

notes

September 29, 2023

1 Comments

```
[1]: # This is a comment
    '''
    This is also a comment
    '''

    print('Hello World')
```

Hello World

2 Multiple statements in single line

```
[2]: a = 3; print(a)
```

3

3 Multiline statements

```
[3]: a = 1 + 3 + 4 + \
      5 + 6 + 7
      print(a)
```

26

4 Multiple variable assignments

```
[4]: a,b = 4,5
      print(a,b)
```

4 5

5 help() & __doc__

```
[5]: def my_func():  
      '''This is a doc-string'''  
      pass  
  
      help(my_func)  
      print('-'*100)  
      print(my_func.__doc__)
```

Help on function my_func in module __main__:

```
my_func()  
    This is a doc-string
```

```
-----  
-----
```

This is a doc-string

6 print() and input()

```
[6]: print?
```

Signature: print(*args, sep=' ', end='\n', file=None, flush=False)

Docstring:

Prints the values to a stream, or to sys.stdout by default.

sep

string inserted between values, default a space.

end

string appended after the last value, default a newline.

file

a file-like object (stream); defaults to the current sys.stdout.

flush

whether to forcibly flush the stream.

Type: builtin_function_or_method

```
[7]: input?
```

Signature: input(prompt='')

Docstring:

Forward raw_input to frontends

Raises

```
-----
```

StdinNotImplementedError if active frontend doesn't support stdin.

File: /usr/local/lib/python3.11/site-packages/ipykernel/kernelbase.py

Type: method

7 Types of statements

7.0.1 Empty Statements

- Empty statements are also known as “pass statements”
- They serve as a placeholder and do nothing when executed
- The `pass` keyword is used to create an empty statement
- Commonly used as a temporary placeholder when writing code or as a stub for future implementation

7.0.2 Simple Statements

- Simple statements are single-line statements that perform a specific action or operation
- They typically end with a newline character or a semicolon
- Common examples include assignment statements, function calls, and print statements

7.0.3 Compound Statements (Block Statements)

- Compound statements consist of one or more simple statements grouped together into a block
- They are often used to control program flow and define structures like loops and conditional statements
- Compound statements are defined using indentation (whitespace) in Python
- Common examples include if statements, while loops, for loops, and function definitions

```
[8]: # Empty statement
def foo():
    pass # Placeholder for function implementation

# Simple statements
x = 5 # Assignment statement
print("Hello, World!") # Print statement
result = min(2, 3) # Function call and assignment

# Compound statements
if x > 0:
    print("x is positive") # Indented block as part of the if statement
else:
    print("x is not positive") # Indented block as part of the else statement

while x < 5:
    print(x) # Indented block as part of the while loop
    x += 1

def greet(name):
    print(f"Hello, {name}!") # Indented block as part of the function
    ↪ definition
```

```
Hello, World!  
x is positive
```

8 Switch/Match

```
[9]: x = 1  
  
match x:  
    case 1: print("x is 1")  
    case 2: print("x is 2")  
    case 3: print("x is 3")  
    case _: print("x is something else")
```

```
x is 1
```

9 String fuctions

(https://www.w3schools.com/python/python_strings_methods.asp)

9.0.1 isprintable()

Basically if all characters are printable. Non-printable characters are things starting with \ like \n, \t, or \x07

9.0.2 isnumeric() vs isdigit()

isnumeric() is more inclusive than isdigit() and can handle a wider range of numeric representations. For eg, 123½ will be returned True by isnumeric() but not by isdigit()

9.0.3 find() vs index()

find() returns -1 if the substring wasn't found while index() raises a ValueError

9.0.4 split() vs rsplit()

- split() splits the string from left to right, while rsplit() splits it from right to left.
- By default, both methods return all substrings, but you can use the maxsplit parameter to limit the number of splits. When using rsplit(), the remaining part of the string appears as the first element in the result list.

9.0.5 casefold() vs lower()

- Use lower() when you need to perform simple case-insensitive operations and you are working with characters primarily from the Latin alphabet.
- Use casefold() when you need robust case-insensitive operations, especially for internationalization and localization, or when dealing with characters from various languages and scripts.

casefold() is generally recommended for case-insensitive string comparisons in most applications, as it provides more consistent and accurate results across a wide range of characters and languages.

9.0.6 partition() vs split()

`s.partition()` and `s.split()` are two string methods in Python used for splitting a string based on a specified delimiter. However, they work slightly differently:

1. `s.partition(delimiter)`:
 - The `s.partition(delimiter)` method splits the string `s` into three parts based on the first occurrence of the specified `delimiter`
 - It returns a tuple containing three elements: the part of the string before the delimiter, the delimiter itself, and the part of the string after the delimiter
 - If the delimiter is not found in the string, the method returns a tuple with the original string as the first element, followed by two empty strings
 2. `s.split(delimiter, maxsplit)`:
 - The `s.split(delimiter, maxsplit)` method splits the string `s` into a list of substrings based on the specified `delimiter`
 - You can optionally specify the `maxsplit` parameter to control the maximum number of splits to perform
 - If the delimiter is not found in the string, the method returns a list with the entire string as a single element
- `partition()` always returns a tuple with three elements, whereas `split()` returns a list of substrings
 - `partition()` splits the string only at the first occurrence of the delimiter, while `split()` can split the string at multiple occurrences
 - `split()` provides more flexibility with the `maxsplit` parameter, allowing you to limit the number of splits

```
[19]: x = 'hello World 123!'
x_t = 'hello\tWorld\t123!'
x_l = 'hello\nWorld\n123!'

print('1 ', x.capitalize())           # Capitalize the first character of
    ↪ the string
print('2 ', x.casefold())              # Perform case-folding on the string
print('3 ', x.center(20, '-'))         # Center-align the string within a
    ↪ width of 20, padding with '-'
print('4 ', x.count('o'))              # Count instances of substring 'o'
print('5 ', x.endswith('123!'))       # Check if the string ends with '123!'
print('6 ', x_t.expandtabs(10))        # Expand tab characters to spaces with
    ↪ tab stops at 10
print('7 ', x.find('o'))               # Find the first occurrence of 'o'
print('8 ', x.index('o'))              # Get the index of the first
    ↪ occurrence of 'o'
print('9 ', x.isalnum())               # Check if all characters are
    ↪ alphanumeric
print('10', x.isalpha())               # Check if all characters are
    ↪ alphabetic
print('11', x.isascii())              # Check if the string is ASCII
```

<code>print('12', x.isdecimal())</code>	<i># Check if all characters are decimal</i>
<i>↳ digits</i>	
<code>print('13', x.isdigit())</code>	<i># Check if all characters are digits</i>
<code>print('14', x.isidentifier())</code>	<i># Check if the string is a valid</i>
<i>↳ identifier</i>	
<code>print('15', x.islower())</code>	<i># Check if all characters are lowercase</i>
<code>print('16', x.isnumeric())</code>	<i># Check if all characters are numeric</i>
<code>print('17', x.isprintable())</code>	<i># Check if the string is printable</i>
<code>print('18', x.isspace())</code>	<i># Check if all characters are</i>
<i>↳ whitespace</i>	
<code>print('19', x.istitle())</code>	<i># Check if the string is titlecased</i>
<code>print('20', x.isupper())</code>	<i># Check if all characters are uppercase</i>
<code>print('21', x.ljust(30, '-'))</code>	<i># Left-justify the string within a</i>
<i>↳ width of 30, padding with '-'</i>	
<code>print('22', x.lower())</code>	<i># Convert the string to lowercase</i>
<code>print('23', x.lstrip('H'))</code>	<i># Remove leading 'H' characters from</i>
<i>↳ the string</i>	
<code>print('24', x.replace('world', 'Earth'))</code>	<i># Replace 'world' with 'Earth' in the</i>
<i>↳ string</i>	
<code>print('25', x.rfind('o'))</code>	<i># Find the last occurrence of 'o'</i>
<code>print('26', x.rindex('o'))</code>	<i># Get the index of the last occurrence</i>
<i>↳ of 'o'</i>	
<code>print('27', x.rjust(30, '-'))</code>	<i># Right-justify the string within a</i>
<i>↳ width of 30, padding with '-'</i>	
<code>print('28', x.rsplit('o'))</code>	<i># Split the string at 'o' from right</i>
<i>↳ to left</i>	
<code>print('29', x.rstrip('!'))</code>	<i># Remove trailing '!' characters from</i>
<i>↳ the string</i>	
<code>print('30', x.split('o'))</code>	<i># Split the string at 'o'</i>
<code>print('31', x_1.splitlines())</code>	<i># Split the string into a list of lines</i>
<code>print('32', x.startswith('H'))</code>	<i># Check if the string starts with 'H'</i>
<code>print('33', x.strip('!'))</code>	<i># Remove leading and trailing '!'</i>
<i>↳ characters from the string</i>	
<code>print('34', x.swapcase())</code>	<i># Swap the case of characters in the</i>
<i>↳ string</i>	
<code>print('35', x.title())</code>	<i># Convert the string to titlecase</i>
<code>print('36', x.upper())</code>	<i># Convert the string to uppercase</i>
<code>print('37', x.zfill(18))</code>	<i># Pad the string with zeros to a total</i>
<i>↳ width of 18</i>	

```

1 Hello world 123!
2 hello world 123!
3 --hello World 123!--
4 2
5 True
6 hello      World      123!
7 4

```

```

8 4
9 False
10 False
11 True
12 False
13 False
14 False
15 False
16 False
17 True
18 False
19 False
20 False
21 hello World 123!-----
22 hello world 123!
23 hello World 123!
24 hello World 123!
25 7
26 7
27 -----hello World 123!
28 ['hell', ' W', 'rld 123!']
29 hello World 123
30 ['hell', ' W', 'rld 123!']
31 ['hello', 'World', '123!']
32 False
33 hello World 123
34 HELLO wORLD 123!
35 Hello World 123!
36 HELLO WORLD 123!
37 00hello World 123!

```

10 Lists

```
[21]: x = [1, 2, 3, 4, 5]
```

```

#
y = x.copy()
y.append(6)
print(y)

y = x.copy()
y.append(6)
print(y)

```

```
[1, 2, 3, 4, 5, 6]
```

```
[11]: %%writefile my_module.py
def greet():
    print('Hello World!')
```

Writing my_module.py