Variables

1. Variables:

   A variable is a name that refers to a value in memory. In Python, you don't need to specify the data type of a variable when you declare it, as the interpreter infers the data type from the value assigned to it. Here's an example:

makefile Copy code

```
# Assigning a string value to a variable called name
name = "John"

# Assigning an integer value to a variable called age
age = 30

# Assigning a boolean value to a variable called is_student
is_student = True
```

Print Function

2. Print function:

   The `print()` function is used to display text or variables on the console. You can pass one or more arguments to the `print()` function, separated by commas, and it will display them on the console. Here's an example:

bash Copy code

```
# Printing a string literal
print("Hello, world!")

# Printing variables
name = "John"
age = 30
print("My name is", name, "and I am", age, "years old.")
```

Input Function

3. Input function:

   The `input()` function is used to get input from the user on the console. It displays a message to the user and waits for them to enter some text, which it returns as a string. Here's an example:

python Copy code

```python
# Getting input from the user and storing it in a variable
name = input("What is your name? ")

# Printing the value of the variable
print("Hello,", name)
```

Data Types

4. Data types:

   In Python, there are several data types, including integers, floats, strings, booleans, and lists, among others. You can check the type of a variable using the `type()` function. Here are some examples:

python Copy code

```python
# Integers
x = 5
print(type(x))  # Output: <class 'int'>

# Floats
y = 3.14
print(type(y))  # Output: <class 'float'>

# Strings
name = "John"
print(type(name))  # Output: <class 'str'>

# Booleans
is_student = True
print(type(is_student))  # Output: <class 'bool'>

# Lists
my_list = [1, 2, 3]
print(type(my_list))  # Output: <class 'list'>
```

Type Conversion

5. Type conversion:

   Sometimes you may need to convert a variable from one data type to another. In Python, you can use built-in functions to do this. Here are some examples:

makefile Copy code

```
# Converting an integer to a float
x = 5
y = float(x)
print(y)  # Output: 5.0

# Converting a string to an integer
age = "30"
age_int = int(age)
print(age_int)  # Output: 30

# Converting a string to a float
pi = "3.14"
pi_float = float(pi)
print(pi_float)  # Output: 3.14
```

First Program

```
# Get the user's name and age
name = input("What is your name? ")
age_str = input("What is your age? ")

# Convert the age to an integer
age = int(age_str)

# Calculate the year the user was born
current_year = 2023
birth_year = current_year - age

# Print a greeting with the user's name and birth year
print("Hello, " + name + "! You were born in " + str(birth_year) + ".")
```

Arithmetic Operators

In Python, arithmetic operators are used to perform mathematical operations. Below are the commonly used arithmetic operators:

1. Addition (+): Adds two or more numbers and returns their sum.

makefile Copy code

```
Example:
a = 5
b = 3
c = a + b
print(c) # Output: 8
```

2. Subtraction (-): Subtracts the second number from the first number and returns the difference.

makefile Copy code

```
Example:
a = 10
b = 3
c = a - b
print(c) # Output: 7
```

3. Multiplication (*): Multiplies two or more numbers and returns their product.

makefile Copy code

```
Example:
a = 4
b = 6
c = a * b
print(c) # Output: 24
```

4. Division (/): Divides the first number by the second number and returns the quotient as a float.

makefile Copy code

```
Example:
a = 10
b = 3
c = a / b
print(c) # Output: 3.3333333333333335
```

5. Floor Division (//): Divides the first number by the second number and returns the quotient as an integer.

makefile Copy code

```
Example:
a = 10
b = 3
c = a // b
print(c) # Output: 3
```

6. Modulus (%): Divides the first number by the second number and returns the remainder.

makefile Copy code

```
Example:
a = 10
b = 3
c = a % b
print(c) # Output: 1
```

7. Exponentiation (**): Raises the first number to the power of the second number.

makefile Copy code

```
Example:
a = 2
b = 3
c = a ** b
print(c) # Output: 8
```

Relational Operators

Relational operators in Python are used to compare two values and determine their relationship with each other. These operators return a Boolean value (True or False) depending on whether the comparison is true or false.

Here are the relational operators in Python:

1. `>` : greater than operator - returns True if the left operand is greater than the right operand, otherwise False.
   Example:

makefile Copy code

```
x = 10
y = 5
print(x > y) # Output: True
```

2. `<` : less than operator - returns True if the left operand is less than the right operand, otherwise False.
   Example:

makefile Copy code

```
x = 10
y = 5
print(x < y) # Output: False
```

3. `>=` : greater than or equal to operator - returns True if the left operand is greater than or equal to the right operand, otherwise False.
   Example:

makefile Copy code

```
x = 10
y = 5
print(x >= y) # Output: True
```

4. `<=` : less than or equal to operator - returns True if the left operand is less than or equal to the right operand, otherwise False.
   Example:

makefile Copy code

```
x = 10
y = 5
print(x <= y) # Output: False
```

5. `==` : equal to operator - returns True if the left operand is equal to the right operand, otherwise False.
   Example:

makefile Copy code

```
x = 10
y = 5
print(x == y) # Output: False
```

6. `!=` : not equal to operator - returns True if the left operand is not equal to the right operand, otherwise False.
   Example:

makefile Copy code

```
x = 10
y = 5
print(x != y) # Output: True
```

Bitwise Operators

In Python, bitwise operators are used to perform bitwise operations on integers. These operators work on the binary representation of the integers. There are six bitwise operators in Python:

1. Bitwise AND (&)
2. Bitwise OR (|)
3. Bitwise XOR (^)
4. Bitwise NOT (~)
5. Left shift (<<)
6. Right shift (>>)

Here are some examples to help illustrate each operator:

1. Bitwise AND (&): The bitwise AND operator returns a 1 in each bit position where both corresponding bits are 1s, and 0 in all other positions.

makefile Copy code

```
a = 60  # 0011 1100
b = 13  # 0000 1101

c = a & b  # 0000 1100

print(c)  # output: 12
```

2. Bitwise OR (|): The bitwise OR operator returns a 1 in each bit position where at least one corresponding bit is a 1.

makefile Copy code

```
a = 60  # 0011 1100
b = 13  # 0000 1101

c = a | b  # 0011 1101

print(c)  # output: 61
```

3. Bitwise XOR (^): The bitwise XOR operator returns a 1 in each bit position where exactly one corresponding bit is a 1.

makefile Copy code

```
a = 60  # 0011 1100
b = 13  # 0000 1101

c = a ^ b  # 0011 0001

print(c)  # output: 49
```

4. Bitwise NOT (~): The bitwise NOT operator returns the complement of a number by changing all the bits from 0 to 1, and from 1 to 0.

makefile Copy code

```
a = 60  # 0011 1100

c = ~a  # 1100 0011

print(c)  # output: -61
```

Note that the bitwise NOT operator returns the two's complement of the input integer, which can be negative in Python.

5. Left shift (<<): The left shift operator shifts the bits of a number to the left by a specified number of positions, filling in the empty spaces with 0s.

makefile Copy code

```
a = 60  # 0011 1100

c = a << 2  # 1111 0000

print(c)  # output: 240
```

6. Right shift (>>): The right shift operator shifts the bits of a number to the right by a specified number of positions, filling in the empty spaces with 0s.

makefile Copy code

```
a = 60  # 0011 1100

c = a >> 2  # 0000 1111

print(c)  # output: 15
```

These are the basic bitwise operators in Python that can be used to manipulate the binary representation of integers.

Logical Operators

Logical operators in Python are used to perform logical operations between two or more values. Python has three logical operators: `and`, `or`, and `not`.

Here are some examples of how these operators work:

`and` Operator

The `and` operator returns True if both conditions are True, and False otherwise.

python Copy code

```python
a = 5
b = 10
c = 15
print(a < b and b < c)  # True, both conditions are True
print(a > b and b < c)  # False, first condition is False
print(a < b and b > c)  # False, second condition is False
print(a > b and b > c)  # False, both conditions are False
```

`or` Operator

The `or` operator returns True if at least one of the conditions is True, and False otherwise.

python Copy code

```python
a = 5
b = 10
c = 15
print(a < b or b < c)   # True, both conditions are True
print(a > b or b < c)   # True, second condition is True
print(a < b or b > c)   # True, first condition is True
print(a > b or b > c)   # False, both conditions are False
```

`not` Operator

The `not` operator is used to reverse the logical state of a condition. If a condition is True, `not` returns False, and if a condition is False, `not` returns True.

python Copy code

```python
a = 5
b = 10
print(not a < b)   # False, condition is True
print(not a > b)   # True, condition is False
```

Note: Logical operators always return a Boolean value (`True` or `False`) depending on the evaluation of the conditions.

Assignment Operators

In Python, assignment operators are used to assign a value to a variable. These operators combine the assignment and arithmetic operations in a single statement.

Here are some common assignment operators in Python:

1. = (Assignment Operator): Assigns the value on the right-hand side to the variable on the left-hand side.

python Copy code

```python
x = 5
```

2. += (Addition Assignment Operator): Adds the value on the right-hand side to the variable on the left-hand side and assigns the result to the left-hand side variable.

python Copy code

```python
x += 2    # same as x = x + 2
```

3. -= (Subtraction Assignment Operator): Subtracts the value on the right-hand side from the variable on the left-hand side and assigns the result to the left-hand side variable.

python Copy code

```python
x -= 2    # same as x = x - 2
```

4. *= (Multiplication Assignment Operator): Multiplies the value on the right-hand side with the variable on the left-hand side and assigns the result to the left-hand side variable.

python Copy code

```python
x *= 2    # same as x = x * 2
```

5. /= (Division Assignment Operator): Divides the variable on the left-hand side by the value on the right-hand side and assigns the result to the left-hand side variable.

python Copy code

```python
x /= 2    # same as x = x / 2
```

6. %= (Modulus Assignment Operator): Computes the modulus of the variable on the left-hand side with the value on the right-hand side and assigns the result to the left-hand side variable.

```python
x %= 2     # same as x = x % 2
```

7. //= (Floor Division Assignment Operator): Performs floor division on the variable on the left-hand side with the value on the right-hand side and assigns the result to the left-hand side variable.

```python
x //= 2     # same as x = x // 2
```

8. **= (Exponentiation Assignment Operator): Raises the variable on the left-hand side to the power of the value on the right-hand side and assigns the result to the left-hand side variable.

```python
x **= 2     # same as x = x ** 2
```

These assignment operators help to write concise and readable code in Python.

Compound Operators

In Python, compound operators are shorthand notations that combine an arithmetic or bitwise operation with an assignment operation. These operators allow you to perform an operation on a variable and store the result back into the same variable in one line of code.

Here are some examples of compound operators in Python:

1. Addition Assignment Operator (+=)
   The addition assignment operator adds the value on the right side of the operator to the value of the variable on the left side and assigns the result back to the same variable.

makefile Copy code

```
x = 5
x += 3
print(x) # Output: 8
```

2. Subtraction Assignment Operator (-=)
   The subtraction assignment operator subtracts the value on the right side of the operator from the value of the variable on the left side and assigns the result back to the same variable.

bash Copy code

```
x = 10
x -= 5
print(x) # Output: 5
```

3. Multiplication Assignment Operator (*=)
   The multiplication assignment operator multiplies the value on the right side of the operator with the value of the variable on the left side and assigns the result back to the same variable.

bash Copy code

```
x = 3
x *= 4
print(x) # Output: 12
```

4. Division Assignment Operator (/=)

   The division assignment operator divides the value of the variable on the left side by the value on the right side of the operator and assigns the result back to the same variable.

bash Copy code

```bash
x = 20
x /= 5
print(x) # Output: 4.0
```

5. Modulo Assignment Operator (%=)

   The modulo assignment operator performs a modulo operation on the value of the variable on the left side and the value on the right side of the operator, and assigns the result back to the same variable.

bash Copy code

```bash
x = 11
x %= 5
print(x) # Output: 1
```

6. Bitwise Assignment Operators (|=, &=, ^=, >>=, <<=)

   These operators perform bitwise operations on the value of the variable on the left side and the value on the right side of the operator, and assigns the result back to the same variable.

Example:

python Copy code

```python
x = 0b1010 # Binary representation of 10
x |= 0b0100 # Perform a bitwise OR operation with binary 0100
print(bin(x)) # Output: 0b1110, which is 14 in decimal
```

Note that the above examples are just a few of the many compound operators available in Python. They make code more concise and readable by reducing the amount of code needed to perform simple operations on variables.

Membership Operators

In Python, membership operators are used to test whether a value or variable is a member of a sequence or collection. There are two types of membership operators in Python:

1. `in` operator: Returns `True` if a value is found in the sequence or collection, otherwise `False`.
2. `not in` operator: Returns `True` if a value is not found in the sequence or collection, otherwise `False`.

Here are some examples of how to use membership operators in Python:

Example 1: Using `in` operator

bash Copy code

```bash
fruits = ['apple', 'banana', 'orange']
if 'banana' in fruits:
    print('Yes, banana is in the fruits list')
else:
    print('No, banana is not in the fruits list')
```

Output:

csharp Copy code

```csharp
Yes, banana is in the fruits list
```

Example 2: Using `not in` operator

bash Copy code

```bash
numbers = [1, 2, 3, 4, 5]
if 6 not in numbers:
    print('Yes, 6 is not in the numbers list')
else:
    print('No, 6 is in the numbers list')
```

Output:

csharp Copy code

```csharp
Yes, 6 is not in the numbers list
```

Example 3: Using membership operators with strings

swift Copy code

```swift
text = 'hello world'
if 'o' in text:
    print('Yes, "o" is in the text')
else:
    print('No, "o" is not in the text')

if 'z' not in text:
    print('Yes, "z" is not in the text')
else:
    print('No, "z" is in the text')
```

Output:

vbnet Copy code

```vbnet
Yes, "o" is in the text
Yes, "z" is not in the text
```

In summary, membership operators are useful for checking if a value is a member of a collection or sequence, and can be used with various data types in Python.

Identity Operators

In Python, identity operators are used to compare the memory locations of two objects. These operators are used to check whether the two objects have the same identity or not.

There are two identity operators in Python:

1. `is` operator: It returns True if two variables refer to the same object, otherwise False.
2. `is not` operator: It returns True if two variables do not refer to the same object, otherwise False.

Here are some examples:

Example 1:

python Copy code

```python
x = [1, 2, 3]
y = [1, 2, 3]
z = x

print(x is y)       # False, x and y refer to two different lists
print(x is z)       # True, x and z refer to the same list
print(x is not y)   # True, x and y do not refer to the same list
```

Example 2:

css Copy code

```
a = 5
b = 5

print(a is b)       # True, a and b refer to the same integer object in memo
print(a is not b)   # False, a and b refer to the same integer object in mem
```

Example 3:

vbnet Copy code

```
s1 = "hello"
s2 = "hello"
s3 = "world"

print(s1 is s2)     # True, s1 and s2 refer to the same string object in mem
print(s1 is not s3) # True, s1 and s3 do not refer to the same string object
```

In general, identity operators should be used only when we need to check whether two objects refer to the same object in memory. For checking the equality of two objects, we should use the equality operator (`==`).