



Let's Learn Basic Python !

Sintaks Group

This project is created by
Sintaks Group, with the
following members:

- Hasballah Askar
- Galang Setia Nugroho
- Khalishah Fiddina
- Tifani Amalina
- Muhammad Ilham Hakiqi



Use case summary

Objective Statement

Get insight into:

- Dynamic Typing
- Data Type Conversion
- String Transformation
- Replace String Element
- String Inspections
- Formating on String
- Operations on the list, set, and string
- Operators, Operands, and Expressions
- Conditional Expressions

Challenges

Syntax in python is case sensitive, therefore accuracy is needed in using syntax to perform an operation.

Methodology

Syntax and Operations for Dynamic Typing, Data Type Conversion, String Transformation, Replace String Element, String Inspections, Formatting on String, Operators, Operand and Expressions, and Conditional Expressions.

Use case summary

Benefit

Understand how to do and use -

- Dynamic Typing
- Data Type Conversion
- String Transformation
- Replace String Element
- String Inspections
- Formating on String
- Operations on the list, set, and string
- Operators, Operands, and Expressions
- Conditional Expressions

Expected Outcome

Knowing the syntax used to perform -

- Dynamic Typing
- Data Type Conversion
- String Transformation
- Replace String Element
- String Inspections
- Formating on String
- Operations on the list, set, and string
- Operators, Operands, and Expressions
- Conditional Expressions

Dynamic Typing in Python

Dynamic Typing in Python

Python is one of the dynamically-typed programming languages. Dynamically-typed languages are those where the interpreter assigns variables a type at runtime based on the variable's value at the time.


```
In [1]: 1 # Dynamic Typing
        2 a = 1
        3 b = "Hello World"
        4 c = 1.0
        5 d = 2 + 3j
        6
        7 print(type(a))
        8 print(type(b))
        9 print(type(c))
       10 print(type(d))
```

```
<class 'int'>
<class 'str'>
<class 'float'>
<class 'complex'>
```

Data Type Conversion

int to float


Here we want to convert 3 which is an integer type into a float which is 3.0



```
[ ] 3
[ ] 3
[ ] type(3)
    int
[ ] float(3)
    3.0
[ ] print(type(float(3)))
    <class 'float'>
```

float to int


Here we want to convert 2.5 which is a float type into a integer which is 2



```
[ ] 2.5
    2.5
[ ] type(2.5)
    float
[ ] int(2.5)
    2
[ ] print(type(int(2.5)))
    <class 'int'>
```


string to float


Here we want to convert "1.5" which is a string type into a float which is 1.5



```
[ ] "1.5"  
    '1.5'  
  
[ ] type("1.5")  
    str  
  
[ ] float("1.5")  
    1.5  
  
[ ] type(float("1.5"))
```

list to set

Here we want to convert [4,5,6] which is a list type into a set which is {4,5,6}



```
[ ] [4,5,6]  
    [4, 5, 6]  
  
[ ] type([4,5,6])  
    list  
  
[ ] set([4,5,6])  
    {4, 5, 6}  
  
[ ] type(set([4,5,6]))  
    set
```

string to list


Here we want to form a list that is 'D', 'a', 't', 'a' of strings.



```
[ ] 'Data'
'Data'
[ ] list("Data")
['D', 'a', 't', 'a']
```

list to dictionary

From a list, namely `[[4,5], [6,7]]`, we can form a dictionary from that list, which is `{4:5, 6:7}`




```
[ ] [[4,5],[6,7]]
[[4, 5], [6, 7]]
[ ] dict([[4,5],[6,7]])
{4: 5, 6: 7}
[ ] [(4,5),(6,7)]
((4, 5), (6, 7))
[ ] dict(((4,5),(6,7)))
{4: 5, 6: 7}
```

Input and Output


What is a Variable?

A Python variable is a reserved memory location to store values of a certain data type.

Storing value "3" (int type) on variable x

 `x = 3`

Storing value "Sintax" (str type) on variable w

 `w = "Sintax"`

Input

Any information or data sent to the computer from the user through the keyboard is called **Input**.

Input Process

```
Group = input("Group Name: ")
```

The result:

```
Group Name: Sintax
```

Output

The information produced by the computer to the user is called **Output**. The syntax for the output in python is **print()**.

Output Process

There are many ways to display an output in python, including:

```
a = "yz"
```

```
print(a)
```

```
#produced value on the variable a
```

```
yz
```

1

```
x = 3
```

```
print("Nilai variabel x adalah : ",x)
```

```
Nilai variabel x adalah : 3
```

2

```
print("This is SINTAX {}".format("Group"))
```

```
This is SINTAX Group
```

3

```
Group = "Our"  
Name = "Sintax"  
People = 5
```

```
print("This is %s Group \nThe name of this group is %s \nThere are %d people in this group" %(Group,Name,People))
```

```
This is Our Group  
The name of this group is Sintax  
There are 5 people in this group
```

4

%s for Str type

%d for integer type

%f for float type

String Transformation

String Transformation

Python provides several methods to transform text/string data. Here are some of them:

lower()

This method converts all uppercase characters from the given string into lowercase. Any symbols and numbers are ignored. To use this method, follow this syntax: ***string.lower()***

```
"Data SCIENCE - 01".lower()
```

```
'data science - 01'
```

upper()

This method converts all lowercase characters from the given string into uppercase. Any symbols and numbers. To use this method, follow this syntax: ***string.upper()***

```
"Data SCIENCE - 02".upper()
```

```
'DATA SCIENCE - 02'
```

title()

This method converts the first letter of each word from the given string into uppercase. To use this method, follow this syntax: ***string.title()***

```
"Data SCIENCE - 03".title()
```

```
'Data Science - 03'
```

String Transformation

strip()

This method removes all whitespaces from right and left side of the string. Space is the default character to remove. To use this method, follow this syntax: ***string.strip()***

```
"      this is text 3      ".strip()  
'this is text 3'
```

You can also set any characters to remove by following this syntax: ***string.strip(characters)***

```
"aaaaabbbbbthis is text 3-----.....".strip("ab-.")  
'this is text 3'
```


Replace Element in String

Replace element in String

replace()

This method replaces a specified phrase with another specified phrase. To use this method, follow this syntax: **string.replace(oldvalue, newvalue, count(optional)).**

Where:

- **oldvalue** is the string to search for,
- **newvalue** is the string to replace the old value with, and
- **count** is a number specifying how many occurrences of the old value you want to replace (default is all occurrences).

```
[1] Pathway = "data science track"
```

```
[2] #if we want replace word science with engineer  
Pathway.replace("science","engineer")
```

```
'data engineer track'
```

```
[3] text = "data science and data engineer"
```

```
[7] #if we want replace only one word data with real  
text.replace("data","real",1)
```

```
'real science and data engineer'
```

```
▶ #if we want replace two word data with real  
text.replace("data","real",2)
```

```
👉 'real science and real engineer'
```

String Inspection

String Inspection

Python provides several methods to check or inspect text/string data. Here are some of them:

islower()

This method checks whether **all the alphabet** characters of the string are **lowercase** or not. If the string contains at least 1 uppercase alphabet, it returns "False". To use this method, follow this syntax: ***string.islower()***

```
"lesson-01".islower()
```

```
True
```

```
"Lesson-01".islower()
```

```
False
```

isupper()

This method checks whether **all the alphabet** characters of the string are **uppercase** or not. If the string contains at least 1 lowercase alphabet, it returns "False". To use this method, follow this syntax: ***string.isupper()***

```
"LESSON-02".isupper()
```

```
True
```

```
"LESSoN-02".isupper()
```

```
False
```

String Inspection

isalpha()

This method checks whether the string includes **only alphabet characters** or not. If the string contains symbol, number or space, it returns "False". To use this method, follow this syntax: ***string.isalpha()***

```
"LessonFour".isalpha()
```

```
True
```

```
"Lesson four".isalpha()
```

```
False
```

isalnum()

This method checks whether the string includes **only alphanumeric characters** or not. Meaning only alphabet letter (a-z) and numbers (0-9). To use this method, follow this syntax: ***string.isalnum()***

```
"123abc".isalnum()
```

```
True
```

```
"123-abc".isalnum()
```

```
False
```

String Inspection

startswith()

This method checks whether the string **starts with the specified value** or not. To use this method, follow this syntax: *string.startswith(value)*

```
"Hello, Everyone!".startswith("Hell")
```

```
True
```

```
"Hello, Everyone!".startswith("hell")
```

```
False
```

endswith()

This method checks whether the string **ends with the specified value** or not. To use this method, follow this syntax: *string.endswith(value)*

```
"Hello, Everyone!".endswith("one!")
```

```
True
```

```
"Hello, Everyone!".endswith("One!")
```

```
False
```

Note: the value is case-sensitive!

Formatting on String

zfill()

This method adds zeros (0) at the beginning of the string, until it reaches the specified length. If the value of the len parameter is less than the length of the string, no filling is done. To use this method, follow this syntax: ***string.zfill(len)***.

Where:

- **len**: A number specifying the desired length of the string

```
[8] year = 2022
```

```
[11] #if we fill value more than the length of the string  
str(year).zfill(7)
```

```
'0002022'
```

```
[12] #if we fill value less than the length of the string  
str(year).zfill(3)
```

```
'2022'
```


Operations on List, Set and String

Len()

The `len()` function returns the number of items in an object.

When the object is a string, the `len()` function returns the number of characters in the string.

```
[2] b={3,5,5,7,7,7}
```

```
[3] print(b)
```

```
{3, 5, 7}
```

```
[4] len(b)
```

```
3
```

because sets
don't count the
same value so the
lens only has 3

```
[ ] a = [3,5,5,7,7,7]
```

```
[ ] print(a)
```

```
[3, 5, 5, 7, 7, 7]
```

```
[ ] len(a)
```

```
6
```

Min () and Max ()

Min() : This function is used to compute the minimum of the values passed in its argument and lexicographically smallest value if strings are passed as arguments.



Max() : This function is used to compute the maximum of the values passed in its argument and lexicographically largest value if strings are passed as arguments.



```
[ ] a = [3,5,7]
```

```
[ ] min(a)
```

```
3
```

```
[ ] max(a)
```

```
7
```

Count()

The count() method returns the number of elements with the specified value.

we want to count the number of 5 values in the list which amount to 2



```
[ ] a = [3,5,5,7,7,7]
```

```
[ ] a.count(5)
```

```
2
```

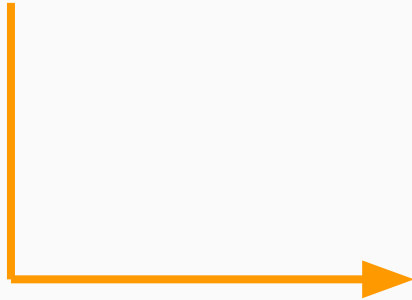
```
[ ] c = "Data Science"
```

```
[ ] c.count("a")
```

```
2
```

Merging and Replication

We can also do merging and replication in python



```
[ ] a = [3,5,5,7,7,7]
```

```
[ ] d = ["d","a","t","a"]
```

```
[ ] a + d
```

```
[3, 5, 5, 7, 7, 7, 'd', 'a', 't', 'a']
```

```
[ ] d*2
```

```
['d', 'a', 't', 'a', 'd', 'a', 't', 'a']
```

Range

The range() function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and stops before a specified number.



```
[ ] for i in range(3):  
    print(i)
```

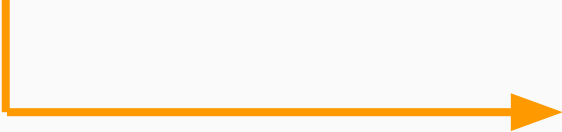
```
0  
1  
2
```

```
[ ] for i in range(0,10):  
    print(i)
```


```
0  
1  
2  
3  
4  
5  
6  
7  
8  
9
```

In and Not In

In operator in Python checks whether a specified value is a constituent element of a sequence like [string](#), [array](#), [list](#), or [tuple](#), etc.



Not in operator in Python works exactly the opposite way as the in operator works. It also checks the presence of a specified value inside a given sequence but its return values are opposite to that of the in operator.



```
[ ] family = "father and mother"

[ ] "father" in family
    True

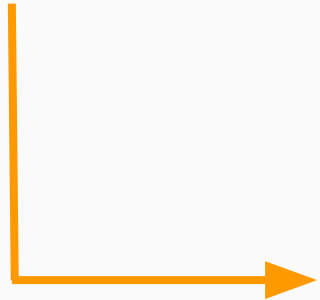
[ ] "mother" in family
    True

[ ] "father" not in family
    False

[ ] "mother" not in family
    False
```

Assign values to multiple variables

Python allows you to assign values to multiple variables



```
[ ] Food = ["Fried rice", "Chicken sausage", "Spicy"]
```

```
[ ] name, topping, taste = Food
```

```
[ ] print(name)
```

```
Fried rice
```

```
[ ] print(taste)
```

```
Spicy
```

```
[ ] print(taste)
```

```
Spicy
```


Sort()

The `sort()` method sorts the list ascending by default.

The reverse sort method sorts the list descending



```
[6] b = ["d", "i", "a"]
```

```
[7] b.sort()
```

```
[8] print(b)
```

```
['a', 'd', 'i']
```

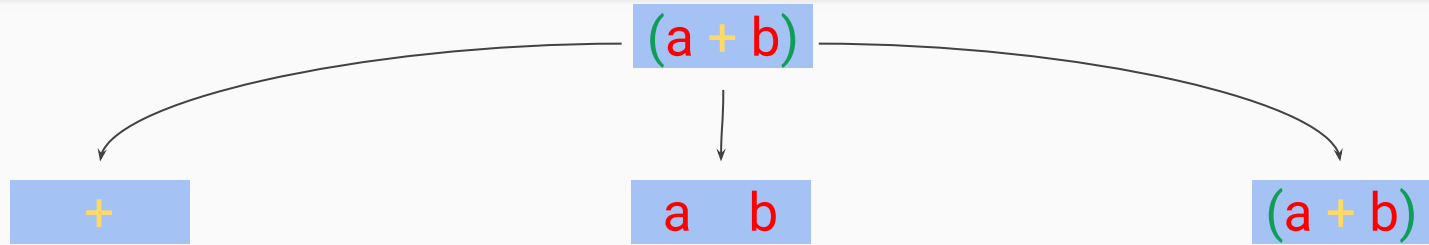
```
[9] b.sort(reverse=True)
```

```
[10] print(b)
```

```
['i', 'd', 'a']
```

Operators, Operands and Expressions

Operators, Operands and Expressions



Operators

Operators are special symbols that designate that some sort of computation should be performed

Operands

The values that an operator acts on are called operands

Expression

An expression is a combination of operators and operands that is interpreted to produce some other value.

Relational Operators

```
x = int(input("Enter x value = "))
y = int(input("Enter y value = "))
z = int(input("Enter z value = "))
print("=====")
print("Relational Operators")
print("\n")
# Relational Operators
print("Less than result = ", x<y<z)
print("Greater than result = ", x>y>z)
print("Less than or equal to result = ", x<=y<=z)
print("Greater than or equal to result = ", x>=y>=z)
print("Equal to result = ", x==y==z)
print("Not equal to result = ", x!=y!=z)
print("=====")
```

The output:



Relational Operators includes:

- Less than (<)
- Greater than (>)
- Less than or Equal to (<=)
- Greater than or Equal to (>=)
- Equal to (==)
- Not Equal to (!=)

Enter x value = 12

Enter y value = 13

Enter z value = 14

=====

Relational Operators

Less than result = True

Greater than result = False

Less than or equal to result = True

Greater than or equal to result = False

Equal to result = False


Not equal to result = True

=====

Arithmetic Operators

```
print("\n")
print("Arithmetic Operations")
print("\n")
x = 11
y = 12
z = 13
# Arithmetic Operations
print("Addition = ", x+y+z)
print("Subtraction = ", x-y-z)
print("Multiplication = ", x*y*z)
print("Division = ", x/y)
print("Modulus = ", x%z)
print("Exponentiation = ", x**y)
print("Floor division = ", y//z)
```

The output:



Arithmetic Operators includes:

- Addition (+)
- Subtraction (-)
- Multiplication (*)
- Division (/)
- Modulus (%)
- Exponentiation (**)
- Floor Division (//)

Arithmetic Operations

```
Addition = 36
Subtraction = -14
Multiplication = 1716
Division = 0.9166666666666666
Modulus = 11
Exponentiation = 3138428376721
Floor division = 0
```

Logical Operators

```
a = True
b = False
c = True
```

Logical Operators includes:

- AND
- OR
- NOT

```
# Logical Operators AND
print(a and b)
print(a and c)
print(b and c)
print(a and b and c)
```

The output of AND Operation:

```
False
True
False
False
```

```
# Logical Operators OR
print(a or b)
print(a or c)
print(b or c)
print(a or b or c)
```

The output of OR Operation:

```
True
True
True
True
```

```
# Logical Operators NOT
print(not a)
print(not b)
print(not c)
```

The output of nOT Operation:

```
False
True
False
```

Identity, Membership and Assignment Operators

Identity Operators includes:

- IS and IS NOT

```
# Identity Operators
x = ["ROG", "Legion", "HP", "Logitech"]
y = ["ROG", "Legion", "HP", "Logitech"]
z = x

print(x is z)

print(x is y)

print(z is not x)
```

The output:

```
True
False
False
```

Membership Operators includes:

- IN and NOT IN

```
# Membership Operators
x = ["ROG", "Legion", "HP", "Logitech"]
y = ["ROG", "Legion", "HP", "Logitech"]

print("ROG" in x )

print("Legion" not in y)
```

The output:

```
True
False
```

Assignment Operators includes:

- Addition (+=)
- Subtraction (-=)
- Multiplication (*=)

```
# Assignment Operators
x = 10
x += 10
print(x)

x = 10
x -= 10
print(x)

x = 10
x *= 10
print(x)
```

The output:

```
20
0
100
```

Conditional Expressions

Conditional Expressions

- Streams that control Python program code based on conditional testing.
- The syntax used to create control flow in the form of conditional statements in Python is **if**, **elif** (else if) and **else**.

if Syntax:

```
#IF  
  
Group = "Sintax Group"  
  
if Group:  
    print("This is {}".format(Group))
```

```
This is Sintax Group
```

Conditional Expressions

else Syntax:

```
#ELSE

Group = int(input("Members of the Group: "))

if Group ==5:
    print("The number of group members matches")
else:
    print("The number of group members doesn't match")
```

The output:

Members of the Group:

```
Members of the Group: 5
The number of group members matches
```

If the number of members entered isn't equal to 5, then it proceed to **else**.

```
Members of the Group: 8
The number of group members doesn't match
```

Conditional Expressions

elif Syntax:

```
#ELSEIF -> Else If

Score = int(input("How much your score: "))

if Score >= 80:
    print("Congrats! your score it's Awesome")
elif Score >= 70:
    print("Congrats! your score it's Great")
elif Score >= 60:
    print("Congrats! your score it's Good")
else:
    print("Sorry, you have to retake")
```

The output:

The output will be displayed according to each result of the condition statement.

```
How much your score: 100
Congrats! your score it's Awesome
```

```
How much your score: 80
Congrats! your score it's Great
```

```
How much your score: 60
Congrats! your score it's Good
```

```
How much your score: 40
Sorry, you have to retake
```

Thank You!

