Types of Expressions in Python

Your Name

April 12, 2024

1 Constant Expressions

These are the expressions that have constant values only.

```
[language=Python]
# Constant Expressions
x = 15 + 1.3

print(x)
```

Output:

```
16.3
```

2 Arithmetic Expressions

An arithmetic expression is a combination of numeric values, operators, and sometimes parentheses. The result of this type of expression is also a numeric value. The operators used in these expressions are arithmetic operators like addition, subtraction, etc. Here are some arithmetic operators in Python:

- Addition: $x + y$
- Subtraction: $x - y$
- Multiplication: $x \times y$
- Division: $\frac{x}{y}$
- Quotient: $x//y$
- Remainder: $x\%y$
- Exponentiation: $x**y$

Let's see an exemplar code of arithmetic expressions in Python:

```
[language=Python]
# Arithmetic Expressions
x = 40
y = 12

add = x + y
sub = x - y
pro = x * y
div = x / y

print(add)
print(sub)
print(pro)
print(div)
```

Output:

```
52
28
480
3.3333333333333335
```

3 Integral Expressions

These are the kind of expressions that produce only integer results after all computations and type conversions.

```
[language=Python]
# Integral Expressions
a = 13
b = 12.0

c = a + int(b)
print(c)
```

Output:

```
25
```

4 Floating Expressions

These are the kind of expressions which produce floating point numbers as a result after all computations and type conversions.

```
[language=Python]
# Floating Expressions
```

```
a = 13
b = 5

c = a / b
print(c)
```

Output:

```
2.6
```

5 Relational Expressions

In these types of expressions, arithmetic expressions are written on both sides of relational operators ($>, <, \geq, \leq$). Those arithmetic expressions are evaluated first, and then compared as per the relational operator and produce a boolean output in the end. These expressions are also called Boolean expressions.

```
[language=Python]
# Relational Expressions
a = 21
b = 13
c = 40
d = 37

p = (a + b) >= (c - d)
print(p)
```

Output:

```
True
```

6 Logical Expressions

These are kinds of expressions that result in either True or False. It basically specifies one or more conditions. For example, $(10 == 9)$ is a condition if 10 is equal to 9. As we know it is not correct, so it will return False. Studying logical expressions, we also come across some logical operators which can be seen in logical expressions most often. Here are some logical operators in Python:

- **and**: $P \wedge Q$ - Returns true if both P and Q are true, otherwise returns false
- **or**: $P \vee Q$ - Returns true if at least one of P and Q is true
- **not**: $\neg P$ - Returns true if condition P is false

Let's have a look at an exemplar code:

```
[language=Python]
P = (10 == 9)
Q = (7 > 5)

# Logical Expressions
R = P and Q
S = P or Q
T = not P

print(R)
print(S)
print(T)
```

Output:

```
False
True
True
```

7 Bitwise Expressions

These are the kind of expressions in which computations are performed at the bit level.

```
[language=Python]
# Bitwise Expressions
a = 12

x = a >> 2
y = a << 1

print(x, y)
```

Output:

```
3 24
```

8 Combinational Expressions

We can also use different types of expressions in a single expression, and that will be termed as combinational expressions.

```
[language=Python]
# Combinational Expressions
a = 16
b = 12
```

```
c = a + (b >> 1)
print(c)
```

Output:

```
22
```

9 Operator Precedence

When we combine different types of expressions or use multiple operators in a single expression, operator precedence comes into play. Operator Precedence simply defines the priority of operators, determining which operator is to be executed first.

Precedence	Name	Operator
1	Parenthesis	( ) [ ] { }
2	Exponentiation	**
3	Unary plus or minus, complement	-a, +a, a
4	Multiply, Divide, Modulo	/, *, //, %
5	Addition & Subtraction	+, -
6	Shift Operators	¿¿, ¡¡
7	Bitwise AND	&
8	Bitwise XOR	ˆ
9	Bitwise OR	—
10	Comparison Operators	¿=, ¡=, ¿, ¡
11	Equality Operators	==, !=
12	Assignment Operators	=, +=, -=, /=, *=
13	Identity and membership operators	is, is not, in, not in
14	Logical Operators	and, or, not

Table 1: Operator Precedence in Python

For example, if we have the expression $10 + 3 * 4$, it must yield 22 due to operator precedence.

```
[language=Python]
# Multi-operator expression

a = 10 + 3 * 4
print(a)

b = (10 + 3) * 4
print(b)

c = 10 + (3 * 4)
print(c)
```

Output:

```
22
52
22
```

Hence, operator precedence plays an important role in the evaluation of a Python expression.