Lecture_11_Python_variables_operators_&_conditional_blocks

15 September 2024 14:09

Operators.ipynb - Colab

*Python operators *

Arithmetic operators

Assignment operators

Comparison operators

Logical operators

Identity operators

Membership operators

Bitwise operators

```
#Arithmetic Operators
# + Addition    x + y
# - Subtraction x - y
# * Multiplication  x * y
# / Division    x / y
# % Modulus x % y
# **    Exponentiation  x ** y
# //    Floor division  x // y
x=5
y=3
print(x+y)
print(x-y)
print(x*y)
print(x/y)
print(x%y)
print(x**y)
print(x//y)
```

```
8
2
15
1.6666666666666667
2
125
1
```

```
# Assignment operators
# =   x = 5   x = 5
# +=    x += 3  x = x + 3   x op= c  x=x op c
# -=    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 |= 3  x = x | 3
# ^=    x ^= 3  x = x ^ 3
# >>=   x >>= 3 x = x >> 3
# <<=   x <<= 3 x = x << 3
x=9
print(x)
x+=9 #x=x+9
print(x)
x-=9 #x=x-9
print(x)
x *= 3 #x=x*3
print(x)
y=7
y/=2
print(y)
y %= 2
print(y)
y **= 5
print(y)
```

```
9
18
9
27
3.5
1.5
7.59375
```

```
# Comparison operators
#== 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=5
print(x==y)
y=6
print(x==y)
print(x>y)
print(x<y)
print(x>=y)
print(x<=y)
```

```
True
False
False
True
False
True
```

```
#Logical operators
# and   Returns True if both statements are true    x < 15 and  x < 10
# or    Returns True if one of the statements is true   x < 15 or x < 4
# not   Reverse the result, returns False if the result is true not(x < 15 and x < 10)
x=25
print(x < 15 or  x < 10)
x=9
print(x < 15 and  x < 10)
x=12
print(not((x < 15 and x < 10))) # not (c1 and c2)  (not c1 or not c2)
```

```
False
True
True
```

```
#Identity Operators
#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=5
y=5
print(x is y)
print(x is not y)
```

```
True
False
```

```
# Membership operators
#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)
```

```
True
```

```
# Bitwise operators
#&  AND Sets each bit to 1 if both bits are 1   x & y
#|  OR  Sets each bit to 1 if one of two bits is 1  x | y
#^  XOR Sets each bit to 1 if only one of two bits is 1 x ^ y
#~  NOT Inverts all the bits    ~x
#<< Zero fill left shift    Shift left by pushing zeros in from the right and let the leftmost bits fall off    x << 2
# >>    Signed right shift  Shift right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off x >>
x=1
#01 in binary
y=3
#11 in binary
#0100
#11
#10
print(x & y)
print(x | y )
print(x ^ y) #10 answer
print(y << 1)
y=3
print(y << 2)
```

```
print(y << 3)
print(y >> 1)
```

```
1
3
2
6
12
24
1
```

The operator precedence in Python is as follows:

1. Parentheses: expressions inside parentheses are evaluated first.
2. Exponentiation: the exponentiation operator ** is evaluated next.
3. Multiplication, division, and modulo: *, /, and % are evaluated from left to right.
4. Addition and subtraction: + and - are evaluated from left to right.
5. Bitwise shift operators: << and >> are evaluated next.
6. Bitwise AND: & is evaluated next.
7. Bitwise XOR: ^ is evaluated next.
8. Bitwise OR: | is evaluated next.
9. Comparison operators: ==, !=, >, <, >=, <=, is, and is not are evaluated from left to right.
10. Boolean NOT: not is evaluated next.
11. Boolean AND: and is evaluated next.
12. Boolean OR: or is evaluated next.
13. Conditional expression: if-else is evaluated last.

It's important to note that you can change the order of evaluation by using parentheses to group expressions. For example, (2 + 3) * 4 will be evaluated as 20 because the parentheses force the addition to be evaluated first.

Start coding or generate with AI.

Variables_K
eywords_...

In Python, variables serve as storage units for holding data values. Unlike other programming languages, Python does not require a separate declaration command for variables.

Instead, a variable is instantiated as soon as a value is assigned to it for the first time.

There are several fundamental data types that are commonly used in Python, including:

Integers: Used to represent whole numbers, such as 1, 2, 3, etc.

Floating-point numbers: Used to represent decimal numbers, such as 1.5, 2.7, etc.

Strings: Used to represent sequences of characters, such as "hello", "world", etc.

Booleans: Used to represent True or False values.

Lists: Used to represent ordered collections of items, which can be of different data types.

Tuples: Similar to lists, but are immutable, meaning that their values cannot be changed after they are created.

Dictionaries: Used to represent collections of key-value pairs, where each key is unique.

To retrieve the data type of a variable, you can make use of the built-in type() function.

```
x = 5
y = "Sabhawna"
print(type(x))
print(type(y))
```

```
<class 'int'>
<class 'str'>
```

```
a=True
b=None
c=7.8
d=2+9j
print(a,b,c,d)
print(type(a))
print(type(b))
print(type(c))
print(type(d))
```

```
True None 7.8 (2+9j)
<class 'bool'>
<class 'NoneType'>
<class 'float'>
<class 'complex'>
```

Variable names are case-sensitive.

```
a='Hello'
A=9
#A will not overwrite a
```

In Python, variables can have either a short or a descriptive name, depending on the context. For instance, a variable that stores the value of an age could be named "age," while a variable that stores the total volume of a liquid could be named "total_volume." There are certain rules that must be followed when naming variables in Python:

A variable name must begin with a letter or underscore character.

A variable name cannot begin with a number.

A variable name can only contain alphanumeric characters and underscores (A-Z, a-z, 0-9, and _).

A variable name cannot be any of the Python keywords.

```
myvar = "Sadbhawna"
my_var = "Sadbhawna"
_my_var = "Sadbhawna"
myVar = "Sadbhawna"
MYVAR = "Sadbhawna"
myvar2 = "Sadbhawna"
```

```
3myvar = "Sadbhawna"
my-var = "Sadbhawna"
my var = "Sadbhawna"
```

```
  File "<ipython-input-5-d7b832fd58fc>", line 1
    3myvar = "Sadbhawna"
    ^
SyntaxError: invalid decimal literal
```

Next steps: Fix error

```
import keyword

print(keyword.kwlist)
print(len(keyword.kwlist))
```

It is possible to assign values to multiple variables in a single line. It is also possible to assign the same value to multiple variables in a single line.

```
x,y,z=1,2.5,3+7j
print(x)
print(y)
print(z)
```

```
x=y=z=1
print(x)
print(y)
print(z)
```

Start coding or generate with AI.

if_else.ipynb - Colab

Python Logical Conditions:

Equals: a == b

Not Equals: a != b

Less than: a < b

Less than or equal to: a <= b

Greater than: a > b

Greater than or equal to: a >= b

Commonly used in "if statements" and loops.

if Statement

if test expression:

```
statement(s)
```

if ... else Statement

if test expression:

```
Body of if
```

else:

```
Body of else
```

```
num =  int(input("Enter an Int: "))
if num > 0:
  print("Positive")
else:
  print("Negative")
```

```
Enter an Int: 0
Negative
```

Python relies on indentation-instead of curly brackets (whitespace - usually 2 tab spaces at the beginning of a line) to define scope in the code.

if...elif...else Statement

The elif keyword specifies a condition (and then corresponding action) which is checked if previous conditions were false.

```
num =  int(input("Enter an Int: "))
if num > 0:
    print("Positive number")
elif num == 0:
    print("ZERO")
else:
    print("Negative Number")
```

```
Enter an Int: 0
ZERO
```

Nested if else Statements

```
# input three integer numbers
a=int(input("Enter A: "))
b=int(input("Enter B: "))
c=int(input("Enter C: "))

# Conditions to find the maximum element
if a>b:
    if a>c:
        m=a
    else:
        m=c
else:
    if b>c:
        m=b
    else:
        m=c

# print the largest number
print("Maximum element is  = ",m)
```

```
Enter A: 46
Enter B: 37
Enter C: 63
Maximum element is  =  63
```

and keyword is used to combine conditional statements.

```
a = 300
b = 9
c = 400
if a > b and c > a:
  print("Both conditions are True")
```

```
Both conditions are True
```