# Basic control flow

# Topics

- Basic control flow
- If statement
- Repetition statements
- Comparison operators
- Pass statement

Digital Career Institute

# What is control flow?

Digital Career Institute

- A program's **control flow** is the order in which the program's code executes.

- The control flow of a Python program is regulated by **conditional statements**, **loops**, and **function calls**.

# Control structures

Python has *three* types of control structures:

- **Sequential** - default mode.
- **Selection** - used for decisions and branching.
- **Repetition** - used for looping, i.e., repeating a piece of code multiple times.

# Sequential statements

- **Sequential statements** are a set of statements whose execution process happens in a sequence.

- The problem with sequential statements is that if the logic has broken in any one of the lines, then the complete source code execution will break.

# Sequential statement - example

Digital Career Institute

```
1  # This is a sequential statement
2
3  a = 20
4  b = 10
5  c = a - b
6  print("Result of subtraction is : ", c)
```

# Selection/decision control statements

Digital Career Institute

- The **selection statement** allows a program to test several conditions and execute instructions based on which condition is **true**.

- In Python, the selection statements are also known as **decision control statements** or **branching statements**.

# Selection/decision control statements

Digital Career Institute

Some Decision Control Statements are:

- Simple **if**
- **If-else**
- nested **if**
- **if-elif-else**

# **If** statements

- ***If statements*** are control flow statements that help us to run a particular code, but only when a certain condition is met or satisfied.

- A *simple* ***if*** only has one condition to check.

# Indentation

Digital Career Institute

- Python relies on indentation (whitespace at the beginning of a line) to define scope in the code.
- Preferred indentation is equal to **4 spaces**
- Other programming languages often use curly-brackets for this purpose.

# Indentation

- If statement, without indentation (will raise an error):

```
a = 33
b = 200
if b > a:
print("b is greater than a") # you will get an error
```

# Elif

- The **elif** keyword is pythons way of saying "if the previous conditions were not true, then try this condition":

```
a = 33

b = 33

if b > a:

    print("b is greater than a")  # indentation

elif a == b:

    print("a and b are equal")  # indentation
```

# Else

- The **else** keyword catches anything which isn't caught by the preceding conditions:

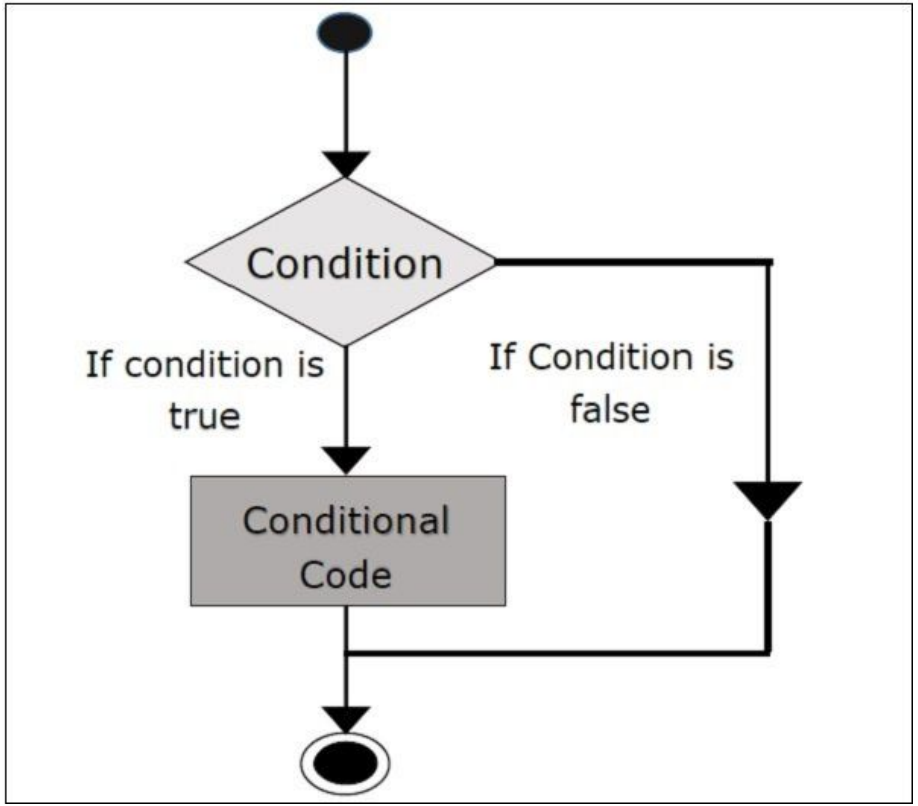
```
a = 200
b = 33
if b > a:
    print("b is greater than a")
elif a == b:
    print("a and b are equal")
else:
    print("a is greater than b")
```

# “Short” if

- If you have **only one** statement to execute, you can put it on the same line as the if statement:

```
if a > b: print("a is greater than b")
```

# Simple **if**


Condition
If condition is true
If Condition is false
Conditional Code

# Simple if - example

Digital Career Institute

```
1  n = 10
2  if n % 2 == 0:
3      print("n is an even number!")
```

# If -else

- The **if-else** *statement* evaluates the condition and will execute the body of **if**, if the test condition is True.

- But if the condition is **False**, then the body of else is executed.

# If -else (algorithm)

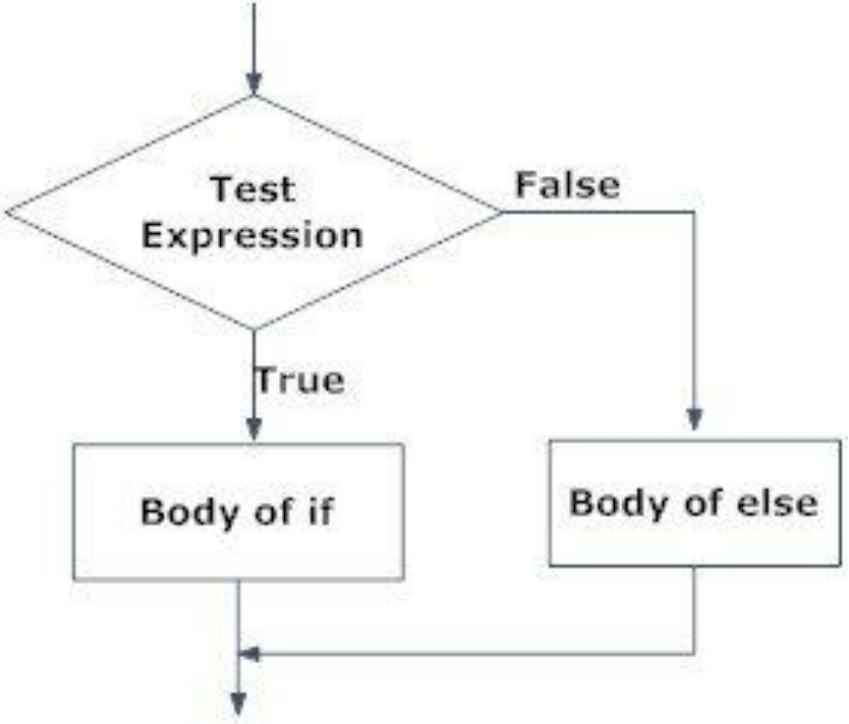
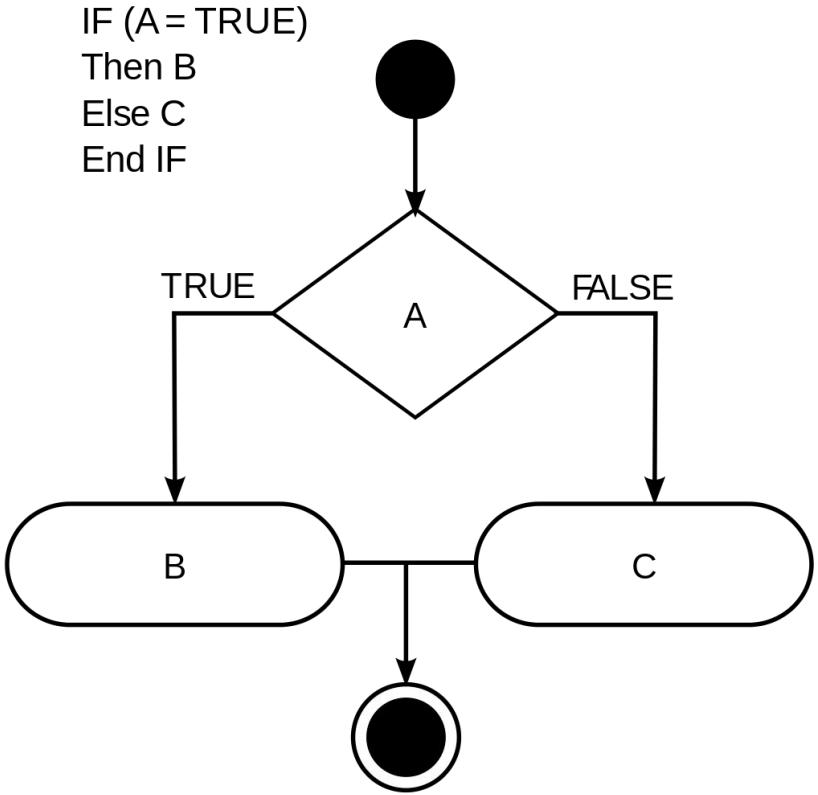



Fig: Operation of if...else statement

# If -else (algorithm ver. 2)



# If -else (code example)

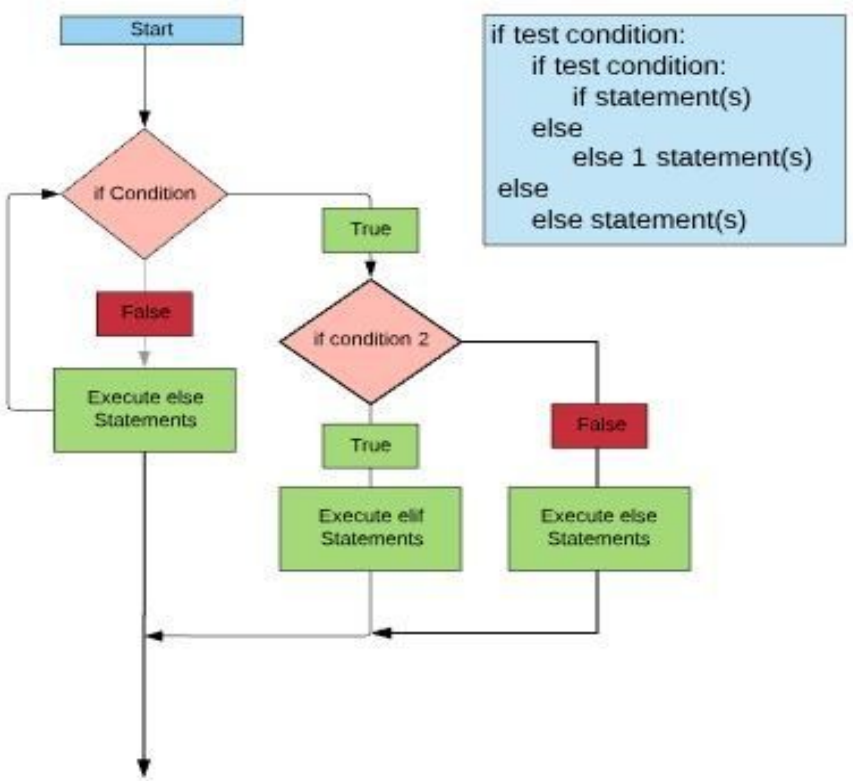
Digital Career Institute

```
n = 5
if n % 2 == 0:
    print("n is even!")
else:
    print("n is odd!")
```

# Nested If

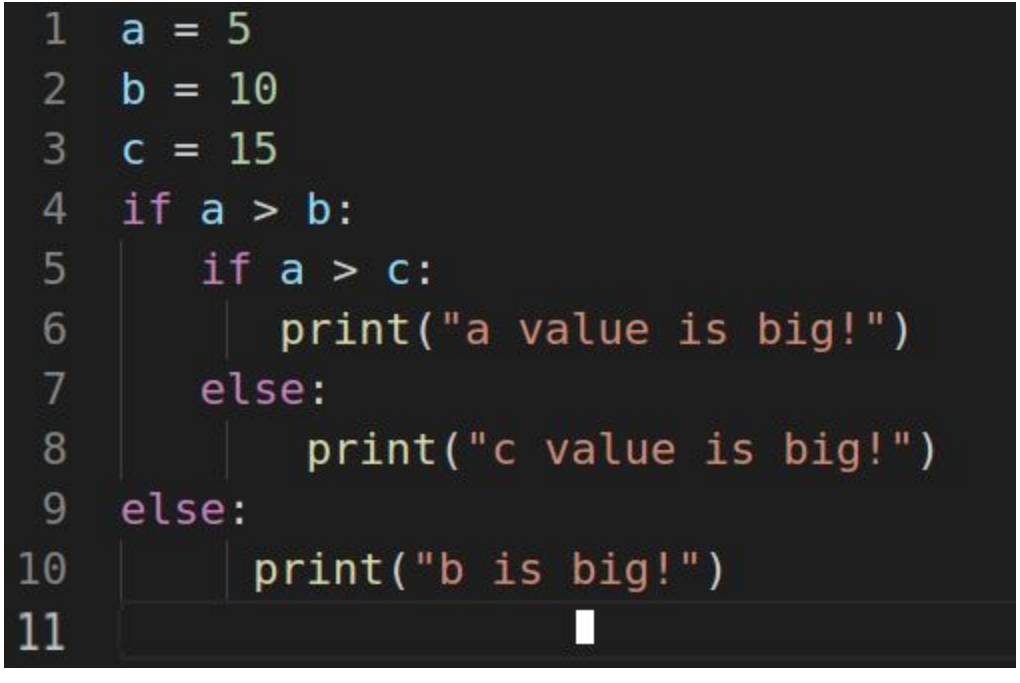
- **Nested** if st*atements* are an if statement **inside** another **if** statement.

# Nested If




```
if test condition:
    if test condition:
        if statement(s)
    else
        else 1 statement(s)
else
    else statement(s)
```

# Nested If


1 a = 5
2 b = 10
3 c = 15
4 if a > b:
5 if a > c:
6 print("a value is big!")
7 else:
8 print("c value is big!")
9 else:
10 print("b is big!")
11

# If-elif-else

- **if-elif-else:** The *if-elif-else statement* is used to conditionally execute a statement or a block of statements.

# If-elif-else

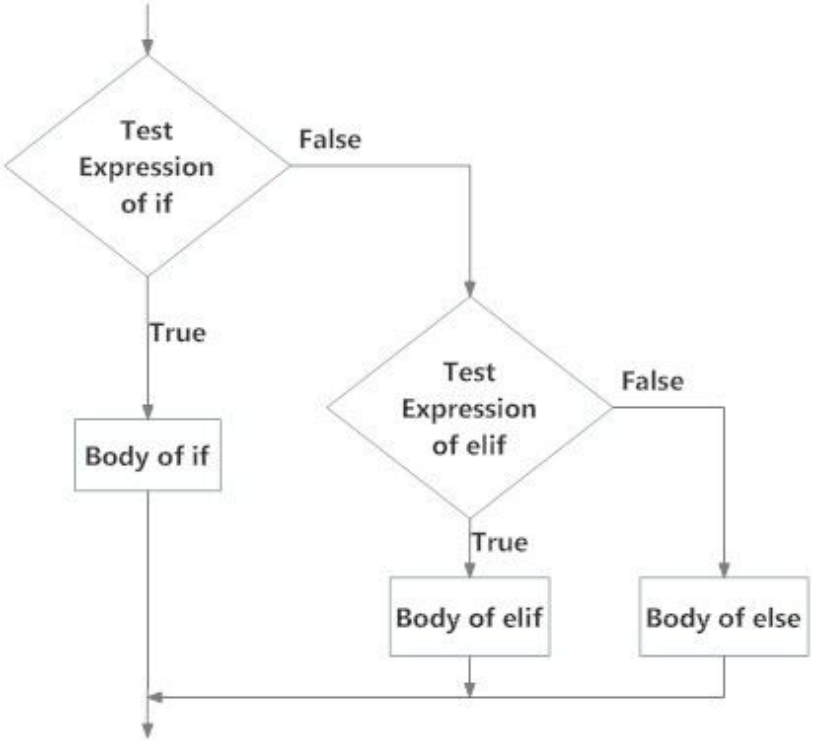



Fig: Operation of if...elif...else statement

# If-elif-else

Digital Career Institute

```
1  x = 15
2  y = 12
3
4  if x == y:
5      print("Both are equal!")
6  elif x > y:
7      print("x is greater than y")
8  else:
9      print("x is smaller than y")
10
```

# Python comparison operators

Digital Career Institute

- Comparison operators are used to **compare** two values.
- The result is always **True** or **False**!

# Python comparison operators

Digital Career Institute

| Operator | Name | Example |
|---|---|---|
| == | Equal | x == y |
| != | Not equal | x != y |
| > | Greater than | x > y |
| < | Less than | x < y |
| >= | Greater than or equal to | x >= y |
| <= | Less than or equal to | x <= y |

# Python comparison operators - example

Digital Career Institute

```
x = 5
y = 3

print(x > y)

# returns True because 5 is greater than 3
```

# Repetition statements

# Repetition statement

Digital Career Institute

- A **repetition statement** is used to repeat a group (block) of programming instructions.

- In Python, we generally have two loops/repetitive statements:
  - **for** loop
  - **while** loop

- A **for** and **while** loops will be covered later in more detail!

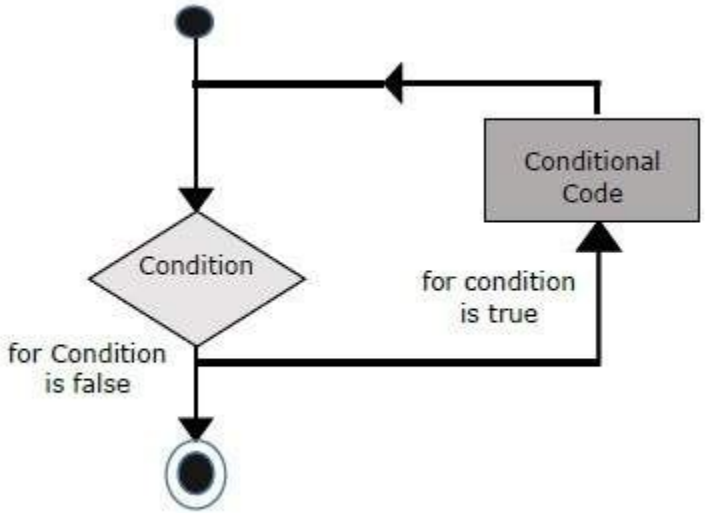
# For loop

Digital Career Institute

- A **for loop** is used to iterate over a sequence that is either a list, tuple, dictionary, or a set (will be covered **later**!).

- We can execute a set of statements once for each item in a sentence.

# For loop


Conditional
Code
Condition
for condition
is true
for Condition
is false

# range() function

- To loop through a set of code a specified number of times, we can use the range() function,

- With **one** argument, e.g. **range(5)**, function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and ends at a specified number (but without this number).

```
for x in range(5):
    print(x)
# prints integers 0, 1, 2, 3, 4
# without number 5
```

# range() function

- It is possible to specify the starting value by adding a parameter: **range(2, 6)**, which means values from 2 to 6 (but not including 6):

```
for x in range(2, 6):
    print(x)
# prints integers 2, 3, 4, 5
# without number 6
```

# range() function

- The range() function defaults to increment the sequence by 1, however it is possible to specify the increment value by adding a **third** parameter:

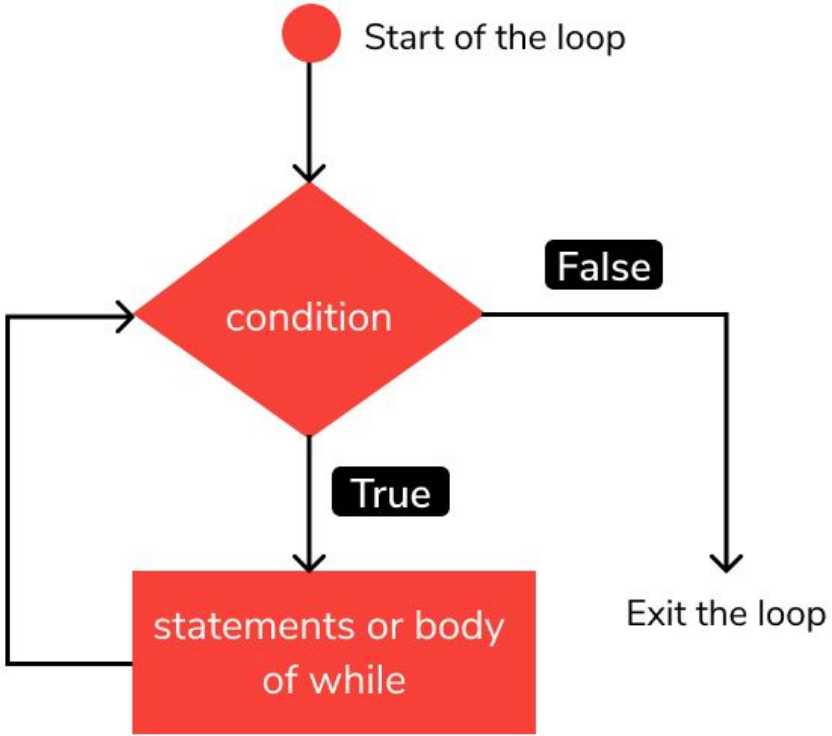
```
for x in range(2, 15, 3):
    print(x)
# prints integers 2, 5, 8, 11, 14
```

# While loop

Digital Career Institute

- In Python, **while loops** are used to execute a block of statements repeatedly until a given condition is satisfied.
- Then, the expression is checked again and, if it is **still true**, the body is executed again.
- This continues until the expression becomes false.

# While loop


Start of the loop
condition
False
True
statements or body of while
Exit the loop

# While loop

```
i = 1
while i < 6:
    print(i)
    i += 1
# prints 1, 2, 3, 4, 5
```

- Remember to increment i, or else the loop will continue **forever**.
- The while loop requires relevant variables to be ready, in this example we need to define an indexing variable **i**, which we set to 1.

# **else** in **for** loop

Digital Career Institute

- The **else** keyword in a **for** loop specifies a block of code to be executed when the loop is finished:

```
for x in range(3):
    print(x)
else:
    print("Finally finished!")
# prints 0, 1, 2, "Finally finished!"
```

# Nested loops

Digital Career Institute

- A nested loop is a loop inside a loop.
- The **"inner loop"** will be executed one time for each iteration of the "**outer loop**":

```
for x in range(10, 21, 10):  # x will be 10, and then 20 in second iteration
    for y in range(2):       # y will 0, and then 1 in second iteration
        print(x + y)
# prints 10, 11, 20, 21
```

# Pass statement

- The **pass** statement is used as a placeholder for future code.

- When the **pass** statement is executed, nothing happens, but you avoid getting an **error** when empty code is not allowed.

- Empty code **is not allowed** in loops, function definitions, class definitions, or in if statements.

# Pass statement

- for loops **cannot** be empty, but if you for some reason have a for loop with no content, put in the **pass** statement to avoid getting an error.

```
for x in range(10):
    pass
# # having an empty for loop like this,
# would raise an error without the pass statement
```

# At the core of the lesson

Lesson learned:

- We know basic control flow rules
- We know comparison operators
- We know repetition statements

Digital Career Institute

# Documentation

# Documentation

1. Python tutorial on W3 Schools
2. Control-flow Documentation
3. Comparison operators
4. Repetition statements


**Contact Details**
**DCI Digital Career Institute gGmbH**

Digital Career Institute