Shell Script

- A text file that contains a sequence of commands
  we add .sh at the end of the file to identify that this file is shell script
- **shebang line :** #!/bin/bash
  # --> Pound, Hash
  ! --> exclamation mark, bang
- specifies which program/shell should be called to run the script.
- Used to tell the operating system the path it should use to interpret the file
- shell script will run in bash
- should always be first line, won't work if we move it to any other line
- to see default shell : echo $SHELL
- if default shell is not bash then it will run in bash shell
- --> cat /etc/shells

- **-x : debugging mode**
- #!/bin/bash –x running the shell script in debugging mode

MINGW64:/c/Users/Chank/Desktop/TFP-627 Batch/Day3/ShellScript

```
  GNU nano 6.4     forloop.sh
#!/bin/bash -x

for files in `ls *.txt`
do
        echo $files;
done




^G Help    ^O Write O^W Where I^K Cut
^X Exit    ^R Read Fi^\ Replace^U Paste
```

Programming Construct

A Programming Construct is to control the order/flow in which instructions are executed.

In programming languages, the expression which translates to an instruction is called a programming statement or just statement.

Programming construct classification :

1. Sequences
2. Selection
3. Repetition

Sequence Statements

A sequence construct tells the CPU (processor) which statement is to be executed next.

Demonstrating add.sh & Execution Thread

BridgeLabz
Employability Delivered

```
GNU nano 2.0.6                File: add.sh

#!/bin/bash -x
x=100;
y=100;
z=$(( $x + $y ))
echo $z
```

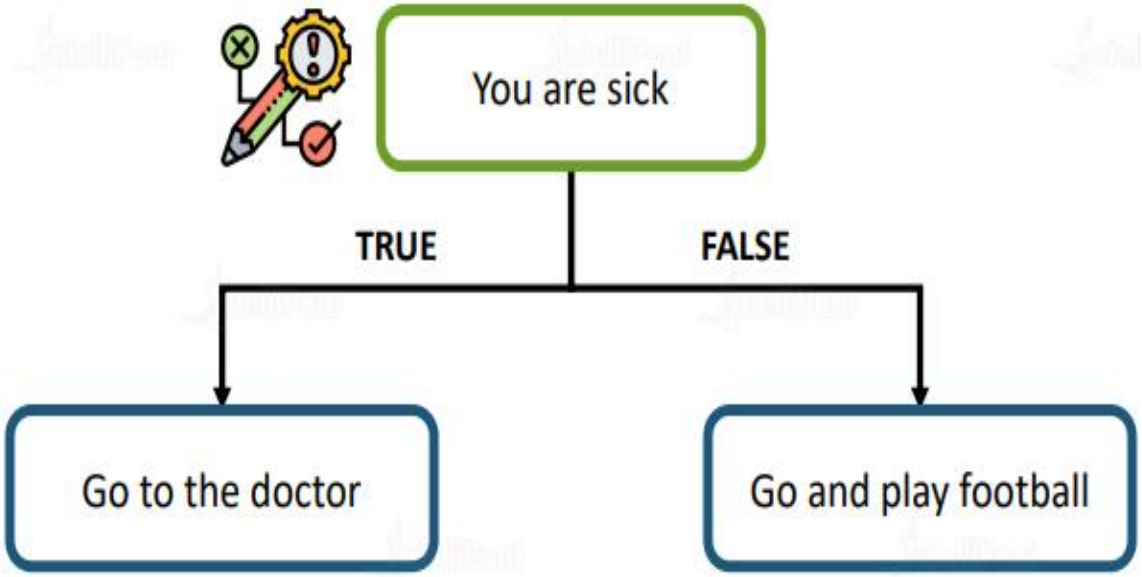
add.sh program with following Instructions
1. Variable $x Declaration and Assignment
2. Variable $y Declaration and Assignment
3. Addition of $x and $y and assigning to variable $z
4. Displaying $z
5. NOTE : Arithmetic Expression *$((expression))*

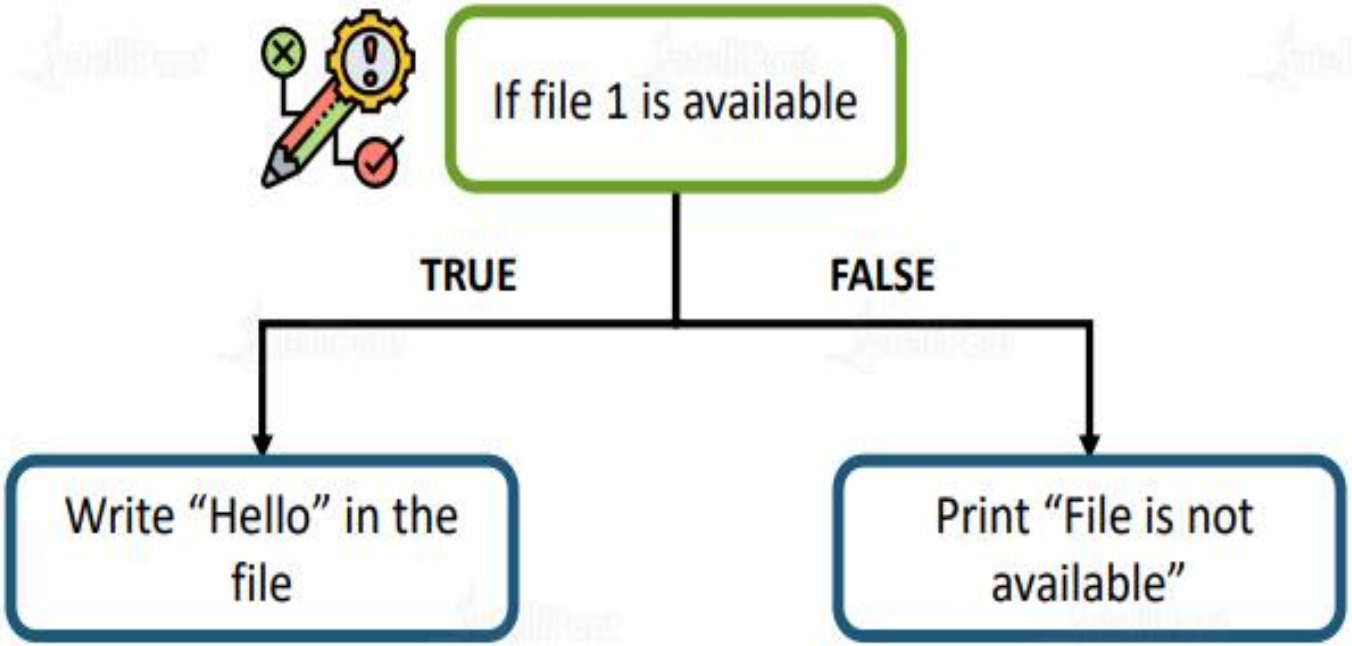
```
Narayans-MacBook-Pro:TerminalCommands narayan$ nano add.sh
Narayans-MacBook-Pro:TerminalCommands narayan$ ./add.sh
+ x=100
+ y=100
+ z=200
+ echo 200
200
Narayans-MacBook-Pro:TerminalCommands narayan$
```

Terminal showing 4 execution steps indicated by "+"
1. Space allocated in RAM for $x
2. Space allocated in RAM for $y
3. CPU fetches value of $x and $y from memory. Adds $x + $y and stores the value in $z
4. Display the value of $z

Selection Statements

These statements are used to change the flow of execution when a provided condition is True or False




If file 1 is available
TRUE
FALSE
Write “Hello” in the file
Print “File is not available”

IF Statement

If a condition provided is true, it will do a certain set of actions and if false another set of actions

```
If [Condition]
then
    Statements
fi
```

The statements will execute if the specified condition is True

IF-Else Statement

If a condition provided is true, it will enter the statements after **then**. If false, then statements inside **else** will be executed

```
If [Condition]
then
    Statements
else
    Statements
fi
```

Else IF Statement

- **If condition1 is True**, Statements inside its **then** will execute.
- **If condition1 is False**, then condition2 is checked.
  - If True, Statements inside it will be executed.
  - If False, the Statements inside else will execute.

```
if [ condition1 ]
then
  Statements
elif [ condition2 ]
then
  Statements
else
  Statements
fi
```

Nested If

- **If condition1 is True**, statements inside its **then** will execute.
- **If condition1 is False**, then it goes inside else. Now it checks the **condition2.**
- **If condition2 is True**, statements inside the second if statement is executed.
- **If condition2 is False**, statements inside the second else executes.

```
if [ condition1 ]
then
  Statements
else
then
  If [ condition2 ]
      Statements
  else
      then
          Statements
fi
```

Looping/Repetition Statements

Types of loops

While Loop

If the given command is TRUE, loop executes. If FALSE, comes out of loop

Until Loop

Same as while, but it will loop until the test case becomes true

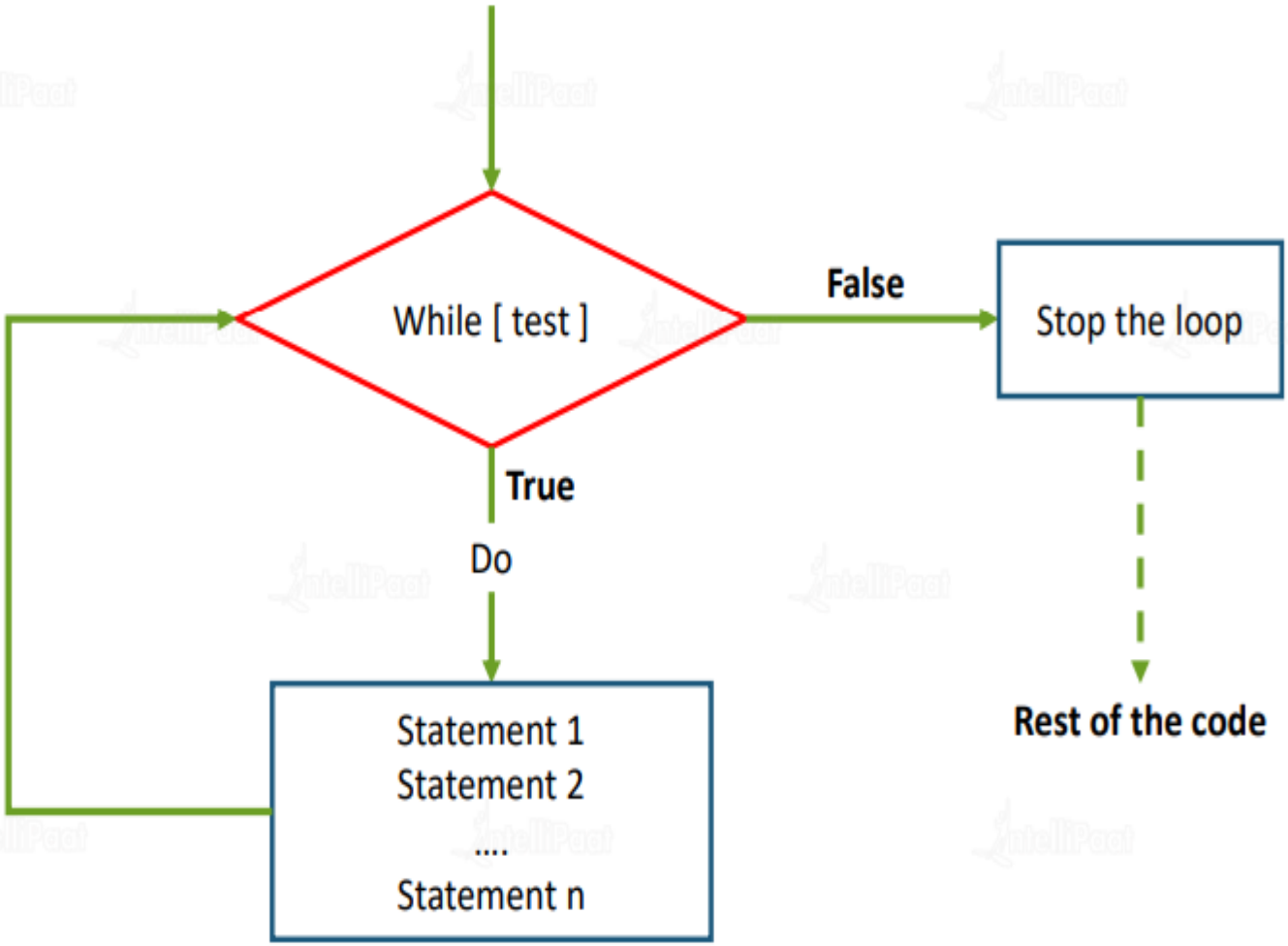
For Loop

It uses a given set of data to iterate until the given command is FALSE

While Loop

It is simple. When the command is true, it keeps executing the statements

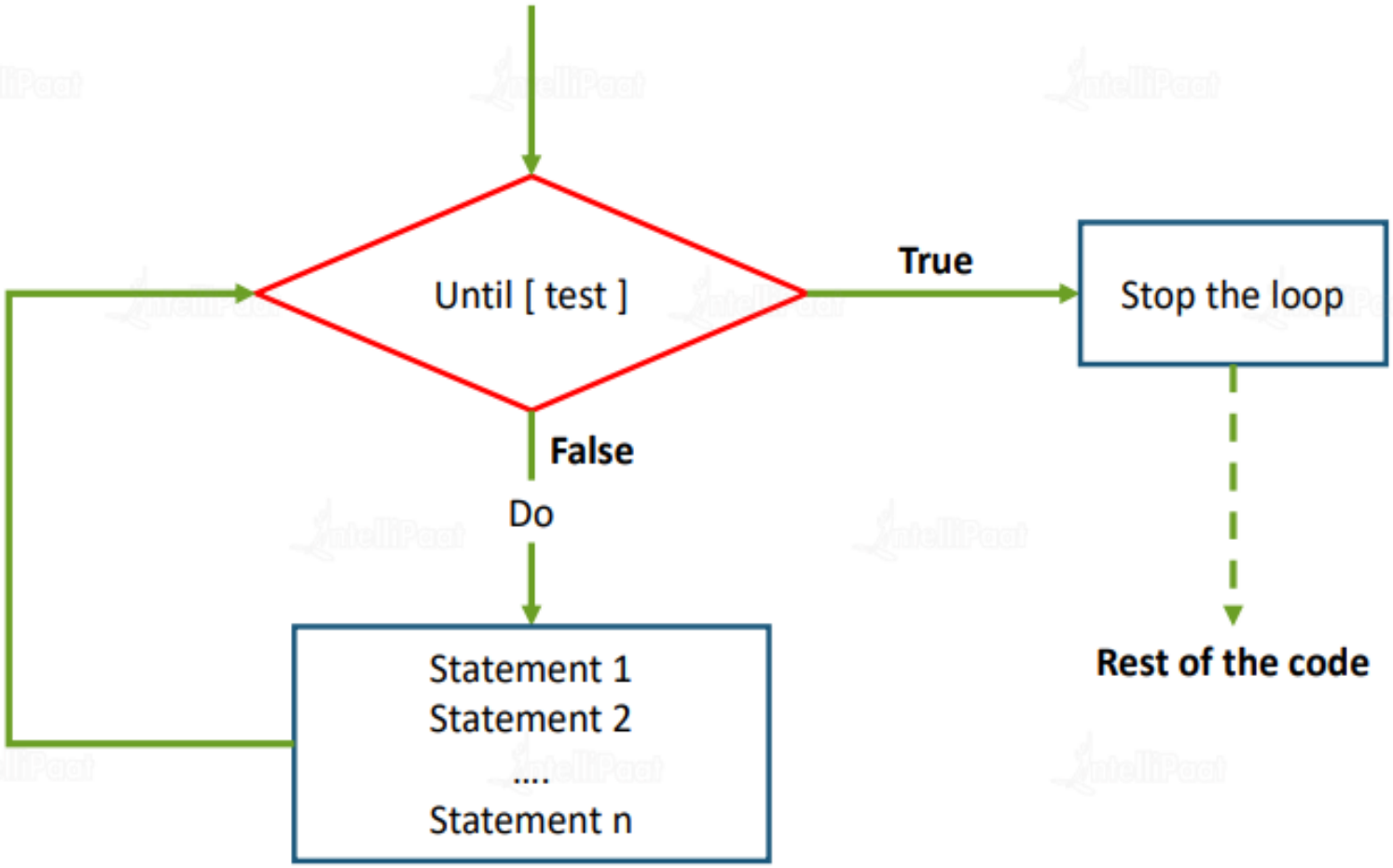
```
while [ command ]
do
    Statements
done
```


While [test]
False
Stop the loop
True
Do
Statement 1
Statement 2
....
Statement n
Rest of the code

Until Loop

It keeps executing the statements until the command becomes True

```
until [ test case ]
do
    Statements
done
```


Until [test]
True
Stop the loop
False
Do
Statement 1
Statement 2
....
Statement n
Rest of the code

For Loop

According to the given list, it executes the commands for each item

```
for variable in <list>
do
    Statements
done
```

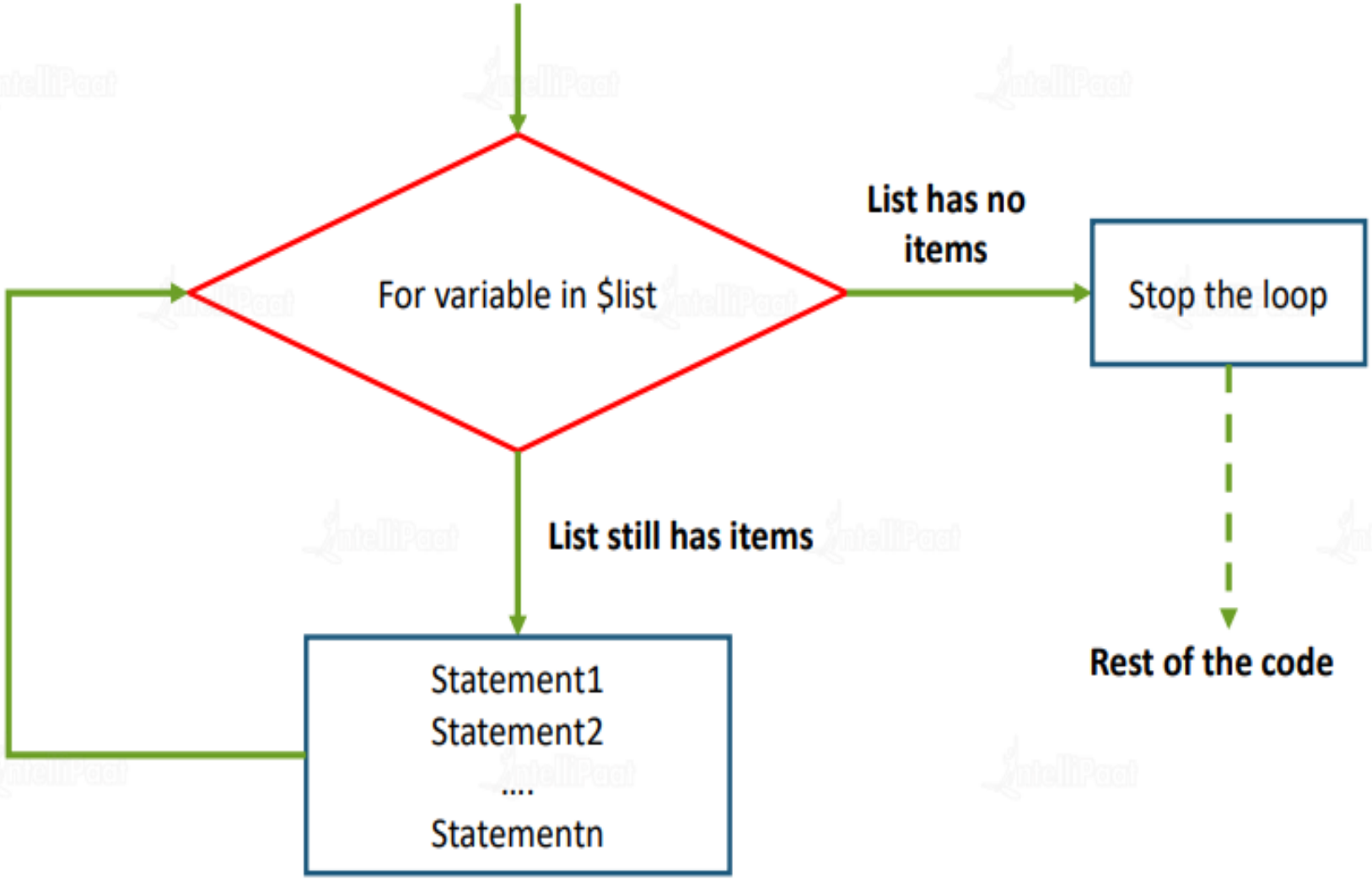
For Loop

The for loop operates on lists of items. It repeats a set of commands for every item in a list.

Syntax :

```
for var in word1 word2 ... wordN
do
  Statement(s) to be executed for every word.
done
```

Here var is the name of a variable and word1 to wordN are sequences of characters separated by spaces (words). Each time the for loop executes, the value of the variable var is set to the next word in the list of words, word1 to wordN.


For variable in $list
List has no items
Stop the loop
List still has items
Statement1
Statement2
....
Statementn
Rest of the code

Flow control statements

Break

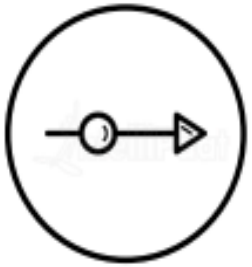
Tells Bash to leave the loop whenever it encounters a Break statement

Continue

Tell Bash to stop the current iteration and start a new iteration altogether

Break

```
while [ command ]
do
    If [ command ]
    then
        break
    fi
done
```



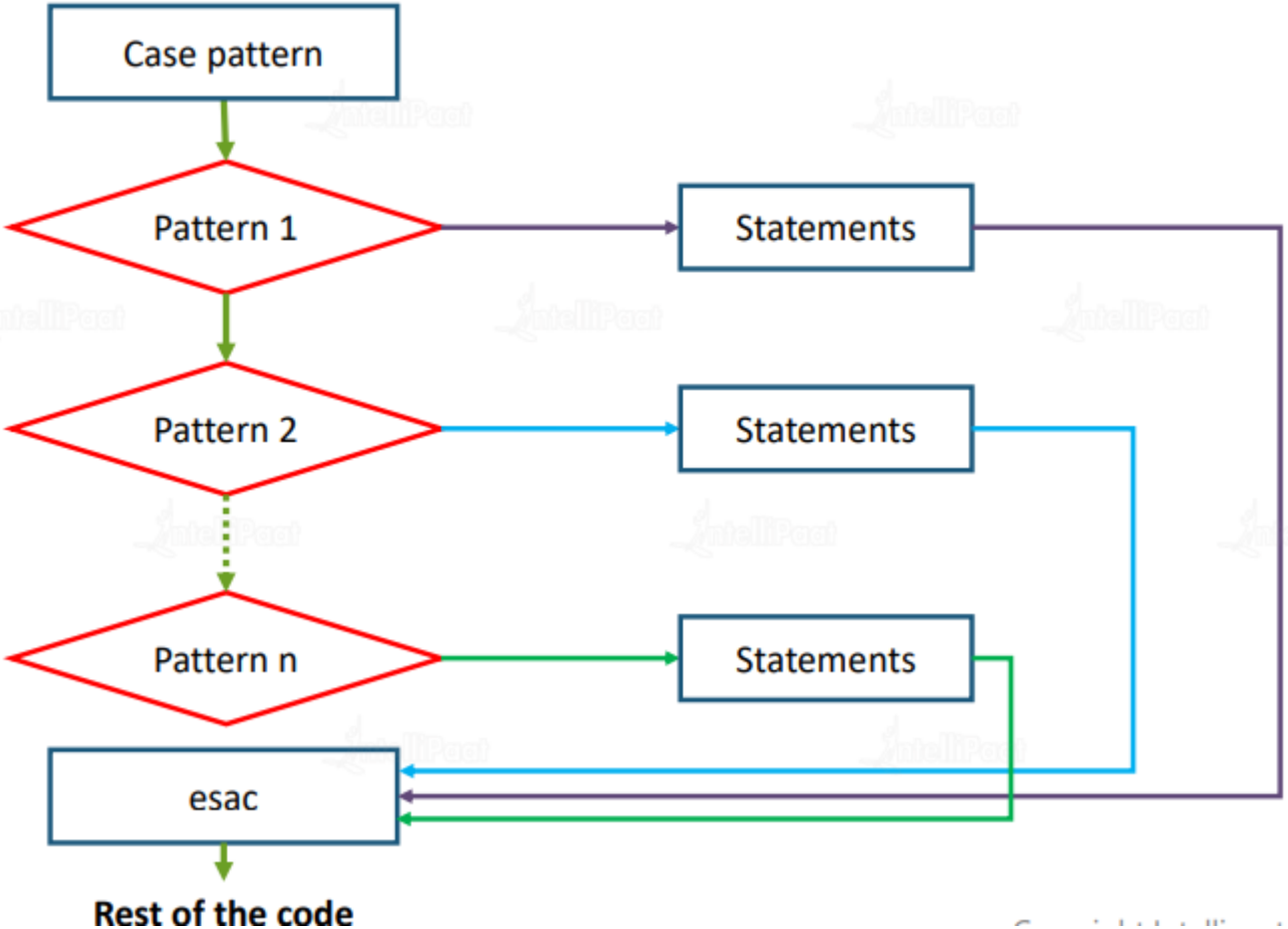
Continue

```
while [ command ]
do
    If [ command ]
    then
        continue
    fi
done
```

case…esac Statement

Use Switch statement to choose between multiple options and execute a set of statements under the selected option

```
case word in
        pattern1)
                Statement(s)
                ;;
        pattern2)
                Statement(s)
                ;;
        pattern3)
                Statement(s)
                ;;
        *)
                Default Statement(s)
                ;;
esac
```


Case pattern
Pattern 1
Statements
Pattern 2
Statements
Pattern n
Statements
esac
Rest of the code