Part 1 - Functions

Exercise 1

Question 1.

In [15]:
```
def RetSqaured(Number):
    # Returns the square
    Squared = Number ** 2
    return Squared
```

Question 2.

In [18]:
```
for n in range(1,11):
    print(RetSqaured(n))
```

```
1
4
9
16
25
36
49
64
81
100
```

Question 3.

In [19]:
```
def CalcPower(Number, Power):
    return Number ** Power
```

Question 4.

In [20]:
```
for p in range(1, 11):
    print(CalcPower(2, p))
```

```
2
4
8
16
32
64
128
256
512
1024
```

Question 5.

In [22]:
```
def CalcPower(Number, Power = 1):
    return Number ** Power
```

Question 6. Here's one solution to this problem

In [23]:
```
def CalcPowers(Number, Powers):
    ans = []
    for p in Powers:
        ans.append(Number ** p)
    return ans

print(CalcPowers(2, [1, 3, 4]))
```

```
[2, 8, 16]
```

We can also solve this problem using list comprehensions

In [24]:
```
def CalcPowers(Number, Powers):
    return [Number ** p for p in Powers]

print(CalcPowers(2, [1, 3, 4]))
```

```
[2, 8, 16]
```

The unpacking operator * can also be used when dealing with an unknown number of arguments

---

Question 7.

In [28]:
```python
def inverse(x):
    if x == 0:
        return 'undefined'
    else:
        return 1 / x

print(inverse(2))
print(inverse(0))
```

```
0.5
undefined
```

Question 8.

In [29]:
```python
def swap(a, b):
    """
    This function swaps the values of two variables a and b by
    returning them in the opposite order.  This function should
    be called as follows: a, b = swap(a, b)
    """
    return (b, a)

help(swap)

a = ['red', 'blue', 'green']
b = {1, 3, 5}

a, b = swap(a, b)
print(a)
print(b)
```

```
Help on function swap in module __main__:

swap(a, b)
    This function swaps the values of two variables a and b by
    returning them in the opposite order.  This function should
    be called as follows: a, b = swap(a,b)

{1, 3, 5}
['red', 'blue', 'green']
```

Exercise 2 - Modelling with functions

Question 1

In [2]:
```
a = 0
b = 1

def y(x):
    return x**2


for N in [1, 10, 100]:

    dx = (b - a) / N
    A = 1/2 * (y(a) + y(b)) * dx

    for i in range(1, N):
        x_i = a + i * dx
        A += y(x_i) * dx

    print('A = ', A)
```

```
A =  0.5
A =  0.3350000000000001
A =  0.33335000000000015
```

In [1]:
```
from math import *

# Physical parameters.  A dictionary is used to store the parameters
L = 0.1
params = {"alpha":2, "beta":0.2}

# Domain of integration
a = 0
b = L

# Function that defines the curve; parameters are passed as an argument
def y(x, params):

    return params["alpha"] * exp(-x**2 / params["beta"]**2)

# a for loop to compute the area for different values of N
for N in [1, 10, 100, 1000]:

    dx = (b - a) / N
    A = 1/2 * (y(a, params) + y(b, params)) * dx

    for i in range(1, N):
        x_i = a + i * dx
        A += y(x_i, params) * dx

    # print the value of A to the screen
    print('A = ', A)
```

```
A =  0.1778800783071405
A =  0.18444748897241
A =  0.1845117535631123
A =  0.18451239607511036
```

In [12]:
```python
def trapezium(y, a, b, N, params = {}):
    """
    This function computes the area under a curve y(x) from
    x = a to x = b using the trapezium rule with N trapezoids

    Inputs:
    y(x, params) - a function to compute the value of the curve
    a - the left point of the area
    b - the right point of the area
    N - the number of trapezoids to use
    params - a dictionary of parameters, set to an empty dict by default
    """

    dx = (b - a) / N
    A = 1/2 * (y(a, params) + y(b, params)) * dx

    for i in range(1, N):
        x_i = a + i * dx
        A += y(x_i, params) * dx


    return A

# define the curves
def y(x, params):
    return x**2

A = trapezium(y, 0, 1, 100)
print(A)


params = {"theta": 0.5}
def y2(x, params):
    return exp(-params["theta"] * x)

A = trapezium(y2, -1, 1, 1000, params)
print(A)
```

```
0.33335000000000015
2.084381395673423
```

Part 2 - Function arguments and scope

Exercise 3 - Variadic functions

Question 1.

```
In [68]: def sum_prod(*numbers):

             # set the value of the sum (s) to zero
             s = 0
             # set the value of the product (p) to one
             p = 1

             for n in numbers:
                 s += n
                 p *= n

             return s, p

         s, p = sum_prod(1, 2, 3, 4)
         print(s, p)
```

```
10 24
```

Question 2.. Here we use some of Python's built-in functions. Recall that `len` computes the number of entries in a tuple

```
In [70]: def analyse_numbers(*numbers):

             return min(numbers), max(numbers), sum(numbers) / len(numbers)

         maximum, minimum, mean = analyse_numbers(1, 2, 3, 4, 5, 6, 7)
         print(maximum)
         print(minimum)
         print(mean)
```

```
1
7
4.0
```

Exercise 4 - Variable scope

Question 1. This prints "I hate spam" because S is a global variable (since it is defined in the main body of code)

```
In [73]: def F():
             print(S)

         S = "I hate spam"
         F()
```

```
I hate spam
```

Question 2. This prints "Me too" followed by "I hate spam". When S is defined locally in the function F, it overrides the value of S defined in the main body of Python code

In [74]:
```
def F():
    S = "Me too"
    print(S)

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

```
Me too
I hate spam
```

Question 3. This prints "Me too" twice. In the function F we declare that S is a global variable with global scope. Therefore, when the function F is called, it overwrites the value of S that was set in the main body of code.

In [75]:
```
def F():
    global S
    S = "Me too"
    print(S)

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

```
Me too
Me too
```

Question 4. This creates an error. Although the function F runs correctly when it is called, the variable T has local scope and cannot be accessed outside of this function. Therefore, calling print(T) leads to an error since T is not defined

In [77]:
```
def F():
    T = "I am a variable"
    print(T)

F()
print(T)
```

```
I am a variable

---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-77-55ad425ac11d> in <module>
      4
      5 F()
----> 6 print(T)

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

Part 3 - Recursive functions

Exercise 5 - Recursive functions

Question 1.

In [80]:
```
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n - 1)

print(factorial(5))
```

```
120
```

Question 2 and 3

In [155]:
```
def Fib(n):
    if n == 1 or n == 2:
        return 1
    else:
        return Fib(n-1) + Fib(n-2)

for i in range(1,15):
    print(Fib(i), end = " ")
```

```
1 1 2 3 5 8 13 21 34 55 89 144 233 377
```

Part 4 - Advanced questions

Exercise 6 - More functions

Question 1.

In [21]:
```
def median(a, b, c):
    if a < b:
        if b < c:
            return b
        if c < a:
            return a
    else:
        if a < c:
            return a
        if c < b:
            return b

print(median(5, 4, 3))
```

```
4
```

Question 2.: In this solution, we convert the integer into a string. We then loop through the entries of the string, convert them into an integer, and sum them up.

In [81]:
```
def sum_digits(N):
    string = str(N)
    S = 0
    for s in string:
        S += int(s)
    return S

print(sum_digits(1234))
```

```
10
```

Another solution is to use maths without converting between types. The trick here is to note that the digits of a number N are the divisors of powers of 10 and can be calculated using floor division. For example, $1234 = 1 * 10^3 + 2 * 10^2 + 3 * 10^1 + 4 * 10^0$. The first step in the code is to calculate the largest power of 10 that (floor) divides N. Let's call this power p. In the example when $N = 1234$ we have $p = 3$.

Once we have p, we can extract the first digit using floor division and calculating $N//10^p$. In the example above, $N//10^3 = 1$, which is indeed the first digit.

To calculate the second digit in the example, we use the fact that $1234 - 1 * 10^3 = 2 * 10^2 + 3 * 10^1 + 4 * 10^0$. So, by substracting off the first digit times 10^p from N, we can floor divide the result by 10^{p-1} to obtain the second digit.

We then repeat this process to extract all of the digits and sum them up

In [105]:
```
def sum_digits(N):

    # use floor division to calculate the largest power of 10 that divides N
    p = 0
    while N // 10**(p+1) > 0:
        p += 1

    # set the sum of the digits to zero
    S = 0

    # extract the digits of N
    while p >= 0:

        # calculate the digit
        digit = N // 10**p

        # add the digit to the sum
        S += digit

        # substract off the current digit times the current power of ten
        N -= digit * 10**p

        # decrease the power of 10
        p -= 1

    return S

print(sum_digits(1234))
```

```
10
```

Question 3. We solve this problem in two parts. First we create a function that determines whether an integer is prime. Then we create a function that computes the prime factorization. The first thing we do here is create a list of all of the prime numbers that are between 2 and N. We then divide N by these prime numbers. If there is zero remainder, then we've found a prime factor. We then reassign the value of N by dividing it by this prime factor and we repeat the calculation

In [149]:
```
# define a function that tests whether a number is prime
def is_prime(N):
    if N == 1:
        return True
    else:
        for i in range(2, N):
            if N % i == 0:
                return False

        return True

# the function that computes the prime factorizations
def prime_factorization(N):

    # a list of prime factors
    factors = []

    # a list of prime numbers between 2 and N
    primes = [n for n in range(2, N+1) if is_prime(n)]

    # looping through the primes to find the factors
    while N > 1:
        for p in primes:
            if N % p == 0:
                factors.append(p)
                N = int(N / p)
                break

    # return the list
    return factors

print(prime_factorization(147))
```

```
[3, 7, 7]
```

Question 4.. There are many ways to sort a list. In the approach below, we sort a list L by looking for the smallest element, second-smallest, third-smallest, etc and then building a list with these values.

We begin by looking for the smallest number in the list L (called min_L with index idx). We then swap the position of the smallest entry (i.e. min_L) and the first entry of L (i.e. L[0])

We then examine the sub-list L[1,:], which consists of all of the entries of L except the first. We then find the smallest entry of this sublist, which will be the second-smallest number in the original list L. We save the value of the smallest number as min_L and its index as idx. We then swap the first entry in the sublist L[1,:], which is L[1], with its smallest value. The result is that the first two entries of the list L will now have the smallest and second-smallest numbers.

We then consider the sub-list L[2,:] and find the smallest number, corresponding to the third-smallest number of L. We then swap this with L[2]. The process then repeats

In [3]:
```
def my_sort(L):

    # first we find the length of L
    N = len(L)

    # we use a for loop to examine the lists L[:], L[1:], L[2:] and find the smal
    for i in range(0, N):

        # this bit of code finds the smallest number in the sub-list given by L[i
        min_L = L[i]
        idx = i
        for j in range(i+1, N):
            if L[j] < min_L:
                min_L = L[j]
                idx = j

        # now we swap the i-th entry of L with the minimum value in the sublist L
        L[idx] = L[i]
        L[i] = min_L

    return L

L = [4, 3, 2, 6, 3, 2, 6, 7]
S = my_sort(L)
print(S)
L.sort()
print(L)
```

```
[2, 2, 3, 3, 4, 6, 6, 7]
[2, 2, 3, 3, 4, 6, 6, 7]
```