Recursion

A recursive function is a function that calls itself.

Recursion can be used instead of iteration, but it is generally less efficient.

A recursive function must contain a **general case** and at least one **base case**. A base case is used to determine the condition for the recursion to stop. A recursive function must have a base case.

Without a base case the recursion will be stuck in an infinite recursion depth.

Model for recursive functions

```
function
 if some condition is met
  return base case statements
 else
  general case statements
  functions calls itself
```

Factorial is an example of an augmenting recursive function that has pending operations that get performed on return from each recursive call

```
def factorial(n):
 if n == 1: # base case
  return 1
 else: # general case
  return n * factorial(n-1)
```

Example: Trace code where n=8

n	Call		Return
1	Call 8	factorial(1)	1
2	Call 7	factorial(2)	2
3	Call 6	factorial(3)	6
4	Call 5	factorial(4)	24
5	Call 4	factorial(5)	120
6	Call 3	factorial(6)	720
7	Call 2	factorial(7)	5040
8	Call 1	factorial(8)	40320

Programming paradigms

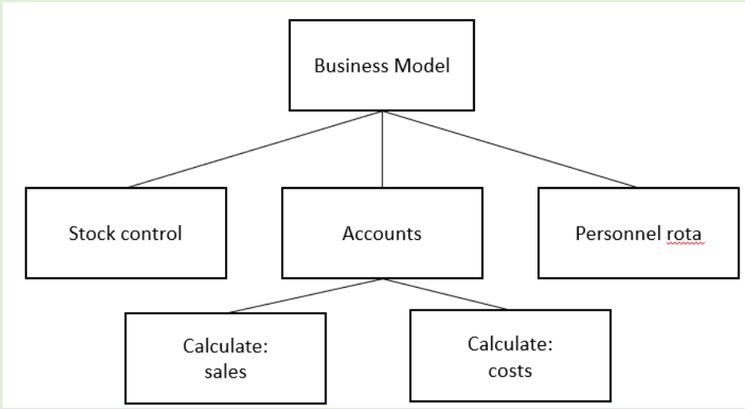
Procedural versus object-oriented paradigm

Procedural	Object oriented programming
Code is divided into functions Focus is placed on the functions Procedures contain a sequence of instructions that are executed sequentially step-by-step Data are passed into functions through parameters and returned. Data and functions are separate	Code is divided into objects Focus is placed on the data Objects have attributes (data) and methods (functions) that perform operations on the data Objects model the real world more closely by handling data and procedures together. Objects can interact with one another Object oriented code is easier to reuse

Procedural-oriented programming

Structured programming divides a computer program into sererate sub programs / modules. This is important for large coding projects allowing **decomposition** of the problem. This means the code is easier to debug and make changes and allows the reuse of code.

Example hierarchy chart with corresponding code



Stack Frames

The Call stack is a dynamic data structure stored in RAM.

Controls how functions call each other and how functions pass parameters to each other.

Each time a call is made to a function the following details are added to the stack frame:

- Return addresses
- Parameters
- local variables

This is necessary so the algorithm can proceed from where the function was called once that function has been executed. It retrieves the necessary data from the stack before the call to the function was made.

Example stack in operation

```
1.    main(name)
2.      time=0900
3.      g=greeting(name,time)
4.      return g

5.    greeting(name,time)
6.      if time < 1200
7.        hello(name)
8.        return True
9.      else
10.       return False

11.   hello(name)
12.     g = "Good morning "+name
13.     return g

14.   g=main("Homer")
15.   print(g)
```


Stack Frame	Line order of operation
Call 1: main() Return address: line 14 Parameters: name="Homer" Local variables: time=0900	14 Call 1
Call 2: greeting() Return address: line 3 Parameters: name="Homer", time="0900" Local variables: None Call 1: main() Return address: line 14 Parameters: name="Homer" Local variables: time=0900	14 Call 1 1 2 3 Call 2
Call 3: hello() Return address: line 8 Parameters: name="Homer" Local variables: g="Good morning Homer" Call 2: greeting() Return address: line 3 Parameters: name="Homer", time="0900" Local variables: None Call 1: main() Return address: line 14 Parameters: name="Homer" Local variables: time=0900	14 Call 1 1 2 3 Call 2 5 6 7 Call 3
Call 2: greeting() Return address: line 3 Parameters: name="Homer", time="0900" Local variables: None Call 1: main() Return address: line 14 Parameters: name="Homer" Local variables: time=0900	14 Call 1 1 2 3 Call 2 5 6 7 Call 3 11 12 13 8
Call 1: main() Return address: line 14 Parameters: name="Homer" Local variables: time=0900	14 Call 1 1 2 3 Call 2 5 6 7 Call 3 11 12 13 8 3
Stack empty	14 Call 1 1 2 3 Call 2 5 6 7 Call 3 11 12 13 8 3 4 14 15