# Week 4 Functions

## Part 1 Functions in Python

## Exercise 1 - 99 Bottles of Beer ¶

For this exercise please refer the solution already provided to you on Blackboard

## Exercise 2 - Powers

### 1,2

In [1]:

```
def RetSquared(Number):
    # Returns the square
    Squared = Number ** 2
    return(Squared)

# Testing our new function

for i in range(1,10):
    print(RetSquared(i))
```

```
1
4
9
16
25
36
49
64
81
```

### 3

In [2]:

```
def RetCubed(Number):
    # Returns the square
    Squared = Number ** 3
    return(Squared)

for i in range(1,10):
    print(RetCubed(i))
```

```
1
8
27
64
125
216
343
512
729
```

## 4,5,6

In [3]:

```
def RetPower(Number, Power = 1): # Power = 1 assigns the default value for power
    return Number ** Power

print([RetPower(2,x) for x in range(1,11)]) # I used list comprehension to create a lis
t with all the result and print it you could alternatively use a for loop for the same
 range
```

```
[2, 4, 8, 16, 32, 64, 128, 256, 512, 1024]
```

## 7

Below are two implementations of the power function, RetPowers uses list comprehension and RetPowers2 uses a standard for loop

In [4]:

```
def RetPowers(Number, Powers): #here powers will be included in a list
    return [Number ** x for x in Powers]

def RetPowers2(Number, Powers):
    result = []
    for x in Powers:
        result.append(Number ** x)
    return result

## testing to make sure that the output is the same
print(RetPowers(2,[1,2,3,10]))
print(RetPowers2(2,[1,2,3,10]))
```

```
[2, 4, 8, 1024]
[2, 4, 8, 1024]
```

## 8

Prime factors (if they exist if not $n$ is prime) for $n$ are contained between 2 and the $sqrt(n)$. We therefore start at the smallest prime and check if it's square is less and equal to the number, this is the condition for the while loop. This means that the loop will start at two and at most $sqrt(n)$. If the number is NOT a factor then we increment factor by one otherwise if it is then we divided the number by its factor. This will prevent us from including non prime factors. At the end of the loop if n is greater than 1 it means that the number itself is prime therefore it is appended to the list as the sole factor.
For instance the number 15 will go through the following steps:

### Step 1:

factor = 2
2 doesn't divide 15
increment factor
factor is now 3

### Step 2:

3 divides 15
Now we are looking for factor of 5
3 is appended to the list.

### Step 3:

3 doesn't divide 5
increment factor
factor is now 4

### Step 4:

Similar to tree factor is now 5

### Step 5:

5 divdes 5, number is now 1 and is added to the list of factor
The loop condition is broken, exit the loop

### Step 6

n is not greater than 1, the list of factors is returned

See https://stackoverflow.com/questions/15347174/python-finding-prime-factors (https://stackoverflow.com/questions/15347174/python-finding-prime-factors) for more information

In [5]:

```
def PrimeFactors(n):
    factor = 2 ## init to 2 the smallest prime
    factors = []
    while factor ** 2 <= n:
        if n % factor:
            factor += 1
        else:
            n //= i
            factors.append(factor)
    if n > 1:
        factors.append(n)
    return factors
```

## Exercise 3 Word Scrambler

### 1,2

let's divide this Exercise into multiple function as it will make it easier to follow Firstly let's define a text example

In [67]:

```
sample_text = "Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nam maximus feu
giat nisl at faucibus. Vivamus id nisl odio. Aenean porttitor vehicula tellus. In ac mo
lestie augue. Vivamus non elit justo. Sed scelerisque vehicula tellus, eu aliquam augue
pulvinar a. Nam at pulvinar ligula. Suspendisse potenti. Nulla quis nunc a lectus sempe
r faucibus. Aenean ullamcorper massa ut leo posuere tristique. Etiam vulputate dignissi
m tincidunt. Sed iaculis orci quis metus aliquet, at mattis nisl elementum. In eros eni
m, finibus at libero a, tristique luctus urna."
# This is our sample text
```

In [7]:

```
import random
def wordscramble(word): # This function scrambles a single word without punctuation
    char_list = list(word)
    if len(char_list) == 3:
        return "".join(char_list)
    else:
        temp = char_list[1:-1]
        random.shuffle(temp)
        temp.insert(0,char_list[0])
        temp.insert(len(char_list)-1,char_list[-1])
    return "".join(temp)

# Let's test our function
wordscramble("Pizza")
```

Out[7]:

```
'Pziza'
```

We can now scramble a single word let's use this function for an entire list of words. The map() function allows us to do this efficiently. Map takes a functions that accepts a single argument and maps it to a list. We use to list() function to return the result from map(). This

In [8]:

```
result = list(map(wordscramble,"the quick brown fox jumps over the lazy dog".split()))
# alternatively you can use a for loop
print(result)
# We could interlace spaces between words to get the orignal sentence back
print(" ".join(result))
```

```
['the', 'quick', 'bworn', 'fox', 'jupms', 'over', 'the', 'lzay', 'dog']
the quick bworn fox jupms over the lzay dog
```

If you enclose the above in a function you can now make now call this function on a user's input or on any string that doesn't have punctuation
To take care of punction we need to do the following:

In [9]:

```
import string
invalidChars = set(string.punctuation.replace("_", "")) # list of invalid character
setofdigits  = set(list(string.digits)) # set of digits
immutableset = invalidChars | setofdigits # union of both set representing characters w
e don't want to scramble
print(invalidChars)
print(setofdigits)
print(immutableset)
```

```
{'+', '$', ',', '&', '*', '|', ':', '(', '{', '%', ']', '#', '~', '.',
'[', '}', '=', '\\', '>', '@', '^', '"', ')', '?', '`', ';', '!', '<', '-
', '/', "'"}
{'6', '3', '2', '9', '5', '0', '7', '8', '1', '4'}
{'+', '$', ',', '&', '5', '*', '0', '8', '|', ':', '1', '(', '{', '%',
']', '#', '~', '.', '[', '6', '}', '=', '9', '\\', '>', '@', '^', '4',
'"', ')', '?', '`', '3', ';', '2', '!', '<', '-', '/', '7', "'"}
```

We now have a lis of punction and strings if a string contains any of these it should not be changed

In [71]:

```
def indexofpunc(word,punc): # this function returns the index of a a given punctuation
    indices = [i for i,e in enumerate(word) if e == punc]
    if len(indices) == 0:
       return []
    else:
        return indices


def allindexofpunc(word): # returns the index for all punctuation and digits uses the f
unction above
    result = []
    for i in immutableset:
        result += indexofpunc(word,i)
    return result

def wordscrambler(sentence):
    list_words = sentence.split(" ") # split the sentence into individual word
    result = []
    for words in list_words:
        toremove = allindexofpunc(words) # we will remove all punctuation and then add
 it back in
        list_of_chars = list(words)
        list_of_chars_no_punc = list_of_chars
        if len(toremove) != 0:
             list_of_chars_no_punc = [c for i,c in enumerate(list_of_chars) if i not in
toremove] # our list of words devoid of punctuation
        if len(list_of_chars_no_punc) == 0:
            result.append("".join(list_of_chars))
        else:
            temp = wordscramble("".join(list_of_chars_no_punc)) # we now rejoin the wor
d and scramble it, yes this not optimal but allows to re-use the function  we made prev
iously.
            temp = list(temp)
            list(map(lambda x: temp.insert(x,list_of_chars[x]),toremove)) # add the mis
sing char back
            result.append("".join(temp))
    return " ".join(result)
```

In [76]:

```
print(wordscrambler(sample_text))
print()
print(wordscrambler("doesn't should've shouldn't've"))
```

```
Lorem ipusm dolor sit amet, centouectsr aindpcisig eilt. Nam mamxuis feagi
ut nsil at fuiacbus. Vumvais id nsil odio. Aaenen poroitttr vleciuha tllue
s. In ac mtoeslie augue. Vaiumvs non elit jutso. Sed slsirceqeue viclueha
tllues, eu ailuaqm augue puanilvr a.a Nam at pnuvalir llugia. Snuisdspese
ptotnei. Nlula quis nnuc aa letcus spemer fuuciabs. Aeenan umpelcrloar mss
aa ut leo pesuore tiirsqtue. Eaitm vtulapute dgsiisinm tindniuct. Sed iali
cus ocri qius metus aluieqt, at mitats nisl elneetumm. In eros enim, fibiu
ns at leibro a,a tiqustire luucts urna.

dneso't sldvuh'oe studlno'h've
```

As you can see we can now scramble words with punctuation and keep digits untouched if you wanted to shuffle every other letter you could use the enumerate function to only shuffle odd indices or even. You could also add to this program to make word such as "should've" don't scramble the "ve".

## Part 2 Function Arguments and Scope

## Exercise 4

### 1,2,3,4

In [79]:

```
def F():
    print(S)
S = "I hate spam"
F()
```

```
I hate spam
```

The function can access global argument

In [80]:

```
def F():
    S = "Me too"
    print(S)
S = "I hate spam"
F()
print(S)
```

```
Me too
I hate spam
```

The function defines S to "me too" which is local to itself globally S hasn't changed

In [82]:

```
def F():
    global S
    S = "Me too"
    print(S)
S = "I hate spam"
F()
print(S)
```

```
Me too
Me too
```

The variable S labeled as global, no local variable is created changed on S are reflected outside the function as well

In [85]:

```
del S # deleted the variable due to how jupyter cell work
def F():
    S = "I am a variable"
    print(S)
F()
print(S)
```

```
---------------------------------------------------------------------------
-
NameError                                 Traceback (most recent call las
t)
<ipython-input-85-8042dab085b7> in <module>
----> 1 del S # deleted the variable due to how jupyter cell work
      2 def F():
      3     S = "I am a variable"
      4     print(S)
      5 F()

NameError: name 'S' is not defined
```

S is not defined in the global scope

## Exercise 5 - Variadic Functions

### 1,2

In [92]:

```
def sumofnumbers(*args):
    return sum(args)
print(sumofnumbers(1,2,3,4,5,6,7,8,9,10))

def sumofnumbers2(*args):
    acc = 0
    for i in args:
        acc += i
    return acc

print(sumofnumbers2(1,2,3,4,5,6,7,8,9,10))
```

```
55
55
```

Above are two implementation of the sum of number function using the sum() function or using a for loop

In [101]:

```
import math
def prodofnumbers(*args): # you could also use a for loop return a variable called prob
initialized at 1
    return math.prod(args)

print(prodofnumbers(1,2,3,4,5,6,7,8,9,10))
```

```
25401600
```

In [102]:

```
def operations(*args):
    return min(args), max(args), sum(args)/len(args)

operations(1,2,3,4,5,21,7,8,9,10)
```

Out[102]:

```
(1, 21, 7.0)
```

## Part 3 Recursive functions

## Exercise 6 - Recurisve functions

### 1,2

Please refer to the provided fibonacci code.
Another popular recursive function is the factorial function https://www.mathsisfun.com/numbers/factorial.html (https://www.mathsisfun.com/numbers/factorial.html)

In [104]:

```
def fac(n):
    if n == 0:
        return 1
    else:
        return n*fac(n-1)

fac(10)
```

Out[104]:

```
3628800
```

## Exercise 7

### For 1 look at the solutions to week 2 as the circle funcito it is already given as function

### 2 https://www.w3schools.com/python/python_dictionaries.asp (https://www.w3schools.com/python/python_dictionaries.asp) Dictionaries can be used in data science to store labeled data with the key being the label and the data being the value. This in conjunction with another module called pickle can help you save your dataset efficiently

### 3 Refer to this tutorial: https://www.geeksforgeeks.org/python-program-for-tower-of-hanoi/ (https://www.geeksforgeeks.org/python-program-for-tower-of-hanoi/) for recursive solution. Another interesting ways to solve this problem is to use binary. The youtuber 3blue1brown explains this quite well: https://www.youtube.com/watch?v=bdMfjfT0lKk (https://www.youtube.com/watch?v=bdMfjfT0lKk)

### 4 https://www.geeksforgeeks.org/hangman-game-python/#:~:text=This%20is%20a%20simple%20Hangman%20game%20using%2 (https://www.geeksforgeeks.org/hangman-game-python/#:~:text=This%20is%20a%20simple%20Hangman%20game%20using%2 This interesting tutorial to follow

### 5 Following the rules of blackjack you can implement a deck of cards efficiently as follows:

In [121]:

```
card_ranks =['A','K','Q','J','2','3','4','5','6','7','8','9','10']
card_suits =['Heart','CLUB','DIAMOND','SPADE']
card_deck  = list(map(lambda x: list(map(lambda y: [x,y],card_ranks)),card_suits))
card_deck  = card_deck[0] + card_deck[1] + card_deck[2]+ card_deck[3]
print(card_deck)
```

```
[['Heart', 'A'], ['Heart', 'K'], ['Heart', 'Q'], ['Heart', 'J'], ['Heart',
'2'], ['Heart', '3'], ['Heart', '4'], ['Heart', '5'], ['Heart', '6'], ['He
art', '7'], ['Heart', '8'], ['Heart', '9'], ['Heart', '10'], ['CLUB',
'A'], ['CLUB', 'K'], ['CLUB', 'Q'], ['CLUB', 'J'], ['CLUB', '2'], ['CLUB',
'3'], ['CLUB', '4'], ['CLUB', '5'], ['CLUB', '6'], ['CLUB', '7'], ['CLUB',
'8'], ['CLUB', '9'], ['CLUB', '10'], ['DIAMOND', 'A'], ['DIAMOND', 'K'],
['DIAMOND', 'Q'], ['DIAMOND', 'J'], ['DIAMOND', '2'], ['DIAMOND', '3'],
['DIAMOND', '4'], ['DIAMOND', '5'], ['DIAMOND', '6'], ['DIAMOND', '7'],
['DIAMOND', '8'], ['DIAMOND', '9'], ['DIAMOND', '10'], ['SPADE', 'A'], ['S
PADE', 'K'], ['SPADE', 'Q'], ['SPADE', 'J'], ['SPADE', '2'], ['SPADE',
'3'], ['SPADE', '4'], ['SPADE', '5'], ['SPADE', '6'], ['SPADE', '7'], ['SP
ADE', '8'], ['SPADE', '9'], ['SPADE', '10']]
```

you now have a list you containing all the cards which you can shuffle and use in your black jack implementation