Keywords

- Value Keywords: True, False, None.
- Operator Keywords: and, or, not, in, is.
- Control Flow Keywords: if, elif, else.
- Iteration Keywords: for, while, break, continue, else.
- Structure Keywords: def, class, with, as, pass, lambda.
- Returning Keywords: return, yield.
- Import Keywords: import, from, as.

Python Identifiers

- Identifiers can be a combination of letters in lowercase (a to z) or uppercase (A to Z) or digits (0 to 9) or an underscore _ . ...
- An identifier cannot start with a digit. ...
- Keywords cannot be used as identifiers. ...
- We cannot use special symbols like !, @, #, $, % etc. ...
- An identifier can be of any length.

Variable Names

Rules for Python variables:

- A variable name must start with a letter or the underscore character
- A variable name cannot start with a number
- A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
- Variable names are case-sensitive (age, Age and AGE are three different variables)

Example

```
#Legal variable names:
myvar = "John"
my_var = "John"
_my_var = "John"
myVar = "John"
MYVAR = "John"
myvar2 = "John"

#Illegal variable names:
2myvar = "John"
my-var = "John"
my var = "John"
```

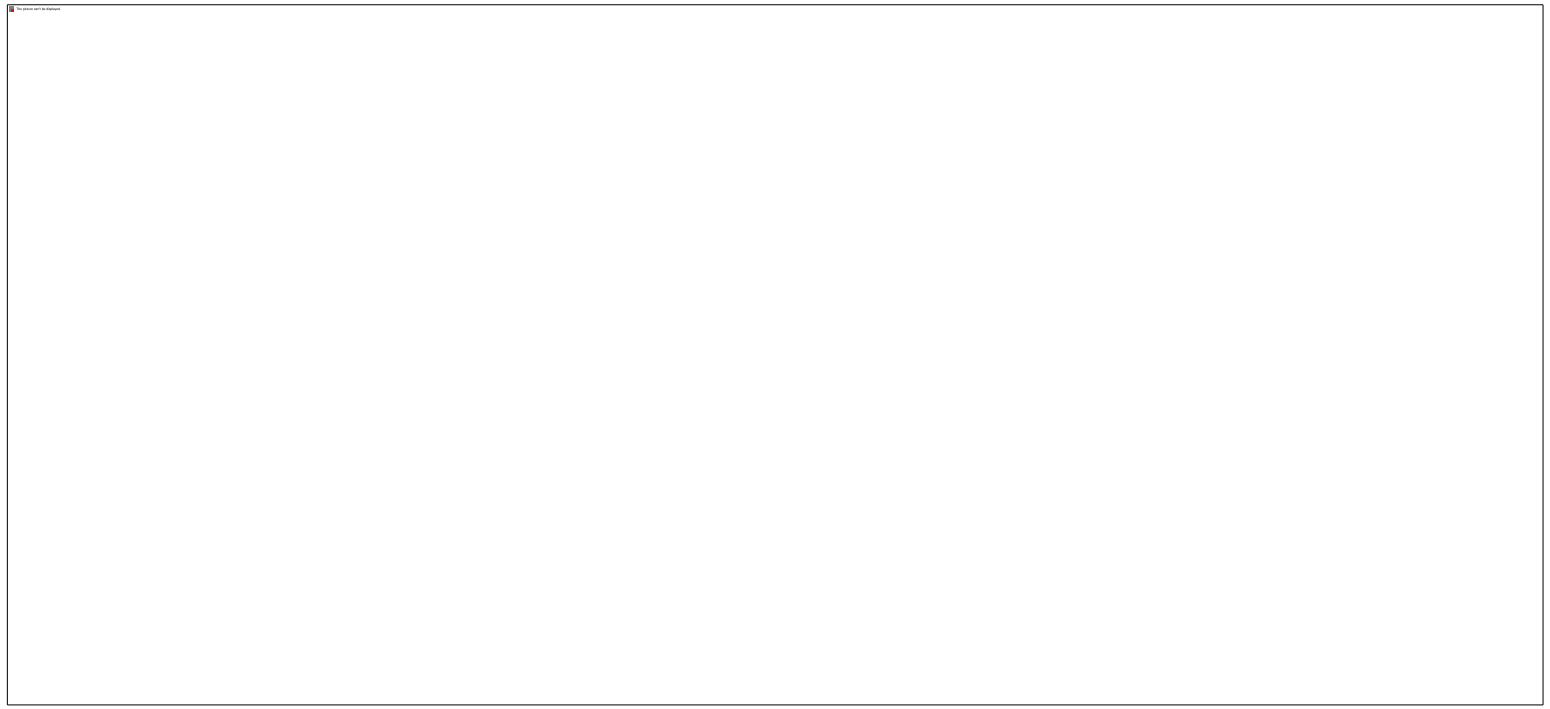
Literals

Literals in Python is defined as the raw data assigned to variables or constants while programming. We mainly have five types of literals which includes string literals, numeric literals, boolean literals, literal collections and a special literal None.

String Literals in Python

```
#string literals
#single line literal
single_quotes_string='Scaler Academy'
double_quotes_string="Hello World"
print(single_quotes_string)
print(double_quotes_string)
```

Numeric Literals in Python



```
# integer literal
#positive whole numbers
x = 2586
#negative whole numbers
y = -9856
# binary literal
a = 0b10101
# decimal literal
b = 505
# octal literal
c = 0o350
# hexadecimal literal
d = 0x12b
print (x,y)
print(a, b, c, d)
```

Boolean Literals in Python

- True- True represents the value 1.
- False-False represents the value 0.

```
#boolean literals
x = (1 == 1)
y = (7 == False)
print("x is", x)
print("y is", y)
```

Special Literals in Python

- Python literals have one special literal known as None. This literal in Python is used to signify that a particular field is not created.

- Python will print None as output when we print the variable with no value assigned to it. None is also used for end of lists in Python.

```
#special literals
val=None
print(val)
```

Literal Collections in Python

- List Literals
- Tuple Literals
- Dictionary Literals
- Set Literals

Punctuators

Punctuators: These are the symbols that used in Python to organize the structures, statements, and expressions. Some of the Punctuators are: [ ] { } ( ) @ -= += *= //= **== =

Data Type

- Numbaers a=1
- String a=”hi” a=’hi’
- List a=[1 , 2 , 3 , ’a’ , ’b’ , 3.44]
- Tuple a=(‘a’ , 1 , 2.44)
- Set a={1 , 2 , 3 , 4}
- Dictionary a={‘a’:1 , ‘b’:2 }

Operators

- Arithmetic operators
- Assignment operators
- Comparison operators
- Logical operators
- Identity operators
- Membership operators
- Bitwise operators

Python Arithmetic Operators

Operator	Name	Example
+	Addition	x + y
-	Subtraction	x - y
*	Multiplication	x * y
/	Division	x / y
%	Modulus	x % y
**	Exponentiation	x ** y
//	Floor division	x // y

```
x = 2
y = 5
z= x**y
print(z) #same as
2*2*2*2*2
```

```
x = 15
y = 2
z=x//2
print(z)
#the floor division // rounds the result down to the nearest whole
number
```

Python Assignment Operators

Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
//=	x //= 3	x = x // 3
**=	x **= 3	x = x ** 3

```
x = 5

x **= 3

print(x)
```

Python Comparison Operators

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

```
x = 5
y = 3

print(x <= y)

# returns False because 5 is neither less than or equal to 3
```

Python Logical Operators

Operator	Description	Example
and	Returns True if both statements are true	x < 5 and x < 10
or	Returns True if one of the statements is true	x < 5 or x < 4
not	Reverse the result, returns False if the result is true	not(x < 5 and x < 10)

```
x = 5

print(x > 3 and x < 10)

# returns True because 5 is greater than 3 AND 5 is less than 10
```

Python Identity Operators

Operator	Description	Example
is	Returns True if both variables are the same object	x is y
is not	Returns True if both variables are not the same object	x is not y

```
x = ["apple", "banana"]
y = ["apple", "banana"]
z = x

print(x is z)

# returns True because z is the same object as x

print(x is y)

# returns False because x is not the same object as y, even if they have the same content

print(x == y)

# to demonstrate the difference betweeen "is" and "==": this comparison returns True because x is equal to y
```

```
x = ["apple", "banana"]
y = ["apple", "banana"]
z = x

print(x is not z)

# returns False because z is the same object as x

print(x is not y)

# returns True because x is not the same object as y, even if they have the same content

print(x != y)

# to demonstrate the difference betweeen "is not" and "!=": this comparison returns False because x is equal to y
```

Python Membership Operators

Operator	Description	Example
in	Returns True if a sequence with the specified value is present in the object	x in y
not in	Returns True if a sequence with the specified value is not present in the object	x not in y

```
x = ["apple", "banana"]

print("banana" in x)

# returns True because a sequence with the value "banana" is in the list
```

```
x = ["apple", "banana"]

print("pineapple" not in x)

# returns True because a sequence with the value "pineapple" is not in the list
```

Expression

An expression is a combination of operators and operands that is interpreted to produce some other value. In any programming language, an expression is evaluated as per the precedence of its operators.

```
x = int(1)
y = int(2.8)
z = int("3")
print(x)
print(y)
print(z)
```

int() - constructs an integer number from an integer literal, a float literal (by removing all decimals), or a string literal (providing the string represents a whole number)

float() - constructs a float number from an integer literal, a float literal or a string literal (providing the string represents a float or an integer)

```
x = str("s1") # x will be 's1'
y = str(2)    # y will be '2'
z = str(3.0)  # z will be '3.0'
```

str() - constructs a string from a wide variety of data types, including strings, integer literals and float literals

Conditional Statements

demo.py - C:/Users/anand rawool/Desktop/demo.py (3.8.5)

File Edit Format Run Options Window Help

```
num1= input("enter frist number ")
num2= input("enter sec numbsr")
op= input("enter oprater")
if op == "+":
    num3=num1 + num2
    print(num3)
```

demo.py - C:/Users/anand rawool/Desktop/demo.py (3.8.5)

File Edit Format Run Options Window Help

```
num1= input("enter frist number ")
num2= input("enter sec numbsr")
op= input("enter oprater")
if op == "+":
    num3=num1 + num2
    print(num3)
if op == "-":
    num3=num1 -num2
    print(num3)
if op == "*":
    num3=num1 * num2
    print(num3)
if op == "/":
    num3=num1 / num2
    print(num3)
if op == "%":
    num3=num1 % num2
    print(num3)
```

demo.py - C:/Users/anand rawool/Desktop/demo.py (3.8.5)

File Edit Format Run Options Window Help

```
num1= int(input("enter frist number "))
num2= int(input("enter sec numbsr"))
op= input("enter oprater")
if op == "+":
    num3=num1 + num2
    print(num3)
if op == "-":
    num3=num1 -num2
    print(num3)
if op == "*":
    num3=num1 * num2
    print(num3)
if op == "/":
    num3=num1 / num2
    print(num3)
if op == "%":
    num3=num1 % num2
    print(num3)
else:
    print("not vailed")
```

*demo.py - C:/Users/anand rawool/Desktop/demo.py (

File Edit Format Run Options Window Help

```
num1= int(input("enter frist number "))
num2= int(input("enter sec numbsr"))
op= input("enter oprater")
if op == "+":
    num3=num1 + num2
    print(num3)
elif op == "-":
    num3=num1 -num2
    print(num3)
elif op == "*":
    num3=num1 * num2
    print(num3)
elif op == "/":
    num3=num1 / num2
    print(num3)
elif op == "%":
    num3=num1 % num2
    print(num3)
else:
    print("not vailed")
```