Python Cheat sheet

Intro to Programming with python

Software:

Software is a set of instructions for the hardware.

Programming:

Programming means writing the instructions to create software.

Code:

The instructions that we write to create software are called codes.

Syntax:

Similar to Grammar rules in English, and Hindi, each programming language has a unique set of rules. These rules are called the Syntax of a Programming Language.

Why Python

Python is an easy-to-learn, powerful programming language. With Python, it is possible to create programs with a minimal amount of code. Look at the code in Java and Python used for printing the message "Hello World"

Java:

```
class Main {
  public static void main(String[] args) {
    System.out.println("Hello World");
  }
}
```

JAVA

Python:

```
print("Hello World")
```

Applications of Python

Python is a versatile language which has applications in almost every field

- Artificial intelligence (AI)
- Machine Learning (ML)

- Big Data
- Smart Devices/Internet of Things (IoT)
- Cyber Security
- Game Development
- Backend Development, etc.

Career Opportunities:

Python developers have plenty of opportunities across the world

- DevOps Engineer
- Software Developer
- Data Analyst
- Data Scientist
- Machine Learning (ML) Engineer
- AI Scientist, etc.

Hello World Program in Python

Here is a simple Python code that you can use to display the message **"Hello World"**

Code:

```
print("Hello World!")
```

Output:

```
Hello World!
```

Possible Mistakes:

Possible mistakes you may make while writing Python code to display the message "Hello World"

Spelling Mistake in print

```
print("Hello World!")
```

Uppercase 'P' in Print

```
Print("Hello World!")
```

Missing quotes

```
print(Hello World!)
```

Missing parentheses

```
print("Hello World!"
```

Printing Without Quotes

If we want to print the numerical result of 2 + 5, we do not add quotes.

Code

```
print(2 + 5)
```

Output

```
7
```

If we want to print the exact message "2 + 5", then we add the quotes.

Code

```
print("2 + 5")
```

Output

```
2 + 5
```

Calculations with python

Addition

The addition is denoted by + sign. It gives the sum of two numbers.

Code

```
print(2 + 5)

print(1 + 1.5)
```

Output

```
7

2.5
```

Subtraction

Subtraction is denoted by the ‘-‘ sign. It gives the difference between two numbers.

Code:

```
print(5 - 2)
```

Output

```
3
```

Multiplication

Multiplication is denoted by the * sign.

Code

```
print(2 * 5)

print(5 * 0.5)
```

Output

```
10

2.5
```

Division

The division is denoted by / sign.

Code:

```
print(5 / 2)

print(4/2)
```

Output:

```
2.5

2.0
```

Variables and Data Types

Variables

Variables are like containers for storing values. Values in the variables can be changed.

Values

Consider that variables are like containers for storing information. In context of programming, this information is often referred to as value.

Data Type

In programming languages, every value or data has an associated type to it known as data type. Some commonly used data types

- String
- Integer
- Float
- Boolean

This data type determines how the value or data can be used in the program. For example, mathematical operations can be done on Integer and Float types of data.

String

A String is a stream of characters enclosed within quotes. Stream of Characters

- Capital Letters (A – Z)
- Small Letters (a – z)
- Digits (0 – 9)
- Special Characters (~ ! @ # $ % ^ . ?,)
- Space

Some Examples

- "Hello World!"
- "some@example.com"
- "1234"

Integer

All whole numbers (positive, negative and zero) without any fractional part come under Integers. Examples

...-3, -2, -1, 0, 1, 2, 3,...

Float

Any number with a decimal point

24.3, 345.210, -321.86

Boolean

In a general sense, anything that can take one of two possible values is considered a Boolean. Examples include the data that can take values like

- True or False
- Yes or No
- 0 or 1
- On or Off , etc.

As per the Python Syntax, **True** and **False** are considered as Boolean values. Notice that both start with a capital letter.

Assigning Value to Variable

The following is the syntax for assigning an integer value 10 to a variable age

age = 10

Here the equals to **=** sign is called as **Assignment Operator** as it is used to assign values to variables.

Sequence of Instructions

Program

A program is a sequence of instructions given to a computer.

Defining a Variable

A variable gets created when you assign a value to it for the first time.

Code

```
age = 10
```

Printing Value in a Variable

Code

```
age = 10
print(age)
```

Output

```
10
```

Code:

```
age = 10
print("age")
```

Output

```
age
```

Variable name enclosed in quotes will print variable rather than the value in it. If you intend to print value, do not enclose the variable in quotes.

Order of Instructions

Python executes the code line-by-line.

Code

```
print(age)
age = 10
```

Output

NameError: name **'age'** is not defined

Variable **Age** is not created by the time we tried to print.

Spacing in Python

Having spaces at the beginning of line causes errors.

Code

```
1 a = 10 * 5
2 b = 5 * 0.5
3 b = a + b
```

Output

```
File "main.py", line 3
   b = a + b
   ^
IndentationError: unexpected indent
```

Variable Assignment

Values in the variables can be changed.

Code

```
a = 1
print(a)
a = 2
print(a)
```

Output

```
1
2
```

Examples of Variable Assignment

Code

```
a = 2
print(a)
a = a + 1
print(a)
```

Output

```
2
3
```

Code

```
a = 1
b = 2
a = b + 1
print(a)
print(b)
```

Output

```
3
2
```

Expression

An expression is a valid combination of values, variables and operators.

Examples a * b a + 2 5 * 2 + 3 * 4

BODMAS The standard order of evaluating an expression

- Brackets (B)
- Orders (O)
- Division (D)
- Multiplication (M)
- Addition (A)
- Subtraction (S)

Step by Step Explanation

(5 * 2) + (3 * 4)

(10) + (12)

22

Code

```
print(10 / 2 + 3)
print(10 / (2 + 3))
```

Output

```
8.0
2.0
```

Inputs and Outputs Basics

Take Input From User **input()** allows flexibility to take the input from the user. Reads a line of input as a string.

Code

```
username = input()
print(username)
```

Input

```
Ajay
```

Output

```
Ajay
```

Working with Strings:

String Concatenation:

Joining strings together is called string concatenation.

Code

```
a = "Hello" + " " + "World"
print(a)
```

Output

```
Hello World
```

Concatenation Errors

String Concatenation is possible only with strings.

Code

```
a = "*" + 10
print(a)
```

Output:

```
File "main.py", line 1
   a = "*" + 10
   ^
TypeError:
can only concatenate str (not "int") to str
```

String Repetition * operator is used for repeating strings any number of times as required.

Code

```
a = "*" * 10
print(a)
```

Output

```
**********
```

Code

```
s = "Python”
s = ("* " * 3) + s + (" *" * 3)
print(s)
```

Output

```
* * * Python * * *
```

Length of String len() returns the number of characters in a given string.

Code

```
username = input()
length = len(username)
print(length)
```

Input

```
Ravi
```

Output

```
4
```

String Indexing

We can access an individual character in a string using their positions (which start from 0) . These positions are also called as index.

Code

```
username = "Ravi"
first_letter = username[0]
print(first_letter)
```

Output

```
R
```

IndexError: Attempting to use an index that is too large will result in an error:

Code

```
username = "Ravi"
print(username[4])
```

Output

```
IndexError: string index out of range
```

Type Conversion

String Slicing

Obtaining a part of a string is called string slicing.

Code

```
variable_name[start_index:end_index]
```

Start from the **start_index** and stops at **end_index**

end_index is not included in the slice.

Code:

```
message = "Hi Ravi"
part = message[3:7]
print(part)
```

Output

```
Ravi
```

Slicing to End

If end index is not specified, slicing stops at the end of the string.

Code

```
message = "Hi Ravi"
part = message[3:]
print(part)
```

Output

```
Ravi
```

Slicing from Start

If start index is not specified, slicing starts from the index 0.

Code

```
message = "Hi Ravi"

part = message[:2]

print(part)
```

Output

```
Hi
```

Checking Data Type:

Check the datatype of the variable or value using **type()**

Printing Data Type

Code

```
print(type(10))

print(type(4.2))

print(type("Hi"))
```

Output

```
<class 'int'>

<class 'float'>

<class 'str'>
```

Type Conversion

Converting the value of one data type to another data type is called **Type Conversion or Type Casting**. We can convert

- String to Integer
- Integer to Float
- Float to String and so on.

String to Integer **int()** converts valid data of any type to integer

Code

```
a = "5"

a = int(a)

print(type(a))

print(a)
```

Output

```
<class 'int'>

5
```

Invalid Integer Conversion

Code

```
a = "Five"

a = int(a)

print(type(a))
```

Output

ValueError: invalid literal for int() with base 10: 'Five'

Code

```
a = "5.0"

a = int(a)

print(type(a))
```

Output

```
invalid literal for int() with base 10: '5.0'
```

Adding Two Numbers

Code

```
a = input()

a = int(a)

b = input()

b = int(b)

result = a + b

print(result)
```

Input

```
2

3
```

Output

```
5
```

Integer to String **str()** converts data of any type to a string.

Code:

```
a = input()

a = int(a)

b = input()
```

```
b = int(b)

result = a + b

print("Sum: " + str(result))
```

Input:

```
2

3
```

Output:

```
Sum: 5
```

Summary

1. int()-> Converts to integer data type
2. float()-> Converts to float data type
3. str()-> Converts to string data type
4. bool() -> Converts to boolean data type

Relational Operators

Relational Operators are used to compare values.
Gives **True** or **False** as the result of a comparison.

These are different relational operators

Operator	Name
>	Is greater than
<	Is less than
==	Is equal to
<=	Is less than or equal to
>=	Is greater than or equal to
!=	Is not equal to

Code

```
print(5 < 10)

print(2 > 1)
```

Output

```
True

True
```

Possible Mistakes

Mistake - 1

Code

```
print(3 = 3)
```

Output

SyntaxError: expression cannot contain assignment, perhaps you meant "=="?

Mistake - 2

Code

```
print(2 < = 3)
```

Output

SyntaxError: invalid syntax

Space between relational operators

- ==
- >=
- <=
- !=

is not valid in Python.

Comparing Numbers

Code

```
print(2 <= 3)
print(2.53 >= 2.55)
```

Output

```
True
False
```

Comparing Integers and Floats

Code

```
print(12 == 12.0)
print(12 == 12.1)
```

Output

```
True
False
```

Comparing Strings

Code

```
print("ABC" == "ABC")
print("CBA" != "ABC")
```

Output

```
True
True
```

Case Sensitive

Code

```
print("ABC" == "abc")
```

Output

```
False
```

Python is case sensitive. It means X (Capital letter) and x (small letter) are not the same in Python.

Strings and Equality Operator

Code

```
print(True == "True")
print(123 == "123")
print(1.1 == "1.1")
```

Output

```
False
False
False
```

Logical Operators

The logical operators are used to perform logical operations on Boolean values.
Gives True or False as the result.

Following are the logical operators

- and
- or
- not

Logical AND Operator:

Gives True if both the booleans are true else, it gives False

Code

```
print(True and True)
```

Output

```
True
```

Examples of Logical AND

Code

```
print((2 < 3) and (1 < 2))
```

Step by Step Explanation

(2 < 3) and (1 < 2)

True and (1 < 2)

True and True

Output

```
True
```

Logical OR Operator

Gives True if any one of the booleans is true else, it gives False

Code

```
print(False or False)
```

Output

```
False
```

Examples of Logical OR

Code

```
print((2 < 3) or (2 < 1))
```

Step by Step Explanation

(2 < 3) or (2 < 1)

True or (2 < 1)

True or False

Output

```
True
```

Logical NOT Operator

Gives the opposite value of the given boolean.

Code

```python
print(not(False))
```

Output

```
True
```

Examples of Logical NOT

Code

```python
print(not(2 < 3))
```

Step by Step Explanation

```
not(2 < 3)

not(True)

False
```

Output

```
False
```

Summary

1. Logical AND Operator gives True if all the booleans are true.
2. Logical OR Operator gives True if any of the booleans are true.
3. Logical NOT Operator gives the opposite value

Conditional Statements:

Block of Code

A sequence of instructions are called block of code.
Python executes code in a sequence.

Condition

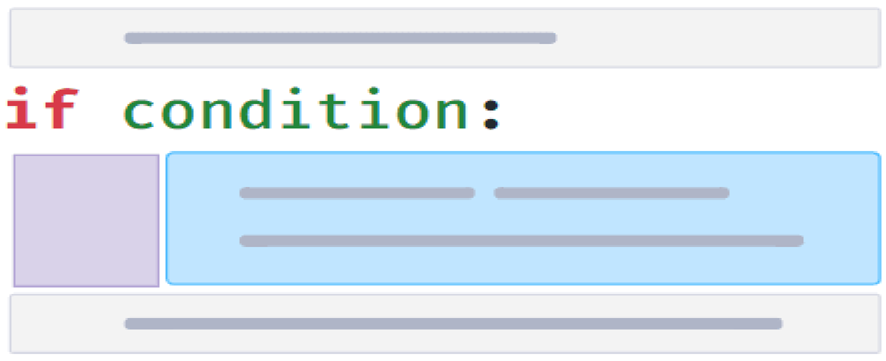
An expression that results in either **True** or **False**

Examples

i. 2 < 3
ii. a == b
iii. True

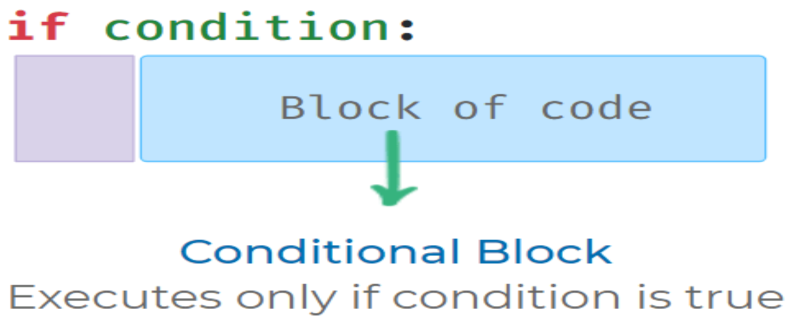
Conditional Statement

Conditional Statement allows you to execute a block of code only when a specific condition is **True**



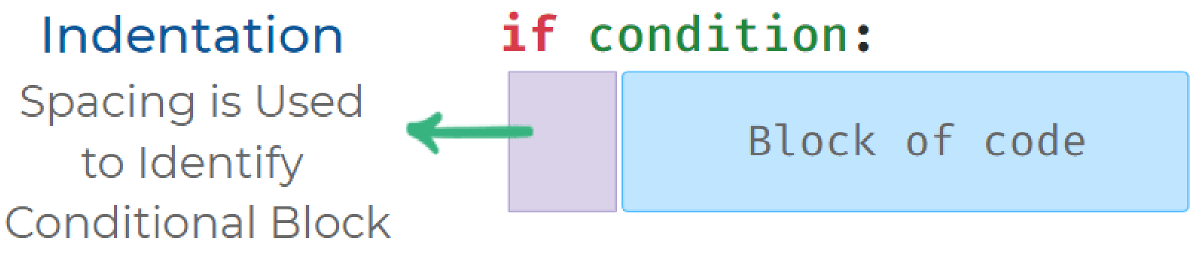
Conditional Block

Block of code which executes only if a condition is True is called Conditional Block.



Indentation:

- Space(s) in front of the conditional block is called indentation.
- Indentation(spacing) is used to identify Conditional Block.
- Standard practice is to use four spaces for indentation.



Possible Mistakes

Each statement inside a conditional block should have the same indentation (spacing).

Wrong Code:

```
if True:
  print("If Block")
      print("Inside If")
```

Output

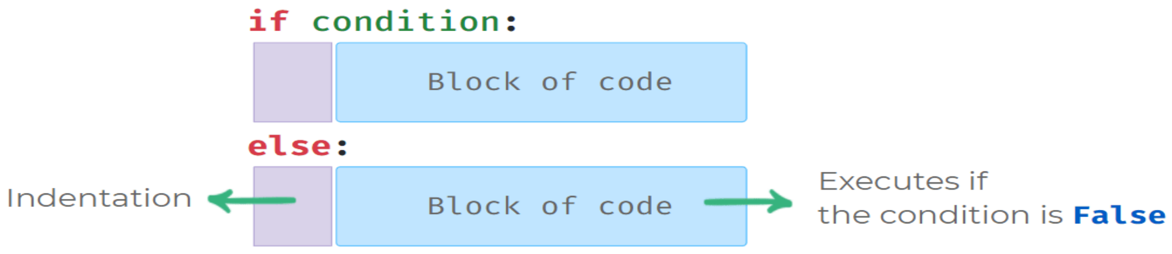
```
IndentationError: unexpected indent
```

Correct Code

```
if True:
  print("If Block")
  print("Inside If")
```

If - Else Syntax

When If - Else conditional statement is used, Else block of code executes if the condition is False



Using If-Else

Code

```
a = int(input())
if a > 0:
  print("Positive")
else:
  print("Not Positive")
print("End")
```

PYTHON

Input

```
2
```

Output

```
Positive
End
```

Possible Mistakes in If-Else

Else can only be used along with if condition. It is written below if conditional block

Code

```
if False:
   print("If Block")
print("After If")
else:
   print("Else Block")
print("After Else")
```

Output

```
SyntaxError: invalid syntax
```

Warning

Note: No code is allowed in between if conditional block and else statement.

More Arithmetic Operators:

Modulus

To find the remainder, we use Modulus operator **%**

- a % b

Code

```
print(6 % 3)
```

Output

```
0
```

Exponent

To calculate a power b, we use Exponent Operator ******

- a ** b

Code

```
print(2 ** 3)
```

Output

```
8
```

You can use the exponent operator to calculate the square root of a number by keeping the exponent as **0.5**

Code:

```
print(16 ** 0.5)
```

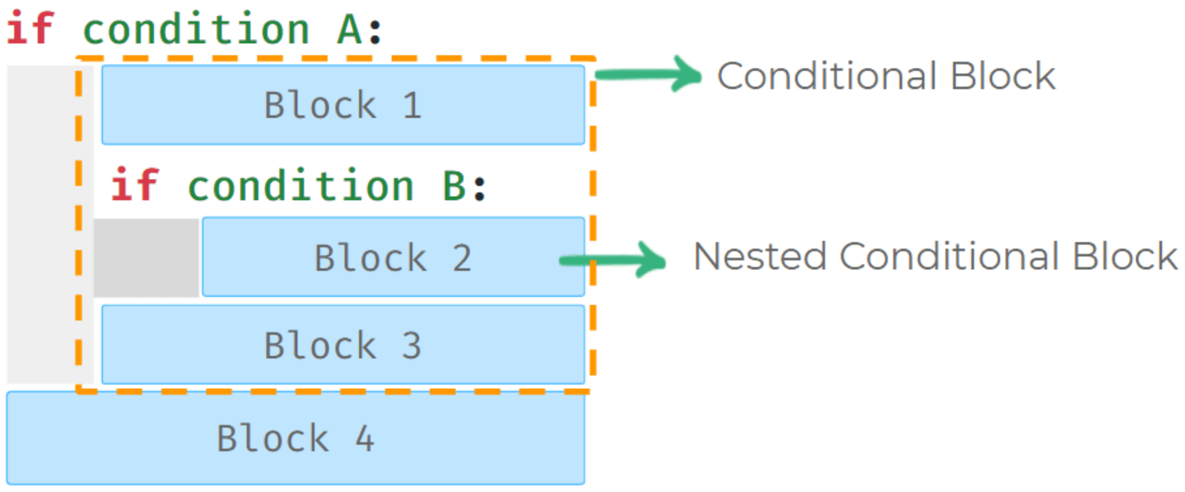
Output

```
4.0
```

Nested Conditional Statements:

Nested Conditions

The conditional block inside another if/else conditional block is called as nested conditional block. In the below example, Block 2 is nested conditional block and condition B is called nested conditional statement.



Code:

```
matches_won = int(input())
goals = int(input())
if matches_won > 8:
    if goals > 20:
        print("Hurray")
    print("Winner")
```

Input:

```
10
22
```

Output

```
Hurray
Winner
```

Input

10

18

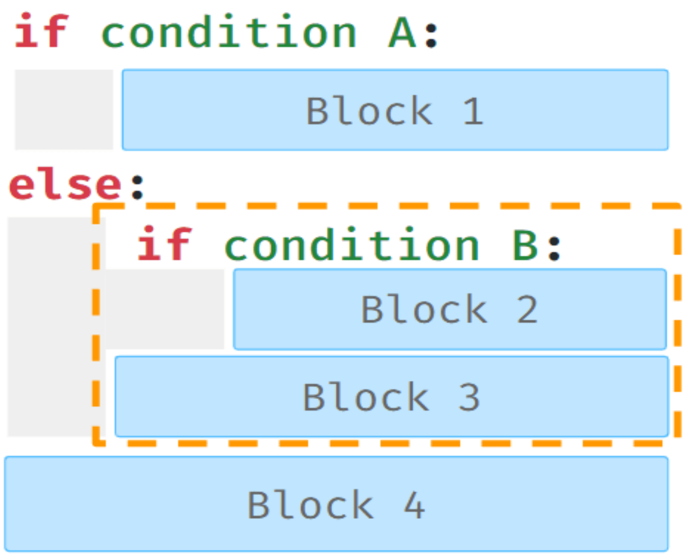
Output

Winner

Nested Condition in Else Block:

We can also write nested conditions in Else Statement.

In the below example *Block 2* is a nested conditional block.



Code:

```
a = 2
b = 3
c = 1
is_a_greatest = (a > b) and (a > c)
if is_a_greatest:
    print(a)
else:
    is_b_greatest = (b > c)
    if is_b_greatest:
        print(b)
    else:
```

```
is_b_greatest = (b > c)
   if is_b_greatest:
      print(b)
   else:
      print(c)
```

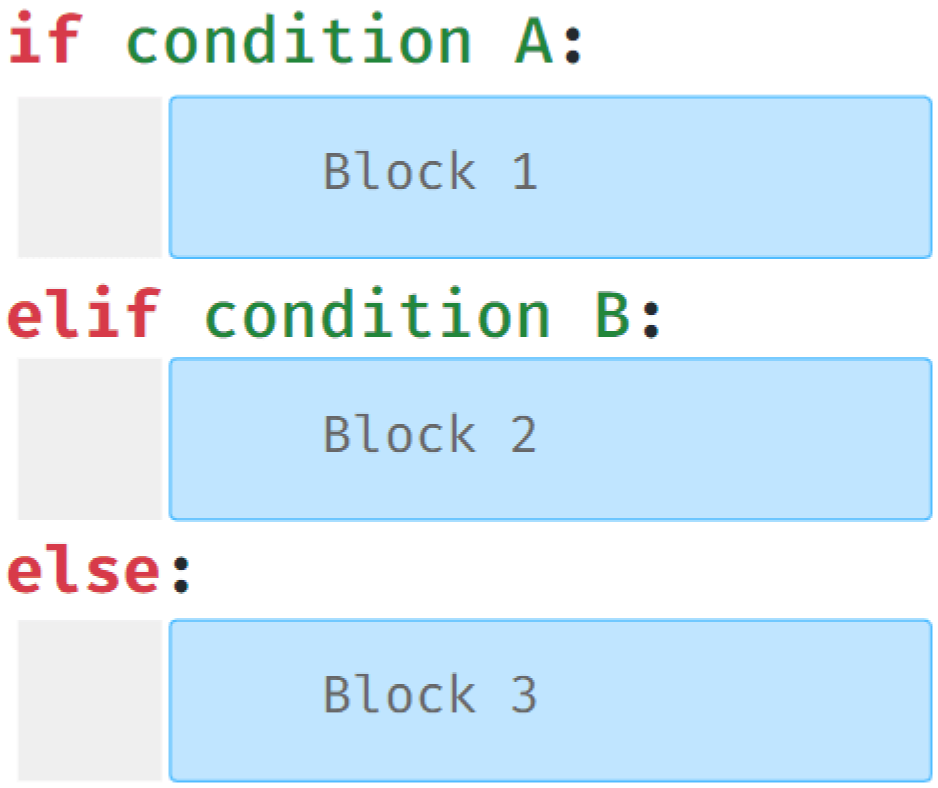
Output

```
3
```

Elif Statement:

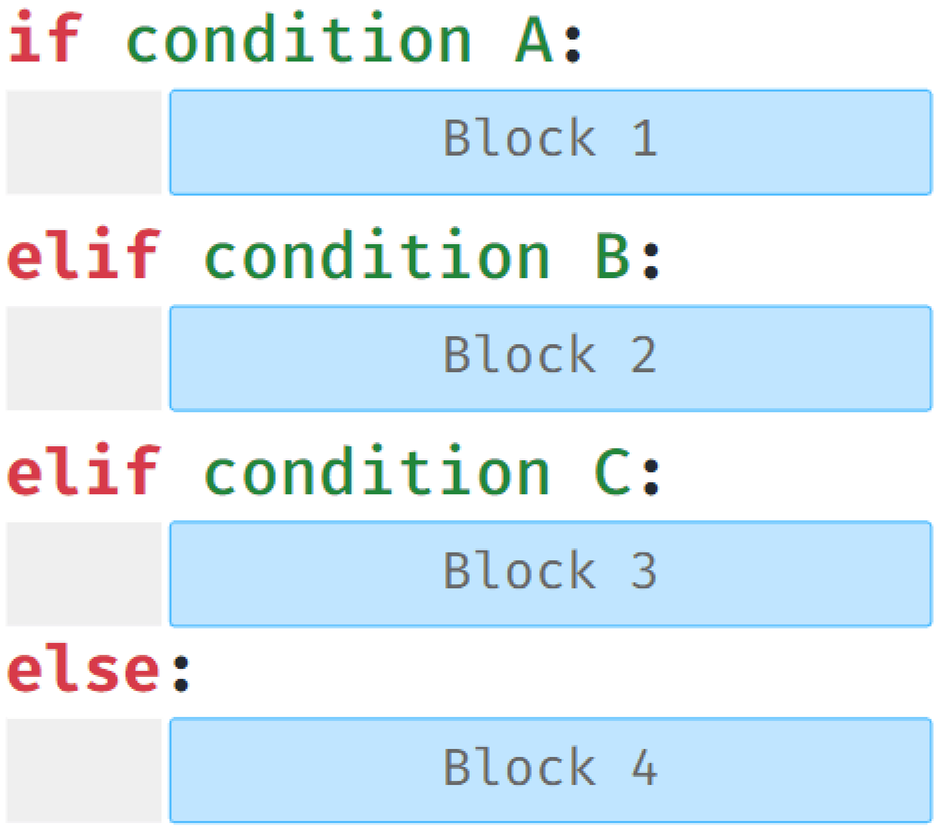
Use the elif statement to have multiple conditional statements between if and else.

The elif statement is optional.



Multiple Elif Statements

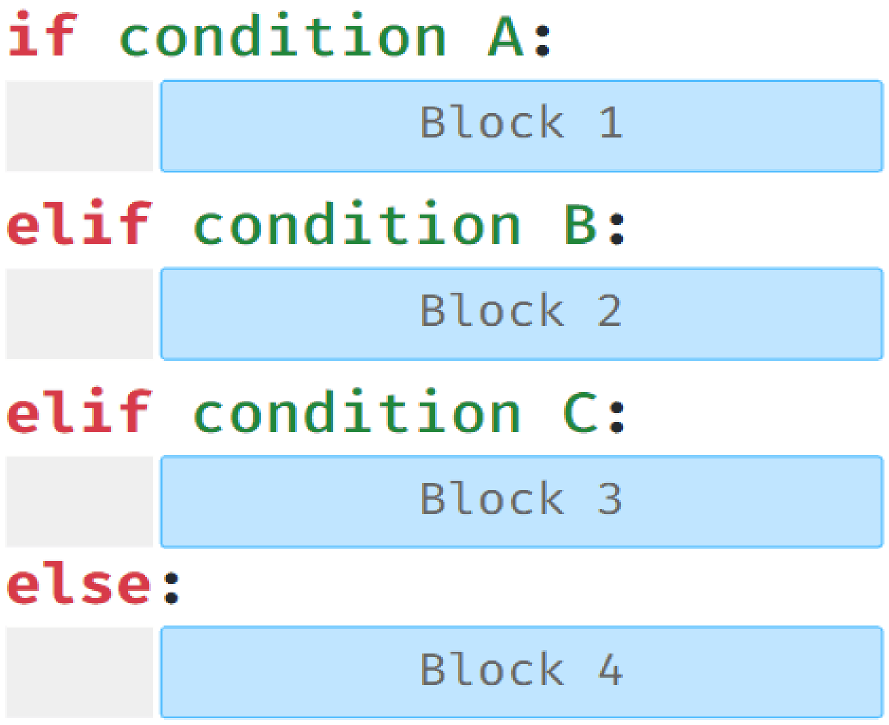
We can add any number of **elif**statements after **if** conditional block.



```
if condition A:
    Block 1
elif condition B:
    Block 2
elif condition C:
    Block 3
else:
    Block 4
```

Execution of Elif Statement

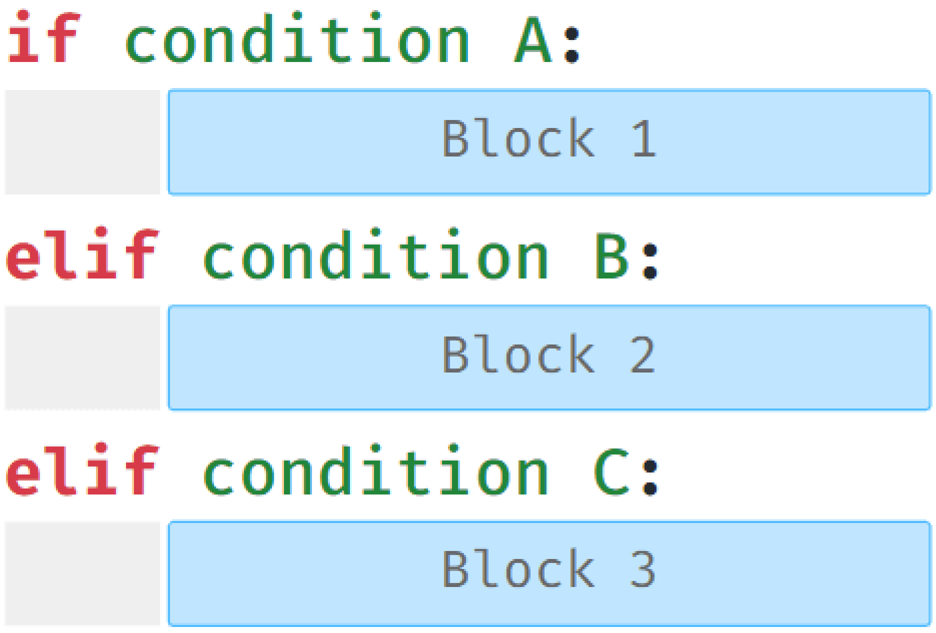
Python will execute the elif block whose expression evaluates to true.
If multiple **elif** conditions are true, then only the first elif block which is True will be executed.



```
if condition A:
    Block 1
elif condition B:
    Block 2
elif condition C:
    Block 3
else:
    Block 4
```

Optional Else Statement:

Else statement is not compulsory after **if – elif** statements.



Code

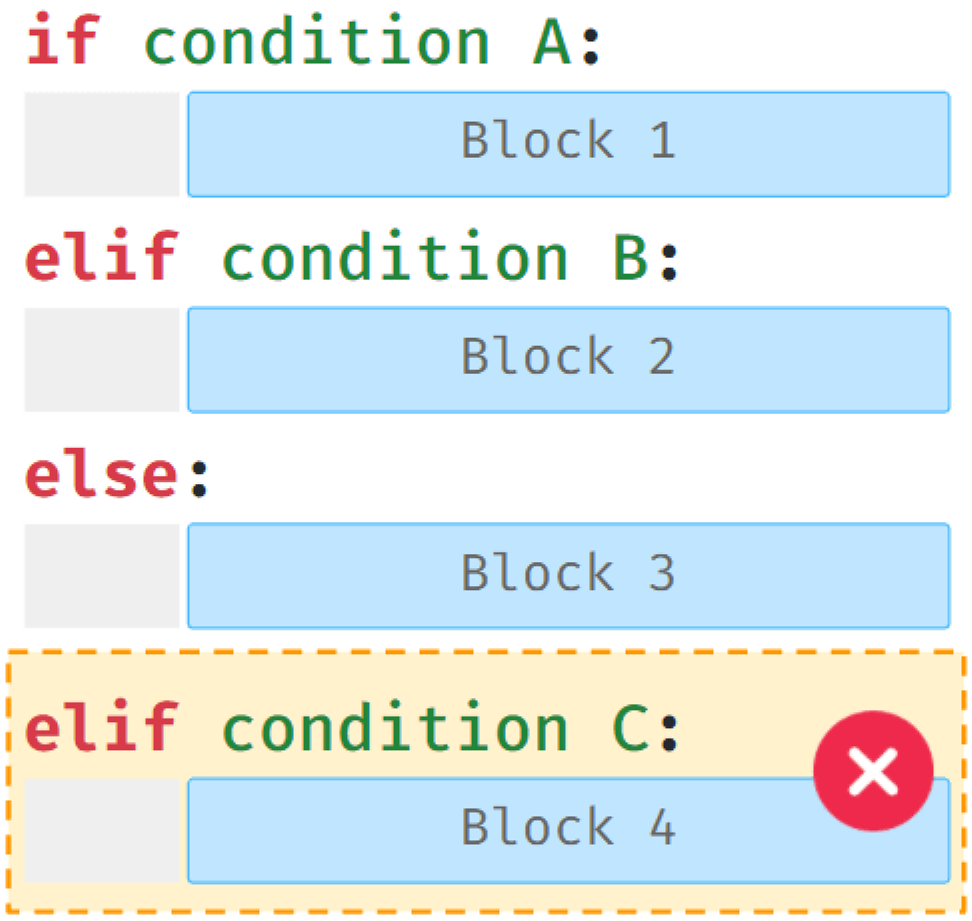
```
number = 5
is_divisible_by_10 = (number % 10 == 0)
is_divisible_by_5 = (number % 5 == 0)
if is_divisible_by_10:
    print("Divisible by 10")
elif is_divisible_by_5:
    print("Divisible by 5")
else:
   print("Not Divisible by 10 or 5")
```

Output

```
Divisible by 5
```

Possible Mistake:

Cannot write an elif statement after **else** statement.



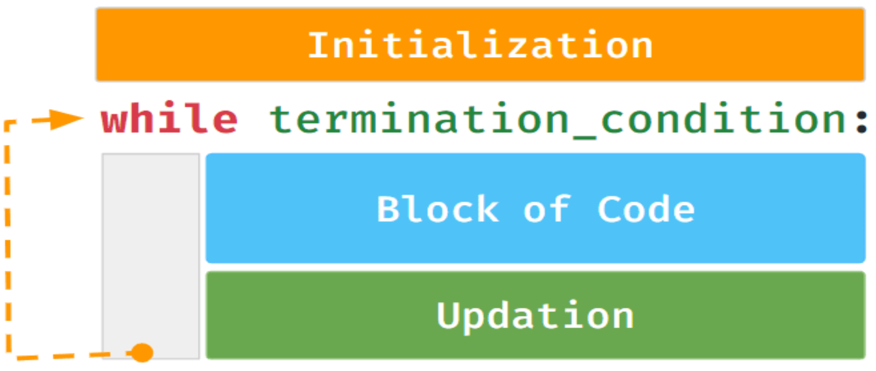
Loops:

So far we have seen that Python executes code in a sequence and each block of code is executed once. **Loops allow us to execute a block of code several times.**

While Loop:

Allows us to execute a block of code several times as long as the condition is **True**

.



While Loop Example:

The following code snippet prints the next three consecutive numbers after a given number.

Code:

```
a = int(input())
counter = 0
while counter < 3:
   a = a + 1
   print(a)
   counter = counter + 1
```

Input:

```
4
```

Output:

```
5
6
7
```

Possible Mistakes

1. Missing Initialization

Code

```
a = int(input())
while counter < 3:
   a = a + 1
   print(a)
   counter = counter + 1
print("End")
```

Input

```
5
```

Output

NameError: name 'counter' is not defined

2. Incorrect Termination Condition

Code

```
a = int(input())

counter = 0

condition = (counter < 3)

while condition:

   a = a + 1

   print(a)

   counter = counter + 1
```

Input

10

Output

The above code runs into an infinite loop.

While block will keep repeating as the value in condition variable is **True**.

3. Not Updating Counter Variable

Code

```
a = int(input())

counter = 0

while counter < 3:

   a = a + 1

   print(a)

print("End")
```

Input

10

Output:

Infinite Loop

As the value of counter is not updating.

Loop concept is over