError', '__IPYTHON__', '__build_class__
'__debug__', '__doc__', '__import__', '__loader__', '__name__', '__package__', '__spec__',
'abs', 'all', 'any', 'ascii', 'bin', 'bool', 'bytearray', 'bytes', 'callable', 'chr', 'classm
'complex', 'copyright', 'credits', 'delattr', 'dict', 'dir', 'display', 'divmod', 'enumerate'
'filter', 'float', 'format', 'frozenset', 'get_ipython', 'getattr', 'globals', 'hasattr', 'ha
'id', 'input', 'int', 'isinstance', 'issubclass', 'iter', 'len', 'license', 'list', 'locals',
'memoryview', 'min', 'next', 'object', 'oct', 'open', 'ord', 'pow', 'print', 'property', 'ran
'round', 'set', 'setattr', 'slice', 'sorted', 'staticmethod', 'str', 'sum', 'super', 'tuple',
```

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',
'asinh', 'atan', 'atan2', 'atanh', 'ceil', 'copysign', 'cos', 'cosh', 'degrees', 'e', 'erf',
'fabs', 'factorial', 'floor', 'fmod', 'frexp', 'fsum', 'gamma', 'gcd', 'hypot', 'inf', 'isclo
'isinf', 'isnan', 'ldexp', 'lgamma', 'log', 'log10', 'log1p', 'log2', 'modf', 'nan', 'pi', 'p
'sinh', 'sqrt', 'tan', 'tanh', 'tau', 'trunc']
```

```
[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__',
'__format__', '__ge__', '__getattribute__', '__getitem__', '__getnewargs__', '__gt__',
'__hash__', '__init__', '__init_subclass__', '__iter__', '__le__', '__len__', '__lt__',
'__mod__', '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__',
'__rmod__', '__rmul__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__',
'capitalize', 'casefold', 'center', 'count', 'encode', 'endswith', 'expandtabs', 'find', 'for
'index', 'isalnum', 'isalpha', 'isdecimal', 'isdigit', 'isidentifier', 'islower', 'isnumeric'
'isspace', 'istitle', 'isupper', 'join', 'ljust', 'lower', 'lstrip', 'maketrans', 'partition'
'rindex', 'rjust', 'rpartition', 'rsplit', 'rstrip', 'split', 'splitlines', 'startswith', 'st
'title', 'translate', 'upper', 'zfill']
```

```
[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

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

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
```

3. Efficiently Reusing Code via Functions

```
[24]: while(True):
          try:
              value=float(input('Please input a non-negative number: '))
              if value>=0:
                  print('User input:', value)
                  break
              else:
                  print('Invalid. Input must be non-negative.')
          except:
              print('Invalid. Input must be a number.')
```

```
Please input a non-negative number: Hi!
Invalid. Input must be a number.
Please input a non-negative number: -1
Invalid. Input must be non-negative.
Please input a non-negative number: 30
User input: 30.0
```

Case 7a: Input Checker v1.0

Write a function `inputNumber` with one argument `prompt` based on the above code logic. For example, the function call `inputNumber('Enter price:')`will continue to ask for a price until the user inputs a non-negative number.

```
[25]: def inputNumber(prompt):
          '''A modification of the input function that ensures inputs are non-negative number
          while(True):
              try:
                  value=float(input(prompt))
                  if value>=0:
                      return value
                  else:
                      print('Invalid. Input must be non-negative.')
              except:
                  print('Invalid. Input must be a number.')
```

```
[26]: a=inputNumber('Enter price: ')
      print('Price is',a)
```

```
Enter price: -1
Invalid. Input must be non-negative.
Enter price: 10
Price is 10.0
```

Case 7b: Input Checker v1.1

Modify your `inputNumber` function to take a second argument `decimalAllowed` with default value `True`. If this value is `True`, then the function behaves exactly as above. Otherwise, the function will only allow integer inputs.

```
[27]: def inputNumber(prompt, decimalAllowed=True):
          '''A modification of the input function that ensures inputs are non-negative number
          If decimalAllowed=False, then the function only allows integer inputs.
          '''
          while(True):
              try:
                  if decimalAllowed:
                      value=float(input(prompt))
                  else:
                      value=int(input(prompt))
                  if value>=0:
                      return value
                  else:
                      print('Invalid. Input must be non-negative.')
              except:
                  if decimalAllowed:
                      print('Invalid. Input must be a number.')
                  else:
                      print('Invalid. Input must be an integer.')
```

```
[28]: p=inputNumber('Enter price: ')
      q=inputNumber('Enter number of items: ',decimalAllowed=False)
      print('Revenue is',p*q,'dollars.')
```

```
Enter price: 12.99
Enter number of items: 12.99
Invalid. Input must be an integer.
Enter number of items: 5
Revenue is 64.95 dollars.
```

Q6: Modify your code for Cases 2 and 3 from last class to use the function `inputNumber` for input check, and the functions `calculateWage` and `orderQuantity` for calculations. (For case 2, your program can simply say `Order 0 units` instead of `Sufficient inventory. No need to order.`)

```
[29]: inventory=inputNumber('Current inventory: ',decimalAllowed=False)
      print('Order',orderQuantity(inventory),'units.')
```

```
Current inventory: 2.5
Invalid. Input must be an integer.
Current inventory: 25
Order 75 units.
```

```
[30]: hours=inputNumber('Hours worked this week: ')
      print('Total pay this week is',calculateWage(hours))
```

```
Hours worked this week: -1
Invalid. Input must be non-negative.
Hours worked this week: 42
Total pay this week is 430.0
```

Case 8. BMI Calculator

Write a program which asks users to input their weight (in lbs) and height (in inches), and compute their body mass index (BMI), which is defind as $BMI = \frac{W}{H^2}$, where W is their weight converted to kg, and H is their height converted to m. (Assume 1 kg=2.205 lbs, and 1 m = 39.37 inches.) Finally, classify their BMI according to the following chart. The program should use the `inputNumber` function from Case 6 for input checking.

	BMI (kg/m^2)
Underweight:	<18.5
Normal:	18.5 -- 25
Overweight:	25 -- 30
Obese:	>30

```
[31]: lbs=inputNumber('Please input your weight (in lbs): ')
      inches=inputNumber('Please input your height (in inches): ')
      kgs=lbs/2.205
      m=inches/39.37
```

```
    bmi=kgs/m**2
    if bmi<18.5:
        answer='underweight'
    elif bmi<=25:
        answer='normal'
    elif bmi<=30:
        answer='overweight'
    else:
        answer='obese'
    print('According to the above chart, your BMI is',round(bmi,2),'which is',answer+'.')
```

```
Please input your weight (in lbs): 143
Please input your height (in inches): 5'11''
Invalid. Input must be a number.
Please input your height (in inches): 71
According to the above chart, your BMI is 19.94 which is normal.
```