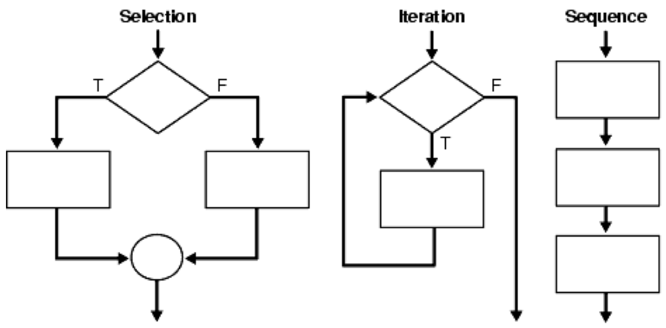
Conditional Branching

Objectives:

- Boolean Logics
- Boolean Evaluation on Objects
- Selection statements

What is Control Flow?

Instead of running statements in top-down sequence, you can change the flow of your program.



Python provides following control flow statements:

- `if` statement
- `for` loop
- `while` loop

1. Boolean Logics

1.1 Comparing Operators

Comparison operators are used to compare values. It returns either `True` or `False` according to the condition.

`<` `<=` `>` `>=` `==` `!=`

In [2]:
```
5 > 4
'Hi' == 'hi'
```
Out[2]:
```
False
```

1.2 Boolean Operators

Boolean operators are used to connect Boolean expressions (and objects) to create compound Boolean expressions.

Python has three Boolean operators, which are plain English words: `and` , `or` and `not` .

Operator	Meaning	Example
and	True if both operands are True	x and y
or	True if either operands is True	x or y
not	True if operand is False	not x

In [1]:
```python
x = 'yes'
x =='YES' or x=='yes'
```

Out[1]:
```
True
```

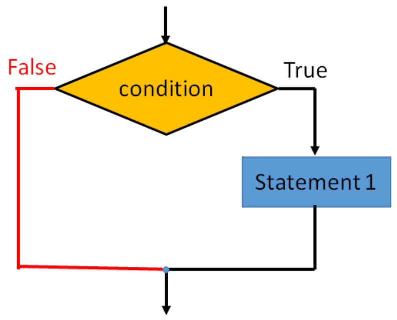
2. Select Statements

To choose statements to execute depending on several mutually exclusive conditions, Python provides **`if ... elif ... else`** construct:

- The `elif` and `else` clauses are optional

```
if <condition>:
    <statement>
elif <condition>:
    <statement>
else:
    <statement>
```

2.1 `if` Statement



Example:

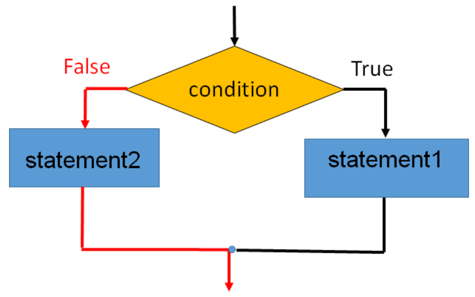
Check if age is greater than or equals to 18.

In [3]:
```python
age = 18

print(f'Your age is {age}')
if age >= 18:
    print('You are an adult')
```

```
Your age is 18
You are an adult
```

2.2 `if ... else` Statement



Exercise: Ask user to input an integer. Check if a number is even or odd.

Sample Output:

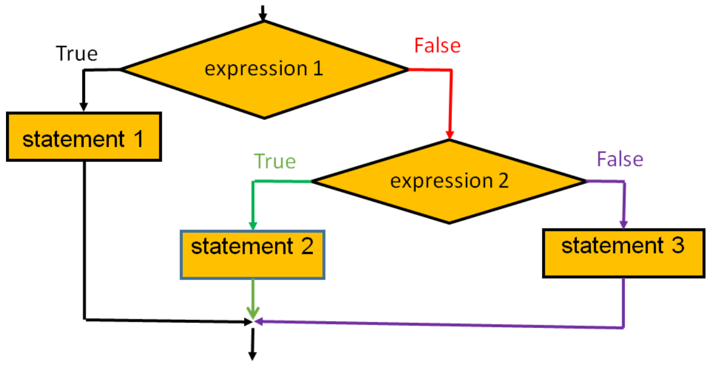
```
Enter an integer: 10
Even Number
Enter an integer: 9
Odd Number
```

In [4]:
```python
num = input('Enter an integer: ')
num = int(num)

if num % 2 == 0:
    print('Even')
else:
    print('Odd')
```

```
Enter an integer: 18
Even Number
```

2.3 `if ... elif ... else` Statement



Nested `if...else` statements can be used to implement above flow.

- But it may be cumbersome when there are more than 2 conditions.

Exercise:

Ask user to input 2 integers, `x` and `y`, check whether they are greater, less than or equal.

Print out either `x > y`, `x < y` or `x = y`.

In [ ]:
```
x = int(input('X: '))
y = int(input('Y: '))

if x > y:
    print('x > y')
elif x < y:
    print('x < y')
else:
    print('x = y')
```

3.5 Conditional Operator

Python also provide a conditional operator or ternary operation:

```
<statement_if_true> if <condition> else <statement_if_false>
```

In [1]:
```
score = 60
grade = 'passed' if score >= 50 else 'failed'

print('You have {}.'.format(grade))
```

```
You have passed.
```

Exercise:

Use conditional operator to check if a number is even number. For example, x = 10

In [24]:
```
x = 10

result = 'Even' if x % 2 == 0 else 'Odd'
print('{} is {}'.format(x, result))
```

```
10 is Even
```

3. Faulsy Values

An object can also be evaluated to be `True` or `False` using built-in `bool()` function.

Faulsy Values are values which will be evaluated as `False` .

- Constants defined to be false: `None` and `False`
- Zero of any numeric type: `0` , `0.0` , `0j`
- Empty sequences and collections: `''` , `()` , `[]` , `{}` , `set()`

Zero Numeric Values

Non-zero values are evaluated to `True` .

Exercise: What's the boolean values of `2` , `-100` , `0.01` and `0.0` ?

In [2]:
```
print(bool(2), bool(-100), bool(0.01), bool(0.0))
```

```
True True True False
```

Empty Strings

Non-empty strings are evaluated to `True` .

Exercise: What's the boolean values of `'Hi'` , `''` , and `' '` ?

In [8]:
```
print(bool('Hi'))
print(bool(''))  # empty string
print(bool(' ')) # with one space
```

```
True
False
True
```

Empty Collections ¶

Empty collections are evaluated as `False` .

Exercise: What's the boolean values of `[]` , and `[0, 0]` ?

In [9]:
```
print(bool([]), bool([0, 0]))
```

```
False True
```