Programming Fundamental

Lab#07

Contents

Input():

Input is used to read integers

```
In [3]: age = input("What is your age? ")
        print "Your age is: ", age

        What is your age? 23
        Your age is:  23
```

```
In [4]: age = input("What is your age? ")
        print "Your age is: ", age

        What is your age? abc

        ---------------------------------------------------------------------------
        NameError                                 Traceback (most recent call last)
        <ipython-input-4-c77460f9d278> in <module>()
        ----> 1 age = input("What is your age? ")
              2 print "Your age is: ", age

        C:\ProgramData\Anaconda2\lib\site-packages\ipykernel\ipkernel.pyc in <lambda>(prompt)
            162             self._sys_eval_input = builtin_mod.input
            163             builtin_mod.raw_input = self.raw_input
        --> 164             builtin_mod.input = lambda prompt='': eval(self.raw_input(prompt))
            165         self._save_getpass = getpass.getpass
            166         getpass.getpass = self.getpass

        C:\ProgramData\Anaconda2\lib\site-packages\ipykernel\ipkernel.pyc in <module>()

        NameError: name 'abc' is not defined
```

Operators

Operators are special symbols in Python that carry out arithmetic or logical computation. The value that the operator operates on is called the operand.

>>> 3+2

5

Here, `+` is the operator that performs addition. `2` and `3` are the operands and `5` is the output of the operation.

```
In [2]: a=2; b=3
        a+b

Out[2]: 5
```

Here + is the operator that performs addition. a and b are the operands and 5 is the output of the operation

Arithmetic Operator

Arithmetic operators are used to perform mathematical operations like addition, subtraction, multiplication etc

Operator	Meaning	Example
+	Add two operands or unary plus	x + y +2
-	Subtract right operand from the left or unary minus	x - y -2
*	Multiply two operands	x * y
/	Divide left operand by the right one (always results into float)	x / y
%	Modulus - remainder of the division of left operand by the right	x % y (remainder of x/y)
//	Floor division - division that results into whole number adjusted to the left in the number line	x // y
**	Exponent - left operand raised to the power of right	x**y (x to the power y)

```
In [3]: x = 15
        y = 4

        # Output: x + y = 19
        print('x + y =',x+y)

        # Output: x - y = 11
        print('x - y =',x-y)

        # Output: x * y = 60
        print('x * y =',x*y)

        # Output: x / y = 3.75
        print('x / y =',x/y)

        # Output: x // y = 3
        print('x // y =',x//y)

        # Output: x ** y = 50625
        print('x ** y =',x**y)
```

```
('x + y =', 19)
('x - y =', 11)
('x * y =', 60)
('x / y =', 3)
('x // y =', 3)
('x ** y =', 50625)
```

Logical Operator

Logical operators are the `and`, `or`, `not` operators. Each of them operates on one or more bool type and returns a bool type

Operator	Meaning	Example
And	True if both the operands are true	x and y
Or	True if either of the operands is true	x or y
Not	True if operand is false (complements the operand)	not x

```
In [5]: x = True
        y = False

        # Output: x and y is False
        print('x and y is',x and y)

        # Output: x or y is True
        print('x or y is',x or y)

        # Output: not x is False
        print('not x is',not x)
```

```
('x and y is', False)
('x or y is', True)
('not x is', False)
```

Relation Operator (Comparison operator)

Comparison operators are used to compare values. It either returns `True` or `False` according to the condition.

Operator	Meaning	Example
>	Greater that - True if left operand is greater than the right	x > y
<	Less that - True if left operand is less than the right	x < y
==	Equal to - True if both operands are equal	x == y
!=	Not equal to - True if operands are not equal	x != y
>=	Greater than or equal to - True if left operand is greater than or equal to the right	x >= y
<=	Less than or equal to - True if left operand is less than or equal to the right	x <= y

```
In [4]: y = 12

        # Output: x > y is False
        print('x > y  is',x>y)

        # Output: x < y is True
        print('x < y  is',x<y)

        # Output: x == y is False
        print('x == y is',x==y)

        # Output: x != y is True
        print('x != y is',x!=y)

        # Output: x >= y is False
        print('x >= y is',x>=y)

        # Output: x <= y is True
        print('x <= y is',x<=y)
```

```
('x > y  is', True)
('x < y  is', False)
('x == y is', False)
('x != y is', True)
('x >= y is', True)
('x <= y is', False)
```

Assignment operators

Assignment operators are used in Python to assign values to variables.

`a = 5` is a simple assignment operator that assigns the value 5 on the right to the variable `a` on the left.

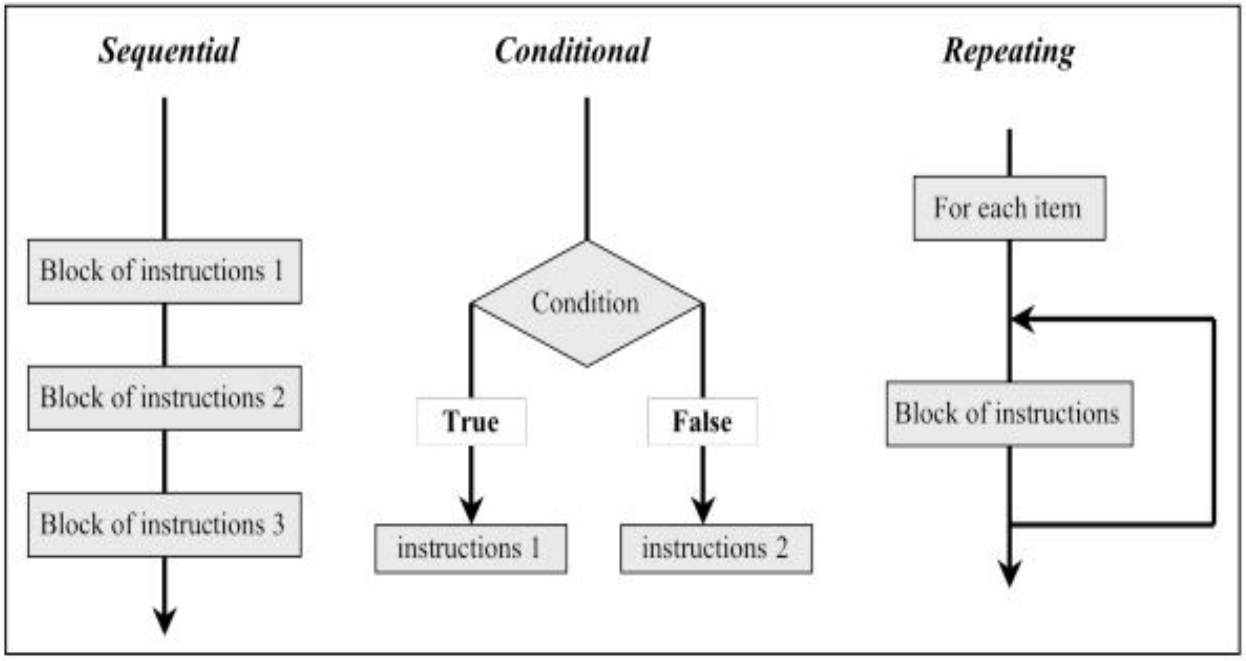
There are various compound operators in Python like `a += 5` that adds to the variable and later assigns the same. It is equivalent to `a = a + 5`.

Operator	Example	Equivatent to
=	x = 5	x = 5
+=	x += 5	x = x + 5
-=	x -= 5	x = x - 5
*=	x *= 5	x = x * 5

/=	x /= 5	x = x / 5
%=	x %= 5	x = x % 5
//=	x //= 5	x = x // 5
**=	x **= 5	x = x ** 5

Control Structures

Algorithms require two important control structures: iteration and selection. Both of these are supported by Python in various forms. The programmer can choose the statement that is most useful for the given circumstance.



1. Selection Statements/control Statement

1. If Statement

In Python there is only one kind of selection structure: the if statement. There are no switch, cases or whatever other selection structure you know in another language. The most fundamental control structure is the if structure. It is used to protect a block of code that only needs to be executed if a prior condition is met (i.e. is TRUE). The generate format of an if statement is:

```
>>> if condition:
        code block
```

The condition is one of the logical expressions we have seen in the previous section. The code block is a set of instructions grouped together. The code block is only executed if the condition is TRUE.

As example, let us write a small program that asks the user to enter a number, reads it in, and checks if it is divisible by n, where n is also read in:

```
number=int(raw_input("Enter your number  -->  "))
n=int(raw_input("Test divisibility by -->  "))
i=number%n
if i != 0:
        print "The number ",number," is not divisible by ",n,"\n"
```

2. Nested if-else

Selection constructs, as with any control construct, can be nested so that the result of one question helps decide whether to ask the next. For example, assume that score is a variable holding a reference to a score for a computer science test.

```
if score >= 90:
   print('A')
else:
   if score >=80:
      print('B')
   else:
      if score >= 70:
         print('C')
      else:
         if score >= 60:
            print('D')
         else:
            print('F')
```

This fragment will classify a value called score by printing the letter grade earned. If the score is greater than or equal to 90, the statement will print A. If it is not (else), the next question is asked. If the score is greater than or equal to 80 then it must be between 80 and 89 since the answer to the first question was false. In this case print B is printed. You can see that the Python indentation pattern helps to make sense of the association between if and else without requiring any additional syntactic elements.

3. If-Elif-Else Statement

An alternative syntax for this type of nested selection uses the elif keyword.
The else and the next if are combined so as to eliminate the need for additional nesting levels. Note that the final else is still necessary to provide the default case if all other conditions fail.

```
if score >= 90:
   print('A')
elif score >=80:
   print('B')
elif score >= 70:
   print('C')
elif score >= 60:
   print('D')
else:
   print('F')
```

2. Iteration

For iteration, Python provides a standard for and while statement

1. For Loop

In Python, there is no C style for loop, i.e., for (i=0; i<n; i++). There is “for in” loop which is similar to for each loop in other languages.

A common use of the for statement is to implement definite iteration over a range of values. The statement

```
>>> for item in range(5):
...     print(item**2)
...
0
1
4
9
16
>>>
```

2. While Loop

The while statement repeats a body of code as long as a condition is true.
For example

```
>>> counter = 1
>>> while counter <= 5:
...     print("Hello, world")
...     counter = counter + 1


Hello, world
Hello, world
Hello, world
Hello, world
Hello, world
```

Prints out the phrase “Hello, world” five times. The condition on the while statement is evaluated at the start of each repetition. If the condition is True, the body of the statement will execute. It is easy to see the structure of a Python while statement due to the mandatory indentation pattern that the language enforces.

Here is another program that prints the number between 0 and N, where N is input:

```
>>> N=int(raw_input(“Enter N --> “))
>>> print “Counting numbers from 0 to “,N,”\n”
 i=0
while  i < N+1:
        print i,”\n”
        i+=1
```

Note that it is important to make sure that the code block includes a modification of the test: if we had forgotten the line i+=1 in the example above, the while loop would have become an infinite loop. Note that any for loop can be written as a while loop. In practice however, it is better to use a for loop, as Python executes them faster

Break points in control Structures

Using for loops and while loops in Python allow you to automate and repeat tasks in an efficient manner.

But sometimes, an external factor may influence the way your program runs. When this occurs, you may want your program to exit a loop completely, skip part of a loop before continuing, or ignore that external factor. You can perform these actions with break, continue, and pass statements.

Break Statement

In Python, the **break** statement provides you with the opportunity to exit out of a loop when an external condition is triggered. You’ll put the break statement within the block of code under your loop statement, usually after a **conditional if statement**.

Let's look at an example that uses the **break** statement in a **for loop**:

```
In [1]: number = 0

        for number in range(10):
           number = number + 1

           if number == 5:
              break    # break here

           print('Number is ' + str(number))

        print('Out of loop')
```

```
Number is 1
Number is 2
Number is 3
Number is 4
Out of loop
```

Continue Statement

The **continue** statement gives you the option to skip over the part of a loop where an external condition is triggered, but to go on to complete the rest of the loop. That is, the current iteration of the loop will be disrupted, but the program will return to the top of the loop.

The **continue** statement will be within the block of code under the loop statement, usually after a conditional **if** statement.

Using the same **for** loop program as in the Break Statement section above, we'll use a **continue** statement rather than a **break** statement:

In [2]:
```
number = 0

for number in range(10):
   number = number + 1

   if number == 5:
      continue    # continue here

   print('Number is ' + str(number))

print('Out of loop')
```

```
Number is 1
Number is 2
Number is 3
Number is 4
Number is 6
Number is 7
Number is 8
Number is 9
Number is 10
Out of loop
```

The **break & continue** statements in Python will allow you to use **for loops and while loops** more **effectively** in your code.