Python Program Structure

Mr. Prakash C O
Asst. Professor,
Dept. of CSE, PESU
coprakasha@pes.edu

Python Program Structure

- Python Statements
- Line Continuation
  - Implicit Line Continuation
  - Explicit Line Continuation
- Multiple Statements Per Line
- Comments
- Whitespace
- Whitespace as Indentation
- Case Sensitive

Python Statements

- Statements are the basic units of instruction that the Python interpreter parses and processes.
- **Python programs are typically organized with one statement per line**.

  In other words, **each statement occupies a single line, with the end of the statement delimited by the newline character that marks the end of the line**.
- **All statements** unless the special construct is used **should start from the first column**. If the rule is not followed, the translator gives an error.

Line Continuation

- As code becomes more complex, statements will unavoidably grow long. To maintain readability, you should break them up into parts across several lines.

- **In Python code, a statement can be continued from one line to the next in two different ways:**
  - **Implicit line continuation and**
  - **Explicit line continuation.**

Line Continuation

- **Implicit Line Continuation**
  - Any statement containing opening parentheses ('('), brackets ('['), or curly braces ('{') is presumed to be incomplete until all matching parentheses, brackets, and braces have been encountered. Until then, the statement can be implicitly continued across lines without raising an error.
  - Examples

```
>>> a = [
...     [1, 2, 3, 4, 5],
...     [6, 7, 8, 9, 10],
...     [11, 12, 13, 14, 15],
...     [16, 17, 18, 19, 20],
...     [21, 22, 23, 24, 25]
... ]
```

Line Continuation

- **Implicit Line Continuation**
  - Examples

```
>>> x = (
...     1 + 2
...     + 3 + 4
...     + 5 + 6
... )
>>> x
21
```

```
>>> print(
...     'foo',
...     'bar',
...     'baz'
... )
foo bar baz
```

```
>>> x1 = {
...     'foo',
...     'bar',
...     'baz'
... }
```

```
>>> a = [
...     'foo', 'bar',
...     'baz', 'qux'
... ]
```

Line Continuation

➢ Explicit Line Continuation

- To indicate explicit line continuation, you can specify a backslash (\) character as the final character on the line
- Note that the backslash character must be the last character on the line. Not even whitespace is allowed after it.

```
>>> s = \
... 'Hello, World!'
>>> s
'Hello, World!'

>>> x = 1 + 2 \
...     + 3 + 4 \
...     + 5 + 6
>>> x
21
```

Multiple Statements Per Line

- Multiple statements may occur on one line, if they are separated by a semicolon (;) character:

```
>>> x = 1; y = 2; z = 3
>>> print(x); print(y); print(z)
1
2
3
```

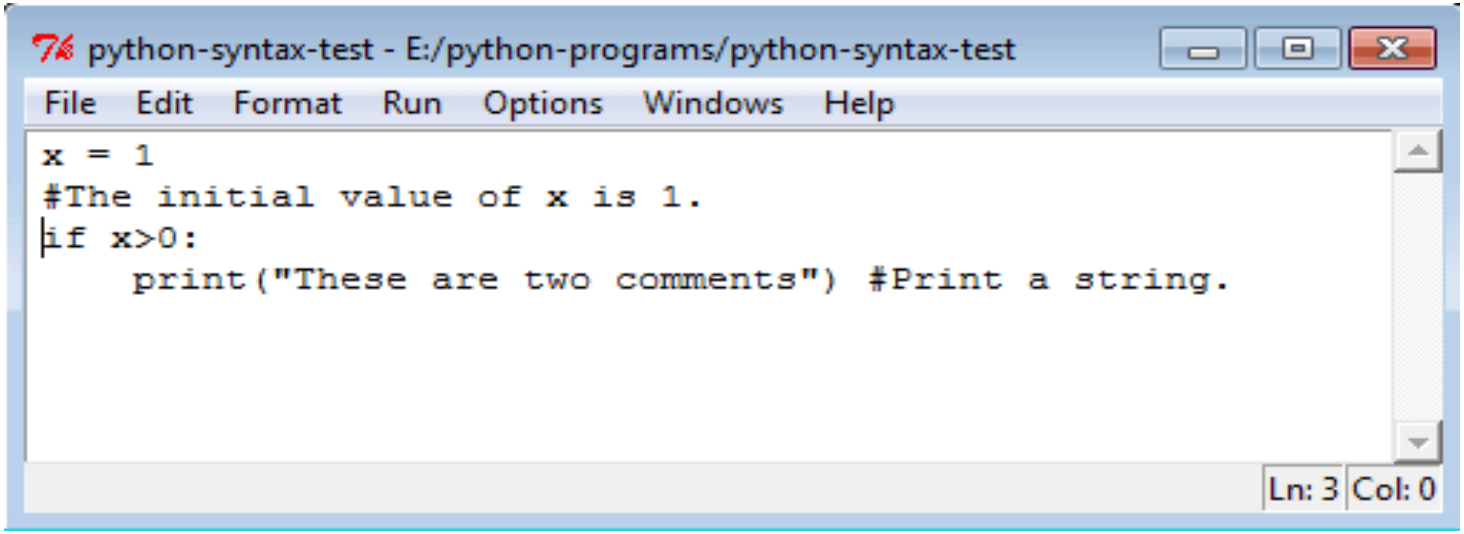
- The following statements are functionally equivalent to the example above, but would be considered more typical Python code:

```
>>> x, y, z = 1, 2, 3
>>> print(x, y, z, sep='\n')
1
2
3
```

Comments

- **Single line comments**
  - **Single line comment begins with a hash character(#) which is not a part of the string literal and ends at the end of the physical line.**

    All characters after the # character up to the end of the line are part of the comment and the Python interpreter ignores them.



Comments

- **Multi-line comments**
  - **Using Multi-line Strings as Comments**
  - Another option for writing “proper” multi-line comments in Python is to use multi-line strings with triple quotes.
  - Here’s an example:

```
"""
This is a "block comment" in Python, made
out of a mult-line string constant.
This actually works quite well!
"""
```

- Triple quotes are treated as regular strings with the exception that they can span multiple lines. By regular strings I mean that if they are not assigned to a variable they will be immediately garbage collected as soon as that code executes. hence are **not ignored** by the interpreter in the same way that #a comment is.

Whitespace

- When parsing code, the Python interpreter breaks up the input into tokens.
  Informally, tokens are just the language elements that you have seen so far: identifiers, keywords, literals, and operators.

- The most common whitespace characters are the following:

Character	ASCII Code	Literal Expression
space	32 (0x20)	' '
tab	9 (0x9)	'\t'
newline	10 (0xa)	'\n'

Whitespace

Note: You can juxtapose string literals, with or without whitespace:

Python >>>

```
>>> s = "foo"'bar''''baz'''
>>> s
'foobarbaz'

>>> s = 'foo' "bar" '''baz'''
>>> s
'foobarbaz'
```

The effect is concatenation, exactly as though you had used the + operator.

Whitespace as Indentation

- **Python uses whitespace (spaces and tabs) to define program blocks** whereas other languages like C, C++ use braces ({}) to indicate blocks of codes for class, functions or flow control.

- **The number of whitespaces (spaces and tabs) in the indentation is not fixed, but all statements within the block must be indented by the same amount of whitespaces.**

Whitespace as Indentation

- Indentation—whitespace that appears to the left of the first token on a line—has very special meaning.
- In most interpreted languages, leading whitespace before statements is ignored.
- In Python, indentation is not ignored. Leading whitespace is used to compute a line's indentation level, which in turn is used to determine grouping of statements.

Whitespace as Indentation

- Example:

```
if age > 18:
    print("adult person")

for i in range(5):
    print(i)
```

Case Sensitive

- **Python is a case sensitive language.**
- **Python distinguishes upper case and lower case characters.**
- **Most of the words we use in Python are in lower case.**
- **Example:**
  - **Line** is different from **line**
  - **P**rint("nalku") # gives an error.