Comments

```
print('hello world') # Note that print is a function
```

```
# Note that print is a function
print('hello world')
```

Numbers

Numbers are mainly of two types - integers and floats.

An example of an integer is `2` which is just a whole number.

Examples of floating point numbers (or *floats* for short) are `3.23` and `52.3E-4`. The `E` notation indicates powers of 10. In this case, `52.3E-4` means `52.3 * 10^-4^`.

> **Note for Experienced Programmers**
>
> There is no separate `long` type. The `int` type can be an integer of any size.

Strings

Single Quote

```
'Quote me on this'
```

Double Quotes

“Quote me on this”

Triple Quotes

```
'''This is a multi-line string. This is the first line.
This is the second line.
"What's your name?," I asked.
He said "Bond, James Bond."
'''
```

Strings Are Immutable

This means that once you have created a string, you cannot change it. Although this might seem like a bad thing, it really isn't. We will see why this is not a limitation in the various programs that we see later on.

> **Note for C/C++ Programmers**
>
> There is no separate `char` data type in Python. There is no real need for it and I am sure you won't miss it.

> **Note for Perl/PHP Programmers**
>
> Remember that single-quoted strings and double-quoted strings are the same - they do not differ in any way.

The format method

file `str_format.py` :

```
age = 20

name = 'Swaroop'


print('{0} was {1} years old when he wrote this book'.format(name, age))

print('Why is {0} playing with that python?'.format(name))
```

Output:

```
$ python str_format.py

Swaroop was 20 years old when he wrote this book

Why is Swaroop playing with that python?
```

The format method


```
# decimal (.) precision of 3 for float '0.333'
print('{0:.3f}'.format(1.0/3))
# fill with underscores (_) with the text centered
# (^) to 11 width '___hello___'
print('{0:_^11}'.format('hello'))
# keyword-based 'Swaroop wrote A Byte of Python'
print('{name} wrote {book}'.format(name='Swaroop', book='A Byte of Python'))
```

Output:

```
0.333
___hello___
Swaroop wrote A Byte of Python
```

print

`print` always ends with an invisible "new line" character (`\n`)

`end` with a blank:

```
print('a', end='')
print('b', end='')
```

Output is:

```
ab
```

`end` with a space:

```
print('a', end=' ')
print('b', end=' ')
print('c')
```

Output is:

```
a b c
```

Strings
Escape Sequences

a string which contains a single quote (')

```
"What's your name?"
```

~~'What's your name?'~~

```
'What\'s your name?'
```

the newline character - `\n`

```
'This is the first line\nThis is the second line'
```

string is continued in the next line

```
"This is the first sentence. \
This is the second sentence."
```

is equivalent to

```
"This is the first sentence. This is the second sentence."
```

the tab: `\t`

Raw String

prefixing `r` or `R`

```
r"Newlines are indicated by \n"
```

Note for Regular Expression Users

Always use raw strings when dealing with regular expressions. Otherwise, a lot of backwhacking may be required. For example, backreferences can be referred to as `'\\1'` or `r'\1'`.

Variables And Literal Constants

```
# Filename : var.py
i = 5
print(i)
i = i + 1
print(i)

s = '''This is a multi-line string.
This is the second line.'''
print(s)
```

Output:

```
5
6
This is a multi-line string.
This is the second line.
```

Variable

Identifier Naming

Object

Datatypes

Logical And Physical Line

semicolon (;) which indicates the end of a logical line/statement.

```
i = 5
print(i)
```

```
i = 5;
print(i);
```

```
i = 5; print(i);
```

```
i = 5; print(i)
```

break it into multiple physical lines by using the backslash.

```
s = 'This is a string. \
This continues the string.'
print(s)
```

Output:

```
This is a string. This continues the string.
```

```
i = \
5
```

```
i = 5
```

Indentation

whitespace at the beginning of the line is important.

```
i = 5
# Error below! Notice a single space at the start of the line
 print('Value is', i)
print('I repeat, the value is', i)
```

```
   File "whitespace.py", line 3
     print('Value is', i)
     ^
 IndentationError: unexpected indent
```

How to indent

Use four spaces for indentation. This is the official Python language recommendation. Good editors will automatically do this for you. Make sure you use a consistent number of spaces for indentation, otherwise your program will not run or will have unexpected behavior.