Sharif University of Technology
Department of Computer Engineering

Fundamentals of Programming

Python Language

Arman Malekzadeh
PhD Candidate in Artificial Intelligence

Table of contents

Variable Scope

Variable Scope in Python

- In Python, variables are either global or local.
- Global variables are declared outside functions and can be accessed anywhere within the program.
- Local variables are declared inside a function and cannot be accessed outside that function.

Variable Scope in Python

```
x = 10 # This is a global variable

def func():
    y = 5 # This is a local variable
    print(x) # This will print 10

func()
print(y) # This will cause an error
```

In the example above, `x` is a global variable so it can be accessed inside `func()`. However, `y` is a local variable to `func()`, so trying to print it outside the function causes an error.

Nonlocal Variables in Python

- In Python, a variable declared outside of the function or in global scope is known as a global variable. This means, a global variable can be accessed inside or outside of the function.

- However, the nonlocal keyword is used to work with variables inside nested functions, where the variable should not belong to the inner function.

Nonlocal Variables in Python

```
def outer_function():
    x = 10
    def inner_function():
        nonlocal x # without this, 10 will be printed!
        x = 20
    inner_function()
    print("x =", x) # prints 20
outer_function()
```

- In this case, we use `nonlocal` keyword to create nonlocal variable inside the `inner_function()`. So, when we change the value of `x` inside the `inner_function()`, the change appears also in the outer function.
- This is essentially what nonlocal variables are for in Python.
- Remember: If we change the value of a nonlocal variable, the changes appear in the local variable.

Representing Numbers

Representing Numbers in Different Bases

In mathematics, representing a number in a base means expressing that number using the digits of that base.

- **Decimal** (base 10): 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
- **Binary** (base 2): 0, 1
- **Octal** (base 8): 0, 1, 2, 3, 4, 5, 6, 7
- **Hexadecimal** (base 16): 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

Representing Numbers in Different Bases

In mathematics, representing a number in a base means expressing that number using the digits of that base.

- For example, the number 10 can be represented in base 2 (binary) as 1010.
- The application of this is most commonly seen in computing, where binary is used to represent numbers and characters as a series of 0s and 1s.
- This allows computers to store data more efficiently and process it quickly.

Representing Numbers in Base 2: Example

1. Start with the number 1053.
2. Divide the number by 2. Write down the quotient and the remainder. In this case, $1053 \div 2 = 526$ (quotient) with a remainder of 1.
3. Repeat step 2 using the quotient obtained in the previous step. Now, $526 \div 2 = 263$ (quotient) with a remainder of 0.
4. Repeat step 3 again. Now, $263 \div 2 = 131$ (quotient) with a remainder of 1.

Representing Numbers in Base 2: Example

5. Continue this process until you reach a quotient of zero. The subsequent quotients and remainders will be as follows:
   - $131 \div 2 = 65$, remainder is 1
   - $65 \div 2 = 32$, remainder is 1
   - $32 \div 2 = 16$, remainder is zero
   - $16 \div 2 = 8$, remainder is zero
   - $8 \div 2 = 4$, remainder is zero
   - $4 \div 2 = 2$, remainder is zero
   - $2 \div 2 = 1$, remainder is zero
   - Finally, $1 \div 2 = 0$ with a remainder of one.

6. The binary representation of the decimal number is found by reading all remainders from bottom to top (from last obtained to first). So for our case, it will be: 10000011101.

Converting Numbers from Base 2 to Base 10

1. Write down the binary number and list the powers of 2 from right to left. The rightmost digit corresponds to 2^0, then 2^1 on its left, then 2^2 on its further left and so on.

$$\begin{array}{ccccccc} 1 & 0 & 1 & 0 & 1 & 1 & 1 \\ 2^6 & 2^5 & 2^4 & 2^3 & 2^2 & 2^1 & 2^0 \end{array}$$

2. Multiply each digit of the binary number with the corresponding power of two. $(1 * 64) + (0 * 32) + (1 * 16) + (0 * 8) + (1 * 4) + (1 * 2) + (1 * 1)$

3. Add up all these numbers. $64 + 16 + 4 + 2 + 1 = 87$

Converting Negative Numbers to Base 2

1. Convert the absolute value of the decimal number to binary.
2. Add leading zeros to the binary number until the number of digits is a multiple of 4.
3. Invert the digits of the binary number.
4. Add 1 to the inverted binary number.

Converting Negative Numbers to Base 2: -61 (7 bits)

1. Ignore the negative sign and start with the number 61.
2. Divide 61 by 2, which equals 30 with a remainder of 1. Write down the remainder.
3. Divide the quotient from step 2 by 2. This gives us 15 with a remainder of 0. Write down this remainder next to the previous one.

Converting Negative Numbers to Base 2: -61 (7 bits)

4. Continue this process of dividing by 2 and writing down the remainders until you reach a quotient of zero.

5. The binary representation of 61 is then given by concatenating all remainders from last to first. So for our case, we get 111101.

6. To represent -61 in binary (using Two's Complement), flip all bits (i.e., change all 1s to 0s and vice versa), add 1 at end, and put another 1 at the beginning for showing the sign. This gives us 1000011.

Adding two Numbers in Base 2

1. Write down the numbers so that they align from the right:

```
  1101
+ 1011
------
```

2. Start from the rightmost digit. Add the digits in each column:

```
  1101
+ 1011
------
  ???0 (since 1 + 1 = 10 in binary | carryover: 1)
```

Adding two Numbers in Base 2

3. Move to the next column. Remember to carry over if there's a '10':

```
  1101
+ 1011
------
 ???00 (since 0 + 1 + carryover(1) = '10' again)
 new carryover: 1
```

4. Continue this process for all columns:

```
  1101
+ 1011
------
 ??000 (since carryover(1) + '1' + '0' = '10')
 new carryover: 1
```

Adding two Numbers in Base 2

5. Last Column

```
  1101
+ 1011
------
 ?1000 (since carryover(1) + '1' + '1' = '11')
 new carryover: 1
```

6. Extra column

```
  1101
+ 1011
------
 11000 (since carryover(1) + nothing = '1')
```

So, $1101_2 + 1011_2 = 11000_2$.

Bitwise Operators

Operator	Description
&	AND
|	OR
^	XOR
~	NOT
<<	SHIFT LEFT
>>	SHIFT RIGHT

Bitwise Operators: Examples

AND operator copies a bit to the result if it exists in both operands.

```
a = 60 # 60 = 0011 1100
b = 13 # 13 = 0000 1101
print(a & b) # 12 = 0000 1100
```

OR operator copies a bit if it exists in either operand.

```
a = 60 # 60 = 0011 1100
b = 13 # 13 = 0000 1101
print(a | b) # 61 = 0011 1101
```

Bitwise Operators: Examples

XOR operator copies the bit if it is set in one operand but not both.

```
a = 60 # 60 = 0011 1100
b = 13 # 13 = 0000 1101
print(a ^ b) # 49 = 0011 0001
```

NOT operator inverts all the bits.

```
a = 60 # 60 = 0011 1100
print(~a) # -61 = 1100 0011
```

Converting -61 in Base 2 to Decimal

1. Recognize that the binary number is negative because it starts with a 1 (or someone has told you).

2. To convert this negative binary number to its positive form, we use the two's complement method. This involves reversing all of the bits (changing 1s to 0s and 0s to 1s) and then adding one.

3. Reverse all of the bits in your binary number: 00111100.

4. Add one to your reversed binary number: 00111101.

Converting -61 in Base 2 (11000011) to Decimal

5. Now, you can convert this positive binary number into decimal. Starting from the rightmost bit, multiply each bit by 2^n, where n is its position starting from 0. Sum up all these values.

6. So, $(0 * 2^7) + (0 * 2^6) + (1 * 2^5) + (1 * 2^4) + (1 * 2^3) + (1 * 2^2) + (0 * 2^1) + (1 * 2^0)$ equals $32 + 16 + 8 + 4 + 1 = 61$.

7. Since we started with a negative binary number, our final decimal result is also negative: -61.

Bitwise Operators: Examples

`SHIFT LEFT` operator moves the bits to the left, discards the far left bit, and assigns the rightmost bit a value of 0.

```
a = 2 # 2 = 10
print(a << 2) # 8 = 1000
```

`SHIFT RIGHT` operator moves the bits to the right, discards the far right bit, and assigns the leftmost bit a value of 0.

```
a = 8 # 8 = 1000
print(a >> 2) # 2 = 10
```

References

References I

[1] B Downey, A. (2015). Think Python: How to Think Like a Computer Scientist-2nd Edition.

[2] Deitel, H. M., & Deitel, P. J. (2004). C: How to program. Pearson Educacion.

Sharif University of Technology
Department of Computer Engineering

Arman Malekzadeh

Fundamentals of Programming

Python Language