Punctuations & Identifiers

In Python, punctuation characters are used for various syntactic and structural purposes. They help define operations, separate code blocks, and structure expressions. Python includes many common punctuation marks such as periods, commas, colons, and brackets.

Common Punctuation Symbols in Python

Symbol	Name	Usage Example
`.`	Dot / Period	Accessing attributes: `object.attribute`
`,`	Comma	Separating items in lists or function arguments: `a, b`
`:`	Colon	Used in function definitions, loops, and conditionals: `if x > 0:`
`;`	Semicolon	Separates multiple statements on one line: `a = 5; b = 10`
`'` or `"`	Single/Double Quotes	Defining strings: `'hello'` or `"world"`
`(` `)`	Parentheses	Function calls or grouping expressions: `print(x)`
`[` `]`	Square Brackets	Indexing lists or arrays: `my_list[0]`
`{` `}`	Curly Braces	Defining dictionaries or sets: `{"a": 1}`
`#`	Hash	Starts a comment: `# This is a comment`
`\`	Backslash	Used for escape characters or line continuation
`@`	At symbol	Decorators in functions or classes: `@staticmethod`
`=`	Equals sign	Assignment: `x = 10`
`+`, `-`, `*`, `/`	Arithmetic Operators	Used in expressions: `a + b`

String Punctuation Using `string.py` file

`path C:\Users\Neeraj\AppData\Local\Programs\Python\Python313\Lib`

```
"""A collection of string constants.

Public module variables:

whitespace -- a string containing all ASCII whitespace
ascii_lowercase -- a string containing all ASCII lowercase letters
ascii_uppercase -- a string containing all ASCII uppercase letters
```

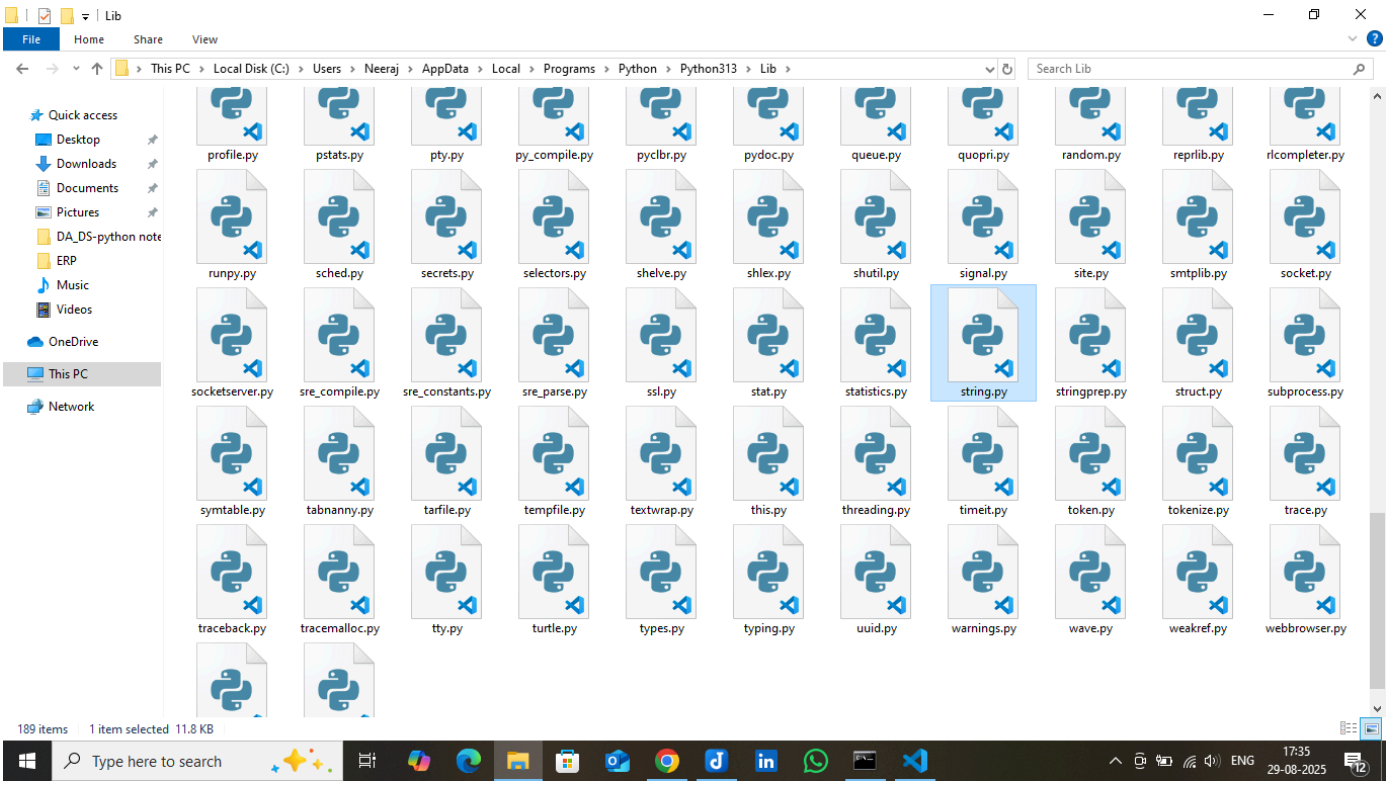
```
ascii_letters -- a string containing all ASCII letters
digits -- a string containing all ASCII decimal digits
hexdigits -- a string containing all ASCII hexadecimal digits
octdigits -- a string containing all ASCII octal digits
punctuation -- a string containing all ASCII punctuation characters
printable -- a string containing all ASCII characters considered printable

"""

__all__ = ["ascii_letters", "ascii_lowercase", "ascii_uppercase", "capwords",
           "digits", "hexdigits", "octdigits", "printable", "punctuation",
           "whitespace", "Formatter", "Template"]

import _string

# Some strings for ctype-style character classification
whitespace = ' \t\n\r\v\f'
ascii_lowercase = 'abcdefghijklmnopqrstuvwxyz'
ascii_uppercase = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
ascii_letters = ascii_lowercase + ascii_uppercase
digits = '0123456789'
hexdigits = digits + 'abcdef' + 'ABCDEF'
octdigits = '01234567'
punctuation = r"""!"#$%&'()*+,-./:;<=>?@[\]^_`{|}~"""
printable = digits + ascii_letters + punctuation + whitespace
```



if you want to get all the pre-define string variable's like whitespace, ascii_lowercase, ascii_uppercase , ascii_letters ,digits, hexdigits, octdigits, printable, punctuation, then you try below mention code:---

```
import string

all_punctuation = string.punctuation
print(all_punctuation)

# output
!"#$%&'()*+,-./:;<=>?@[\]^_`{|}~
```

Removing Punctuation from a String

```
import string

text = "Hello, world! Python!!! from cybrom#$ institute:)"
clean_text = ''.join(char for char in text if char not in string.punctuation)

print(clean_text)

# output
Hello world Python from cybrom institute
```

Why Punctuation Matters in Python

1. **Syntax Control**: Symbols like colons and indentation define control structures.
2. **Data Structures**: Brackets and braces define lists, dictionaries, and sets.
3. **Text Processing**: Punctuation often needs to be handled during natural language processing (NLP).

Identifiers

Identifiers in Python are names used to identify variables, functions, classes, modules, and other objects. An identifier is a sequence of one or more characters that may consist of letters (both uppercase and lowercase), digits (0-9), and underscores (_).

Rules for Naming Identifiers in Python :--

Start with a Letter or Underscore: An identifier must begin with a letter (a-z, A-Z) or an underscore (_). It cannot start with a digit.

```
# Example:--
xyz = 10
print(xyz)

# Example:--
```

```
Xyz = 20
print(Xyz)

# Example:--
_xyz = 30
print(_xyz)

# Output
10
20
30
```

```
# Example:--
1xyz = 10
print(xyz)

# output
ERROR!
Traceback (most recent call last):
  File "<main.py>", line 1
    1xyz = 10
    ^
SyntaxError: invalid decimal literal

# Example:--
$xyz = 30
print(_xyz)

# output
ERROR!
Traceback (most recent call last):
  File "<main.py>", line 4
    $xyz = 30
    ^
SyntaxError: invalid syntax
```

Subsequent Characters: The characters following the initial letter or underscore can be letters, digits, or underscores.

```
_xy123 = 10
print(_xy123)

# Output:
10
```

```
_xy$123 = 10
print(_xy$123)

# Output:
ERROR!
Traceback (most recent call last):
  File "<main.py>", line 1
    _xy$123 = 10
       ^
SyntaxError: invalid syntax
```

Case Sensitivity: Identifiers in Python are case-sensitive. This means myVariable, MyVariable, and myvariable are considered three different identifiers.

```
# Example:--
x = 10
print(x)

# output
10

print(X)
# output
ERROR!
Traceback (most recent call last):
  File "<main.py>", line 4, in <module>
NameError: name 'X' is not defined
```

No Spaces or Special Characters: Identifiers cannot contain spaces or special characters like !, @, #, $, %, etc., except for the underscore (_).

```
# Example:--
x_y =10
print(x_y)

# Output
10

# Example:--
x$y=10
print(x$y)

# Output
ERROR!
```

```
Traceback (most recent call last):
  File "<main.py>", line 1
    x$y=10
     ^
SyntaxError: invalid syntax
```

No Keywords: Identifiers cannot be the same as Python keywords. Keywords are reserved words in Python that have predefined meanings, such as if, else, while, for, def, class, etc. You can check the list of keywords using the keyword module.

```
# Example:--
if = 10
print(if)

# Output
ERROR!
Traceback (most recent call last):
  File "<main.py>", line 1
    if = 10
       ^
SyntaxError: invalid syntax
```

No Built-in Function Names: It is not advisable (though technically possible) to use the names of built-in Python functions and modules (like print, list, str, int, etc.) as identifiers, as this can lead to confusion and bugs.

```
# Example 1:--
print = 10
print(print)

# Output
ERROR!
Traceback (most recent call last):
  File "<main.py>", line 2, in <module>
TypeError: 'int' object is not callable
```