THINKING LOGICALLY

DECISION POINTS

This section refers to identifying points in a solution when a decision has to be taken.

A Boolean expression or condition will evaluate two states, 1 True or 0 False.

If we are testing a condition that has only two possible states (Boolean) it is suitable to use a simple if statement.

```
IF BOOLEAN EXPRESSION THEN
        …..
ELSE:
        ……
END IF
```

Sometimes we may have a loop with a Boolean expression:

```
WHILE BOOLEAN EXPRESSION DO
        ……..
END WHILE
```

```
DO (alternatively REPEAT)
        ……..
LOOP UNTIL BOOLEAN EXPRESSION
```

In cases that we need to make a decision with multiple outcomes. We can use IF and ELSE IF or more effectively, SELECT CASE

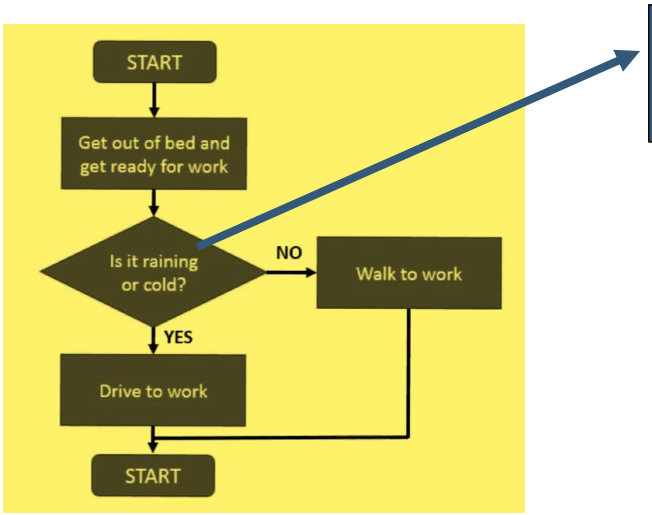
```
SELECT CASE variable
        CASE EXPRESSION(say >5)
                ….
        CASE EXPRESSION(say =6)

                ….
        CASE EXPRESSION(say < 0)
                ….
        CASE ELSE
                ….
END SELECT
```

DETERMINE THE LOGICAL CONDITIONS THAT AFFECT THE OTUCOEM OF A DECISION

In a problem, there may be numerous different solutions that are only appropriate for certain conditions.

E.g. Should I take an umbrella today or not. We must have a logical expression – Is it raining to determine the outcome.

e.g. 2)



HOW DOES A DECISION AFFECT THE FLOW THROUGH A PROGRAM

The results of the previous decision will determine the flow through a program. In essence this is the aftermath of an evaluative statement.

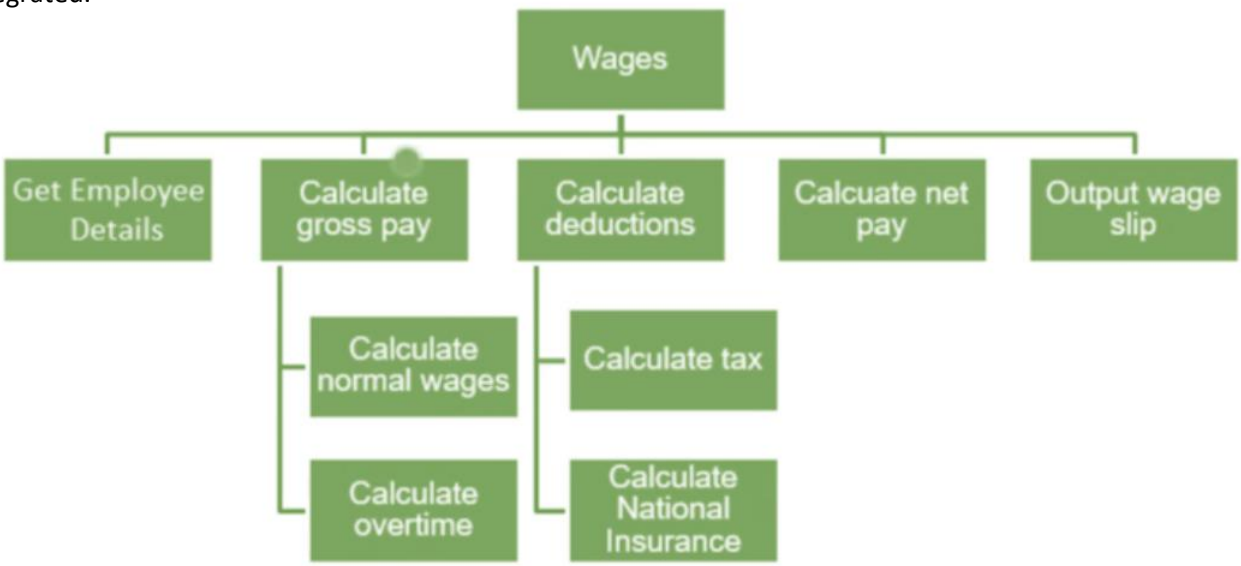
THNIKING PROCEDURALLY

IDENTIFYING THE COMPONENTS OF A PROBLEM

This is about working out the sub parts of a large problem. Lager complex tasks can be tackled in more manageable parts. An analyst must identify components of such tasks.

Step wise refinement is a method of breaking down a problem in a top down modular design. The task the program needs to run is split into sub tasks, these in turn are split into even simpler sub tasks.

This aims to produce small independent modules that can either be written by one person or a team. This allows programmers to write a solution to their isolated task without having to know the details of the larger problem, they only need to know their expected inputs and outputs. The modules can then be integrated.

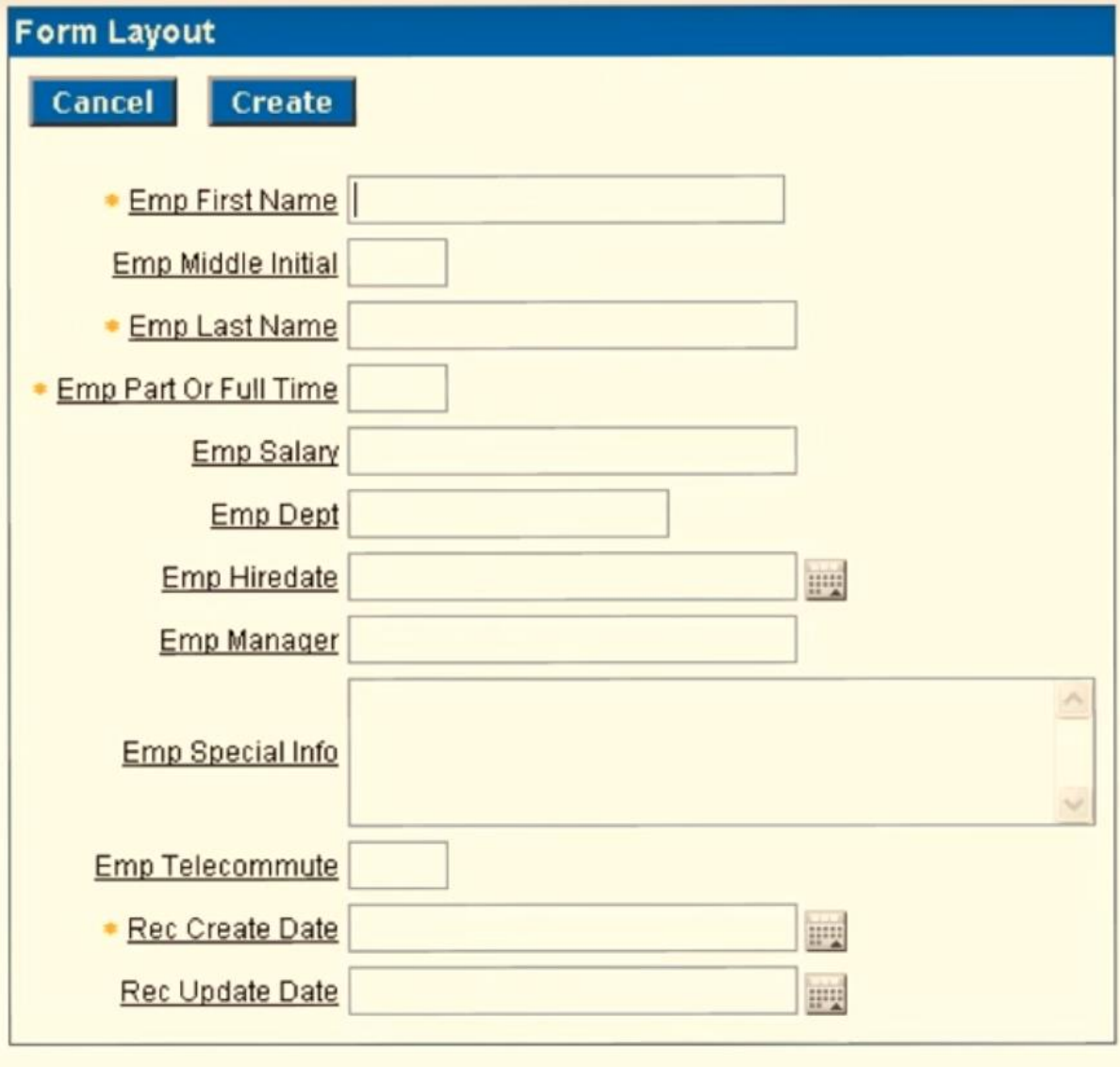


The tasks of a wage program is split into subsections. These subsections are split into components that need to be worked out first before integrated to form that subsection.

For instance, to calculate the Gross Pay: We must calculate the normal wages first (may be an algorithm in a function) as well as calculate the overtime pay (a different procedure). These two items can then be combined in that particular module.

Components of a solution to a particular module.

Components are items of code that are needed, e.g. an input field with associated code to retrieve the employ name, age….. or a calendar component to select the date of birth (a good example of reusable code).



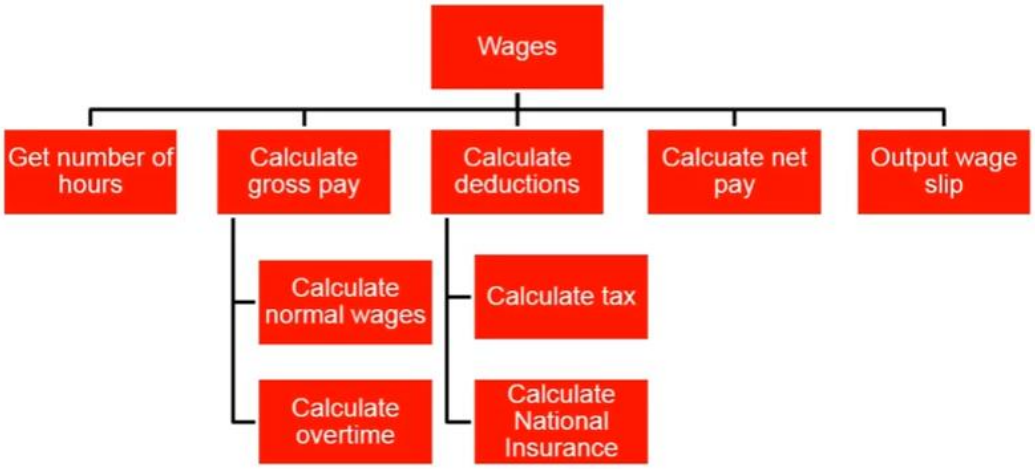
DETERMING THE STEPS NEEDED TO SOLVE A PROBLEM

A key point is that the order of steps taken to devise a solution should be considered. In some programs the order may be very complex or even unpredictable.
A word processor is a complicated computer program with a wide range of functionality. The user may use parts of the program in a random order and cannot be predicted by the programmer. Therefore, the programmer must deign the program to allow the use to access the components in any order.

Some other programs, the order of events is key and should be accounted for. An online booking form for a theatre should not allow the user to select their choice of seats before they have even chose a date. The payment form should be the last component of the program.

IDENTIFYING SUB-PROCEDURES TO SOLVE A PROGRAM

Single modules for a program should be coded to carry out a singular specific task.



Wages	
GetHours()	function which returns an integer in range 0 to 60
CalculateWages(Hours)	function which returns gross wage
CalculateNormalWages(Hours)	function which returns wage for up to 40 hours
CalculateOvertime(Hours)	function which returns pay for any hours over 40
CalculateDeductions(GrossWage)	function which returns total decutions
CalculateTax(GrossWage)	function which returns tax due
CalculateNI(GrossWage)	function which returns NI due
CalculateNetWage(GrossWage, Deductions)	function which returns net wage after deductions
OutputResults(Hours, GrossWage, Tax,NI, Deductions, NetWage)	Procedure to print the wage slip

Each of the modules are broken down into sub-procedure or functions. These can be handed to programmers to code. Breaking down modules into sub procedures like this is useful as it allows repetition to easily be spotted. IF two procedures or functions carry out the same task they can either be entirely reused or customised so saving time or money.