Python String

A Python string is a sequence of characters. There is a built-in class 'str' for handling Python string.

Assign String to a Variable

Assigning a string to a variable is done with the variable name followed by an equal sign and the string:

Example

```
a = "Hello"
print(a)
```

Python String Methods

Python String capitalize()

In Python, the capitalize() method converts first character of a string to uppercase letter and lowercases all other characters, if any.

The syntax of capitalize() is:

```
string.capitalize()
```

Example 1: Capitalize a Sentence

```
string = "python is AWesome."

capitalized_string = string.capitalize()

print('Old String: ', string)
print('Capitalized String:', capitalized_string)
```

Python String center()

The center() method returns a string which is padded with the specified character.

The syntax of center() method is:

```
string.center(width[, fillchar])
```

center() Parameters

The center() method takes two arguments:

- **width -** length of the string with padded characters
- **fillchar** (optional) - padding character

The `fillchar` argument is optional. If it's not provided, space is taken as default argument.

Example 1: center() Method With Default fillchar

```
string = "Python is awesome"

new_string = string.center(24)

print("Centered String: ", new_string)
```

When you run the program, the output will be:

```
Centered String:     Python is awesome
```

Python String casefold()

The casefold() method is an aggressive lower() method which converts strings to case folded strings for caseless matching.

The `casefold()` method removes all case distinctions present in a string. It is used for caseless matching, i.e. ignores cases when comparing.

For example, the German lowercase letter `ß` is equivalent to `ss`. However, since `ß` is already lowercase, the `lower()` method does nothing to it.
But, `casefold()` converts it to `ss`.

The syntax of `casefold()` is:

```
string.casefold()
```

Parameters for casefold()

The `casefold()` method doesn't take any parameters.

Return value from casefold()

Example 1: Lowercase using casefold()

```
string = "PYTHON IS AWESOME"

# print lowercase string
print("Lowercase string:", string.casefold())
```

Python String count()

The string count() method returns the number of occurrences of a substring in the given string.

In simple words, count() method searches the substring in the given string and returns how many times the substring is present in it.
It also takes optional parameters `start` and `end` to specify the starting and ending positions in the string respectively.
The syntax of count() method is:

```
string.count(substring, start=..., end=...)
```

String count() Parameters

count() method only requires a single parameter for execution. However, it also has two optional parameters:

- **substring** - string whose count is to be found.
- **start (Optional)** - starting index within the string where search starts.
- **end (Optional)** - ending index within the string where search ends.

Note: Index in Python starts from 0, not 1.

Example 1: Count number of occurrences of a given substring

```
# define string
string = "Python is awesome, isn't it?"
substring = "is"

count = string.count(substring)

# print count
print("The count is:", count)
```

Python String endswith()

The endswith() method returns True if a string ends with the specified suffix. If not, it returns False.

The syntax of endswith() is:

```
str.endswith(suffix[, start[, end]])
```

endswith() Parameters

The endswith() takes three parameters:

- **suffix** - String or tuple of suffixes to be checked
- **start** (optional) - Beginning position where **suffix** is to be checked within the string.

- **end** (optional) - Ending position where **suffix** is to be checked within the string.

Example 1: endswith() Without start and end Parameters

```
text = "Python is easy to learn."

result = text.endswith('to learn')
# returns False
print(result)

result = text.endswith('to learn.')
# returns True
print(result)

result = text.endswith('Python is easy to learn.')
# returns True
print(result)
```

Python String expandtabs()

The expandtabs() method returns a copy of string with all tab characters '\t' replaced with whitespace characters until the next multiple of tabsize parameter.

The syntax of expandtabs() method is:

```
string.expandtabs(tabsize)
```

Example 1: expandtabs() With no Argument

```
str = 'xyz\t12345\tabc'

# no argument is passed
# default tabsize is 8
result = str.expandtabs()

print(result)
```

When you run the program, the output will be:

```
xyz     12345    abc
```

Python String find() method

The find() method returns the index of first occurrence of the substring (if found). If not found, it returns -1.

The syntax of the `find()` method is:

```
str.find(sub[, start[, end]] )
```

Example 1: find() With No start and end Argument

```
quote = 'Let it be, let it be, let it be'

# first occurance of 'let it'(case sensitive)
result = quote.find('let it')
print("Substring 'let it':", result)

# find returns -1 if substring not found
result = quote.find('small')
print("Substring 'small ':", result)

# How to use find()
if (quote.find('be,') != -1):
    print("Contains substring 'be,'")
else:
    print("Doesn't contain substring")
```

Output

```
Substring 'let it': 11
Substring 'small ': -1
Contains substring 'be,'
```

Python String format()

The string format() method formats the given string into a nicer output in Python.

The syntax of format() method is:

```
template.format(p0, p1, ..., k0=v0, k1=v1, ...)
```

Here, `p0, p1,...` are positional arguments and, `k0, k1,...` are keyword arguments with values `v0, v1,...` respectively.
And, `template` is a mixture of format codes with placeholders for the arguments.

String format() Parameters

format() method takes any number of parameters. But, is divided into two types of parameters:

- **Positional parameters** - list of parameters that can be accessed with index of parameter inside curly braces {index}
- **Keyword parameters** - list of parameters of type key=value, that can be accessed with key of parameter inside curly braces {key}

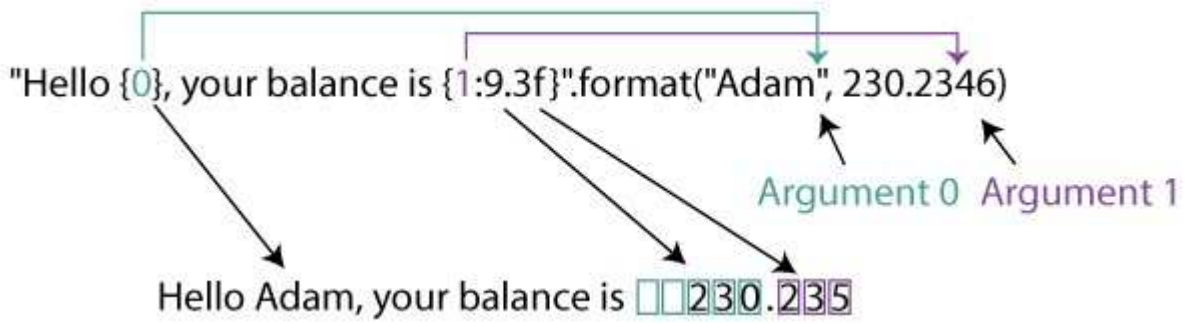
Return value from String format()

The format() method returns the formatted string.

How String format() works?

The format() reads the type of arguments passed to it and formats it according to the format codes defined in the string.

For positional arguments



Here, Argument 0 is a string "Adam" and Argument 1 is a floating number 230.2346.

Note: Argument list starts from 0 in Python.

The string `"Hello {0}, your balance is {1:9.3f}"` is the template string. This contains the format codes for formatting.

The curly braces are just placeholders for the arguments to be placed. In the above example, `{0}` is placeholder for `"Adam"` and `{1:9.3f}` is placeholder for `230.2346`.

Since the template string references format() arguments as {0} and {1}, the arguments are positional arguments. They both can also be referenced without the numbers as {} and Python internally converts them to numbers.

Internally,

- Since `"Adam"` is the 0th argument, it is placed in place of `{0}`. Since, `{0}` doesn't contain any other format codes, it doesn't perform any other operations.
- However, it is not the case for 1st argument `230.2346`.
  Here, `{1:9.3f}` places `230.2346` in its place and performs the operation 9.3f.
- f specifies the format is dealing with a float number. If not correctly specified, it will give out an error.

- The part before the "." (9) specifies the minimum width/padding the number (230.2346) can take. In this case, 230.2346 is allotted a minimum of 9 places including the ".".
  If no alignment option is specified, it is aligned to the right of the remaining spaces. (For strings, it is aligned to the left.)
- The part after the "." (3) truncates the decimal part (2346) upto the given number. In this case, 2346 is truncated after 3 places.
  Remaining numbers (46) is rounded off outputting 235.

Example 1: Basic formatting for default, positional and keyword arguments

```
# default arguments
print("Hello {}, your balance is {}.".format("Adam", 230.2346))

# positional arguments
print("Hello {0}, your balance is {1}.".format("Adam", 230.2346))

# keyword arguments
print("Hello {name}, your balance is {blc}.".format(name="Adam",
blc=230.2346))

# mixed arguments
print("Hello {0}, your balance is {blc}.".format("Adam", blc=230.2346))
```

When you run the program, the output will be same for all:

```
Hello Adam, your balance is 230.2346.
Hello Adam, your balance is 230.2346.
Hello Adam, your balance is 230.2346.
Hello Adam, your balance is 230.2346.
```

Python String index()

The index() method returns the index of a substring inside the string (if found). If the substring is not found, it raises an exception.

The syntax of the `index()` method for string is:

```
str.index(sub[, start[, end]] )
```

index() Parameters

The `index()` method takes three parameters:

- **sub** - substring to be searched in the string `str`.
- **start** and **end**(optional) - substring is searched within **str[start:end]**

Return Value from index()

- If substring exists inside the string, it returns the lowest index in the string where substring is found.
- If substring doesn't exist inside the string, it raises a **ValueError** exception.

The `index()` method is similar to find() method for strings.

The only difference is that `find()` method returns -1 if the substring is not found, whereas `index()` throws an exception.

Example 1: index() With Substring argument Only

```
sentence = 'Python programming is fun.'

result = sentence.index('is fun')
print("Substring 'is fun':", result)

result = sentence.index('Java')
print("Substring 'Java':", result)
```

Output

```
Substring 'is fun': 19

Traceback (most recent call last):
  File "<string>", line 6, in
    result = sentence.index('Java')
ValueError: substring not found
```