Recursions

Definition of Recursion (Retrieved from https://www.python-course.eu/recursive_functions.php)

Recursion is a way of programming or coding a problem, in which a function calls itself one or more times in its body. Usually, it is returning the return value of this function call. If a function definition fulfils the condition of recursion, we call this function a recursive function.

Termination condition:
A recursive function has to terminate to be used in a program. A recursive function terminates, if with every recursive call the solution of the problem is downsized and moves towards a base case. A base case is a case, where the problem can be solved without further recursion. A recursion can lead to an infinite loop, if the base case is not met in the calls.

```
# Display a message recursively
def message(times):
    if times > 0:
        print('This is a recursive function.')
        message(times - 1)

# Start the program
message(5)
```

```
# This program has a recursive function. Infinite output. Press Ctl-C to stop.
def message():
    print('This is a recursive function.')
    message()

# Call the main function.
message()
```

```
This is a recursive function.
This is a recursive function.
This is a recursive function.
This is a recursive function.
This is a recursive function.
This is a recursive function.
This is a recursive function.
Traceback (most recent call last):
File "/Users/staff/Downloads/TEMP/p1.py", line 3, in message
    print ('This is a recursive function.')
KeyboardInterrupt
```

```
# This program uses recursion to calculate the factorial of a number.

def main():
    # Get a number from the user.
    number = int(input('Enter a nonnegative integer: '))

    # Get the factorial of the number.
    fact = factorial(number)

    # Display the factorial.
    print('The factorial of', number, 'is', fact)

# The factorial function uses recursion to calculate the factorial of its argument,
# which is assumed to be nonnegative.
def factorial(num):
    if num == 0:
        return 1
    else:
        return num * factorial(num - 1)

# Call the main function.
main()
```

```
Enter a nonnegative integer: 5
The factorial of 5 is 120
```

```
# The rangeSum function returns the sum of a specified range of items in num_list.
# The start parameter specifies the index of the starting item. The end
# parameter specifies the index of the ending item.

def rangeSum(num_list, start, end):
    if start > end:
        return 0
    else:
        return num_list[start] + rangeSum(num_list, start + 1, end)

# Start the program

# Create a list of numbers.
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9]

# Get the sum of the items at indexes through 5.
my_sum = rangeSum(numbers, 2, 5)

# Display the sum.
print('The sum of items 2 through 5 is', my_sum)
```

```
The sum of items 2 through 5 is 18
```

```
# Python Program for recursive binary search.
# Retrieved from https://www.geeksforgeeks.org/python-program-for-binary-search/

# Returns index of x in arr if present, else -1
def binarySearch (arr, l, r, x):

    # Check base case
    if r >= l:

        mid = l + (r - l)/2

        # If element is present at the middle itself
        if arr[mid] == x:
            return mid
        # If element is smaller than mid, then it can only
        # be present in left subarray
        elif arr[mid] > x:
            return binarySearch(arr, l, mid-1, x)

        # Else the element can only be present in right subarray
        else:
            return binarySearch(arr, mid+1, r, x)

    else:
        # Element is not present in the array
        return -1


# Test array
arr = [ 2, 3, 4, 10, 40 ]
x = 10

# Function call
result = binarySearch(arr, 0, len(arr)-1, x)

if result != -1:
    print "Element is present at index %d" % result
else:
    print "Element is not present in array"
```

Output:

```
Element is present at index 3
```

```
# Iterative Binary Search Function
# It returns location of x in given array arr if present, else returns -1
# Retrieved from https://www.geeksforgeeks.org/python-program-for-binary-search/

def binarySearch(arr, l, r, x):

    while l <= r:

        mid = l + (r - l)/2;

        # Check if x is present at mid
        if arr[mid] == x:
            return mid
        # If x is greater, ignore left half
        elif arr[mid] < x:
            l = mid + 1

        # If x is smaller, ignore right half
        else:
            r = mid - 1

    # If we reach here, then the element was not present
    return -1


# Test array
arr = [ 2, 3, 4, 10, 40 ]
x = 10

# Function call
result = binarySearch(arr, 0, len(arr)-1, x)

if result != -1:
    print "Element is present at index %d" % result
else:
    print "Element is not present in array"
```

Output:

```
Element is present at index 3
```

The following notes were retrieved from https://www.programiz.com/python-programming/recursion

Advantages of Recursion

1. Recursive functions make the code look clean and elegant.
2. A complex task can be broken down into simpler sub-problems using recursion.
3. Sequence generation is easier with recursion than using some nested iteration.

Disadvantages of Recursion

1. Sometimes the logic behind recursion is hard to follow through.
2. Recursive calls are expensive (inefficient) as they take up a lot of memory and time.
3. Recursive functions are hard to debug.