

# Object Methods

## Goals for this Section: "Object Methods"

Python 3

home (../handouts.html)

- **Use** object methods to *process* object values

```
var = 'Hello, World!'
var2 = var.replace('World', 'Mars') # replace substring, return a str
print(var2)                        # Hello, Mars!
```

- **Use** object methods to *inspect* object values

```
var = 'Hello, World!'
var2 = var.count('l')      # count the 'l' characters, return an int
print(var2)                # 3
```

### **Objects are capable of *behaviors*.**

Objects are much more than values. They come equipped with *behaviors*. We call these behaviors *methods*.

Some methods are used to manipulate or process the object's value. Others are used to *inspect* the object and tell us things about its value.

## Methods

*Methods are custom functions that are used only with a particular type.*

**upper()** is a **str** method. It is only used with strings.

```
var = 'hello!'          # str object assigned to var

var2 = var.upper()      # call the upper method on str object var

print(var2)             # HELLO!
```

Note the syntax: *object.method()*. We *call* the method in the same way we call functions: with parentheses.

## Methods vs. Functions

Compare *method* syntax to *function* syntax.

```
mystr = 'HELLO'

x = len(mystr)          # call len() and pass mystr, returning int 5

y = mystr.count('L')    # call count() on mystr, pass str L, returning int 2

print(y)                # 2
```

Methods and functions are both *called* (the parentheses after the name of the function or method).

Both also may take an *argument* and/or may return a *return value*.

## String Methods: **upper()** and **lower()**

These methods return a new string with a string's value uppercased or lowercased.

**upper()** string method

```
var = 'hello'
newvar = var.upper()

print(newvar)           # 'HELLO'
```

### **lower()** string method

```
var = 'Hello There'
newvar = var.lower()

print(newvar)           # 'hello there'
```

## String Methods: **replace()**

The **replace()** string method takes two arguments - a substring to be replaced, and the string with which to replace it. The string with replacements is returned as a new string object.

```
var = 'My name is Joe'

newvar = var.replace('Joe', 'Greta')    # pass 2 arguments to this method:
                                         # substring and replacement string

print(newvar)                           # My name is Greta
```

## String Methods: **format()**

The string **format()** method performs string substitution, placing *any* value (even numbers) within a new, completed string.

```
aa = 'Jose'
var = 34
bb = '{} is {} -- {} years old'.format(aa, var, var) # 3 arguments to rep
print(bb)                                           # Jose is 34 -- 34 years old.

cc = '{name} is {age} years old -- {age}'.format(name=aa, age=var) # 2 ar
```

**format()** is a string method, but it is often called on a *literal string* (that is, a string that is written out literally in your code).

The string must contain *tokens* (marked by curly braces) that will be replaced by values. The values are passed as arguments to **format()**.

**format()** is the preferred way for combining strings with other values. Concatenation or commas are usually too "busy" for such purposes:

```
print(aa + ' is ' + str(var) + ' years old')
```

## String Methods: **isdigit()** and **isalpha()**

These *inspector* methods return **True** if a string is all digits or all alphabetic characters.

Since they return **True** or **False**, they are used in an **if** or **while** expression.

**isdigit()**: return **True** if this string contains all digit characters

```
mystring = '12345'
if mystring.isdigit():
    print("that string is all numeric characters")
else:
    print("that string is not all numeric characters")
```

**isalpha()**: return **True** if this string is composed of all alphabetic characters

```
mystring = 'hello'
if mystring.isalpha():
    print("that string is all alphabetic characters")
```

## String Methods: **endswith()** and **startswith()**

These inspector methods return **True** if a string starts with or ends with a substring.

**endswith()**: return **True** if the string ends with a substring

```
bb = 'This is a sentence.'
if bb.endswith('.'):
    print("that line had a period at the end")
```

**startswith()**: return **True** if the string starts with a substring

```
cc = input('yes? ')
if cc.startswith('y') or cc.startswith('Y'):
    print('thanks!')
else:
    print("ok, I guess not.")
```

## String Methods: **count()** and **find()**

These inspector methods return integer values.

**count()**: return the number of times a substring appears in this string

```
aa = 'count the substring within this string'
bb = aa.count('in')
print(bb)                # 3 (the number of times 'in' appears in the string)
```

**find()**: return the *index position* (starting at 0) of a substring within this string

```
xx = 'find the name in this string'
yy = xx.find('name')
print(yy)                # 9 -- the 10th character in mystring
```

## Method and Function Return Values in an Expression; Combining Expressions

The return value of an expression can be used in another expression.

```
letters = "aabbcddefgafbdchabacc"

varb = letters.count("a")    # 5, number of times "a"
                             # appears in letters

vara = len(letters)          # assign integer object 20 to
                             # new label length

varc = vara / varb           # 20 / 5, or .25, the percentage of a's in the string

print(len(letters) / letters.count("a"))    # the above lines combined, print .25

print(float(letters.count("a")) / len(letters) * 100)    # 25.0

# the above line adds one more calculation:
# the whole-number percentage of a's in this string
```

[This is the last slide in this section.]