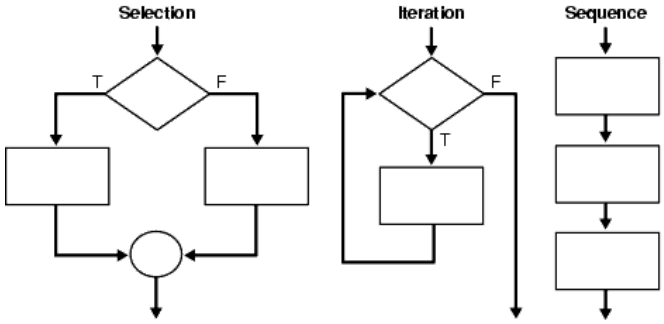
Control Flow Statements

- Boolean Logics
- Boolean Evaluation on Objects
- Selection statements
- For-Loops
- While-loops

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. Recap on Boolean Logics

Let's recap some basic concepts related to Boolean logic in Python.

- **Boolean** is a data type can be either `True` or `False` .
- **Boolean Values** are values `True` or `False` .
- **Boolean Variables** are variables of **Boolean** data type.
- **Boolean Expression** are expressions which can be evaluated to be either `True` or `False` .

1.1 Comparing Operators

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

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

Try Code:

```
5 > 4
'Hi' == 'hi'
```

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

Try Code:

```
x = 'yes'
x =='Yes' or x=='yes'
```

In [6]:
```
x = 'yes'
```

Out[6]:
```
False
```

2. Evaluating Objects

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

- By default, an non-None object is considered `True` (unless its `__bool__()` method returns `False` .

Following are the cases which will be evaluated `False` .

- Constants defined to be false
  - `None` and `False`
- Zero of any numeric type
  - `0` , `0.0` , `0j` , `Decimal(0)` , `Fraction(0, 1)`
- Empty sequences and collections:
  - `''` , `()` , `[]` , `{}` , `set()` , `range(0)`

2.1 Non-Zero Numeric Values

Non-zero values are evaluated to `True` .

Try Code:

```
bool(2)
bool(-100)
bool(0.01)
bool(0.0)
```

In [7]:
```
print(bool(2))
print(bool(-100))
print(bool(0.01))
print(bool(0.0))
```

```
True
True
True
False
```

2.2 Non-Empty Strings

Non-empty strings are evaluated to `True` .

Try Code:

```
bool('Hi')
bool('')  # empty string
bool(' ') # one space
```

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

```
True
False
True
```

2.3 Empty Collections

Empty collections are evaluated as `False` .

Try Code:

```
bool([])
bool([0, 0])
```

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

```
False
True
```

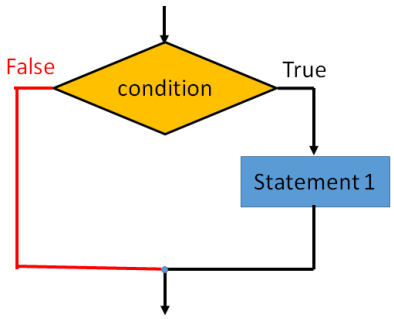
3. Select Statements

3.1 `if` Statement

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>
```



Try Code:

```
weather = 'sunny'

if weather == 'sunny':
    print('Outdoor fun!')

print('Enjoy your day')
```

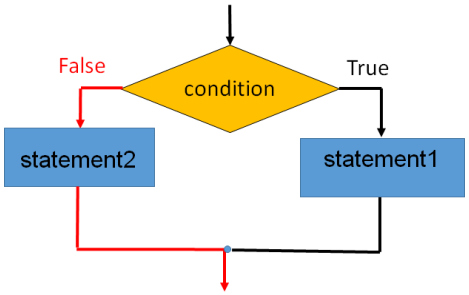
In [11]:
```
weather = 'raining'

if weather == 'sunny':
    print('Outdoor fun!')

print('Enjoy your day')
```

```
Enjoy your day
```

3.2 if ... else Statement



Question:

Ask user to input his/her age. Print out message depends on whether age is above or equals 18.

Sample Output:

```
How old are you? 19
You are an adult. Yeah.
How old are you? 17
You are still young.
```

In [12]:
```
age = input('How old are you? ')
age = int(age)

if age >= 18:
    print('You are an adult. Yeah.')
else:
    print('You are still young.')
```

```
How old are you? 17
You are still young.
```

Question:

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

Sample Output:

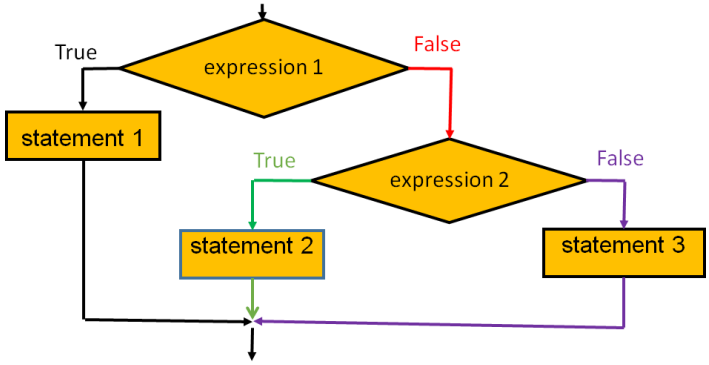
```
Enter an integer: 10
Even Number
Enter an integer: 9
Odd Number
```

In [13]:
```
num = input('Enter an integer: ')
num = int(num)

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

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

3.3 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`.

Use nested `if..else` in your code.

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

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

In this case, `if...elif...else` offers a simpler syntax.

- There can be more than 1 `elif`.

Above example can be re-writen as following:

Exercise:

Write your code in previous Exercise using `if..elif..else` .

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')
```

3.4 Chained Operators

Comparison operators can be chained together.

Try Code:

```
x = 10; y = 20; z = 30
y > x and y <= z
```

In [14]:
```
x = 10; y = 20; z = 30
y > x and y <= z
```

Out[14]:
```
True
```

Try Code:

```
x < y <= z
```

In [15]:
```
x < y <= z
```

Out[15]:
```
True
```

Try Code:

```
x = 10; y = 20; z = 15
x < y >= z
```

In [19]:
```
x = 10; y = 20; z = 15
x < y >= z
```

Out[19]:
```
True
```

Example:

Implement a function `get_grade()` which returns grade based on input score.

- 'A' if >=80 and <=100
- 'B' if >=70 and <80
- 'C' if >=60 and <70
- 'D' if >=50 and <60
- 'E' otherwise

In [21]:
```
def get_grade(mark):
    if mark >= 80:
        return 'A'
    elif mark >=70:
        return 'B'
    elif mark >=60:
        return 'C'
    elif mark >=50:
        return 'D'
    else:
        return 'E'

mark = int(input('Enter your mark: '))
grade = get_grade(mark)
print(grade)
```

```
Enter your mark: 75
B
```

3.5 One-line Statements

It is also possible to put indented statements right after `if` , `elif` and `else` .

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

Example:

Above programe can be writen as following:

```
perscore = int(input('Enter your score (1-100):'))

if 80 <= perscore <= 100: grade = 'A'
elif 70 <= perscore < 80: grade = 'B'
elif 60 <= perscore < 70: grade = 'C'
elif 50 <= perscore < 60: grade = 'D'
else: grade = 'F'

print('Grade = ', grade)
```

3.6 Conditional Operator

Python also provide a conditional operator or ternary operation:

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

Try Code:

```
score = 60
grade = 'passed' if score >= 50 else 'failed'

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

In [23]:
```
score = 60

grade = 'passed' if score >= 50 else 'failed'

# if score >= 50:
#     grade = 'passed'
# else:
#     grade = '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
```