UNIT-II

Type conversion

The process of converting the value of one data type (integer, string, float, etc.) to another data type is called type conversion. Python has two types of type conversion.

1. Implicit Type Conversion
2. Explicit Type Conversion

1. Implicit Type Conversion:

In Implicit type conversion, Python automatically converts one data type to another data type. This process doesn't need any user involvement.

Let's see an example where Python promotes conversion of lower datatype (integer) to higher data type (float) to avoid data loss.

Addition of float(higher) data type and integer(lower) datatype

```
num_int = 120
num_flo = 1.25

num_new = num_int + num_flo

print("Value of num_new:",num_new)

print("datatype of num_new:",type(num_new))
```

In the above program,

- We add two variables `num_int` and `num_flo`, storing the value in `num_new`.
- We will look at the data type of all three objects respectively.
- In the output we can see the datatype of `num_int` is an `integer`, datatype of `num_flo` is a `float`.
- Also, we can see the `num_new` has `float` data type because Python always converts smaller data type to larger data type to avoid the loss of data.

Addition of string(higher) data type and integer(lower) datatype

```
num_int = 123
num_str = "456"

print("Data type of num_int:",type(num_int))
print("Data type of num_str:",type(num_str))


print(num_int+num_str)
```

In the above program,

- We add two variable num_int and num_str.
- As we can see from the output, we got typeerror. Python is not able use Implicit Conversion in such condition.
- However, Python has the solution for this type of situation which is known as Explicit Conversion.

2.Explicit Type Conversion

In Explicit Type Conversion, users convert the data type of an object to required data type. We use the predefined functions like int(), float(), str(), etc to perform explicit type conversion.

This type conversion is also called typecasting because the user casts (change) the data type of the objects.

Syntax: `(required_datatype)(expression)`

Typecasting can be done by assigning the required data type function to the expression.

```
num_int = 100
num_str = "456"

print("Data type of num_str before Type Casting:",type(num_str))

new_num_str = int(num_str)
print("Data type of num_str after Type Casting:",type(new_num_str))

num_sum = num_int + new_num_str

print("Sum of num_int and num_str:",num_sum)

print("Data type of the sum:",type(num_sum))
```

In above program,

- We add num_str and num_int variable.
- We converted num_str from string(higher) to integer(lower) type using int() function to perform the addition.
- After converting num_str to a integer value Python is able to add these two variable.
- We got the num_sum value and data type to be integer.

Key Points to Remember

1. Type Conversion is the conversion of object from one data type to another data type.
2. Implicit Type Conversion is automatically performed by the Python interpreter.
3. Python avoids the loss of data in Implicit Type Conversion.
4. Explicit Type Conversion is also called Type Casting, the data types of object are converted using predefined function by user.
5. In Type Casting loss of data may occur as we enforce the object to specific data type.

Keywords

- Keywords are the reserved words in Python. We cannot use a keyword as variable name, function name or any other identifier.
- They are used to define the syntax and structure of the Python language.
- In Python, keywords are case sensitive. All the keywords except True, False and None are in lowercase and they must be written as it is.

Identifiers

- Identifier is the name given to entities like **class**, **functions**, **variables** etc. in Python.
- It helps in differentiating one entity from another.
- Rules
    - Identifiers can be a combination of letters in lowercase (a to z) or uppercase (A to Z) or digits (0 to 9) or an underscore (_).
        - Names like myClass, var_1 and print_this_to_screen all are valid example.
    - An identifier cannot start with a digit.
        - 1variable is invalid, but variable1 is perfectly fine.
    - Keywords cannot be used as identifiers.
        - global = 5
    - We cannot use special symbols like !, @, #, $, % etc. in our identifier.
        - name@pet = “sweety”

Things to be considered while programming

- Python is a case-sensitive language. This means, Variable and variable are not the same.
- Always better to provide proper name to identifiers based on its purpose. Though c = 10 is a valid, its better in writing count = 10. It would be easier to figure out what it does even when you look at your code after a long gap.
- Multiple words can be separated using an underscore.
    - this_is_a_long_variable its_a_long_method()
- We can also use camel-case style of writing, i.e., capitalize every first letter of the word except the initial word without any spaces.
    - camelCaseExample writeIntoExcel()

Control Statements

Decision making is required when we want to execute a code only if a certain condition is satisfied.

The `if...elif...else` statement is used in Python for decision making.

Python if Statement

```
if test expression:
    statement(s)
```

Here, the program evaluates the `test expression` and will execute statement(s) only if the text expression is `True`.

If the text expression is `False`, the statement(s) is not executed.

In Python, the body of the `if` statement is indicated by the indentation. Body starts with an indentation and the first unindented line marks the end.

Python interprets non-zero values as `True`. `None` and `0` are interpreted as `False`.

```
num = 100
if num > 0:
    print(num, "is a positive number.")
print("This is always printed.")
```

Python if...else Statement

```
if test expression:
    Body of if
else:
    Body of else
```

The `if..else` statement evaluates `test expression` and will execute body of `if` only when test condition is `True`.

If the condition is `False`, body of `else` is executed. Indentation is used to separate the blocks.

```
if num >= 0:
    print("Positive or Zero")
else:

    print("Negative number")
```

In the above example, when num is equal to 3, the test expression is true and body of if is executed and body of else is skipped.

If num is equal to -5, the test expression is false and body of else is executed and body of if is skipped.

If num is equal to 0, the test expression is true and body of if is executed and body of else is skipped.

Python if...elif...else Statement

```
if test expression:
    Body of if
elif test expression:
    Body of elif
else:
    Body of else
```

The elif is short for else if. It allows us to check for multiple expressions.

If the condition for if is False, it checks the condition of the next elif block and so on.

If all the conditions are False, body of else is executed.

Only one block among the several if...elif...else blocks is executed according to the condition.

The if block can have only one else block. But it can have multiple elif blocks.

```
if num > 0:
    print("Positive number")
elif num == 0:
    print("Zero")
else:

    print("Negative number")
```

When variable num is positive, Positive number is printed.

If num is equal to 0, Zero is printed.

If num is negative, Negative number is printed

Python Nested if statements

We can have a if...elif...else statement inside another if...elif...else statement. This is called nesting in computer programming.

Any number of these statements can be nested inside one another. Indentation is the only way to figure out the level of nesting. This can get confusing, so must be avoided if we can.

```
num = 10

if num >= 0:
      if num%2==0 :
            print "Positive even number"
      else:
            print("Positive odd number")
else:
      if num%2==0 :
            print("Negative even number")
      else:

            print("Negative odd number")
```

Check with different inputs for num and observe the control.

for loops

The for loop in Python is used to iterate over a sequence (list, tuple, string) or other iterable objects. Iterating over a sequence is called traversal.

Syntax of for Loop

```
for val in sequence:
      Body of for
```

Here, `val` is the variable that takes the value of the item inside the sequence on each iteration.

Loop continues until we reach the last item in the sequence. The body of for loop is separated from the rest of the code using indentation.

```
numbers = [6, 5, 3, 8, 4, 2, 5, 4, 11]
sum = 0
for val in numbers:
      sum = sum+val
print("The sum is", sum)
```

for loop with else

A for loop can have an optional else block as well. The `else` part is executed if the items in the sequence used in for loop exhausts.

break statement can be used to stop a for loop. In such case, the else part is ignored.

Hence, a for loop's else part runs if no break occurs.

```
my_dict = {"name":"raja", "id":105, ("maths","science","english"):[85, 78,
91], "address":{"door_number": "32-5", "city":"Bangalore",
"state":"Karnataka"}}
for key in my_dict:
      print key
      print my_dict[key]
else:

      print "No items left"
```

while loops

The while loop in Python is used to iterate over a block of code as long as the test expression (condition) is true.

We generally use this loop when we don't know beforehand, the number of times to iterate.

Syntax of while Loop

```
while test_expression:
    Body of while
```

In while loop, test expression is checked first. The body of the loop is entered only if the `test_expression` evaluates to `True`. After one iteration, the test expression is checked again. This process continues until the `test_expression` evaluates to `False`.

In Python, the body of the while loop is determined through indentation.

Body starts with indentation and the first unindented line marks the end.

Python interprets any non-zero value as `True`. `None` and `0` are interpreted as `False`.

```
number = 4
count = sum = 0
while(count <=number):
      sum = sum + count
      count +=1
print sum
```

How the program executes

Iteration #	count	Condition & value	sum	count after
1	0	0<=4 → True	0+0 → 0	1
2	1	1<=4 → True	0+1 → 1	2
3	2	2<=4 → True	1+2 → 3	3
4	3	3<=4 → True	3+3 → 6	4
5	4	4<=4 → True	6+4 → 10	5
6	5	5<=4 → **False**		

break, continue & pass

In Python, break and continue statements can alter the flow of a normal loop.

Loops iterate over a block of code until test expression is false, but sometimes we wish to terminate the current iteration or even the whole loop without checking test expression.

break

The break statement terminates the loop containing it. Control of the program flows to the statement immediately after the body of the loop.

If break statement is inside a nested loop (loop inside another loop), break will terminate the innermost loop.

```
for char in 'PYTHON STRING':
      if char == 'H':
            break
      print(char)
print("The end")
```

Output of above program will be

```
P
Y
T
The end
```

we iterate through the "PYTHON STRING" sequence. We check if the letter is "H", upon which we break from the loop. Hence, we see in our output that all the letters up till "H" gets printed. After that, the loop terminates.

continue

The continue statement is used to skip the rest of the code inside a loop for the current iteration only. Loop does not terminate but continues with the next iteration.

```
for char in 'PYTHON':
      if char == 'O':
            continue
      print(char)
print("The end")
```

Output of above program will be

```
P
Y
T
H
N
The end
```

We continue with the loop, if the string is "O", not executing the rest of the block. Hence, we see in our output that all the letters except "O" gets printed.

pass

In Python programming, pass is a null statement. The difference between a comment and pass statement in Python is that, while the interpreter ignores a comment entirely, pass is not ignored.

However, nothing happens when pass is executed. It results into no operation.

We generally use it as a placeholder.

Suppose we have a loop or a function that is not implemented yet, but we want to implement it in the future. They cannot have an empty body. The interpreter would complain. So, we use the pass statement to construct a body that does nothing.

```
Example 1
def function(args):
    pass # yet to implement

Example 2
sequence = {'p', 'a', 's', 's'}
for val in sequence:

    pass # yet to implement
```