ICS3U: String Information Methods

A **method** is a function that is associated with an object. For our purposes, functions and methods *do* the same thing, but are *called* differently. While a function is called using only its name, as in `add_values(x, y)`, a method is always prefixed by the object, as in `my_object.add_values(x, y)`. Methods associated with strings must always be prefixed by a string literal, or by a variable that is a `str` data type.

Python has many string methods in the form `isSOMETHING`, which return either `True` or `False` if a string possesses a certain quality. For example, to determine whether a string, `s`, is made entirely of uppercase letters, we can use `s.isupper()`. The table below shows some of these methods (you can find more in the official Python documentation). For the examples below, use the strings `value="42"`, `name = "Jon"`, `address="123 Main St."`, `fruit="banana"`, and `blank="  \t\n"`.

Method	Description	Example / Result
`isdigit`	Check if a string is made entirely of the digits 0-9	`value.isdigit()` → `True`
`isalpha`	Check if a string is made entirely of letters a-z or A-Z	`name.isalpha()` → `True`
`isalnum`	Check if a string is made entirely of letters and digits	`address.isalnum()` → `False`
`isupper`	Check if all letters in a string are uppercase	`name.isupper()` → `False`
`islower`	Check if all letters in a string are lowercase	`fruit.islower()` → `True`
`isspace`	Check if a string is made entirely of whitespace chars	`blank.isspace()` → `True`

Below is a function that checks if a postal code is written in the form A9A 9A9, where A is any capital letter A-Z and 9 is any digit 0-9.

```
def validate_postal_code(p_code):
   if len(postal_code) == 7:
      if p_code[0].isalpha() and p_code[2].isalpha() and p_code[5].isalpha():
         if p_code[1].isdigit() and p_code[4].isdigit() and p_code[6].isdigit():
            if p_code[3] == " ":
               return True
   return False
```

You should test it by passing some strings to `validate_postal_code`. Note that we could *not* use `isspace` to check for a space in index 3, since a different whitespace character (such as a tab) would cause `isspace` to return `True`. There are other ways that we could have structured the function, as we will see shortly, but the code above is sufficient.

Consider another function that checks whether a password meets a given set of criteria. Many password systems require a combination of letters, numbers and special characters, as well as some minimum length. Our password will need to satisfy the following criteria:

- must have a minimum of 6 characters
- must not contain any whitespaces
- must contain at least one uppercase and one lowercase letter
- must contain at least one digit
- must contain at least one of the following “special” characters: ‘!’, ‘@’, ‘#’, ‘$’, or ‘%’

We can write a function that uses a simple loop to check each character in the string, and set a flag for each condition. Comments have been added for clarity.

```
def password_valid(password):

   # password too short
   if len(password) < 6:
      return False

   else:
      # set flags
      uppercase = False
      lowercase = False
      digit = False
      special = False

      # check all characters in password
      for char in password:
         if char.isupper():     # found an uppercase character
            uppercase = True
         elif char.islower():   # found a lowercase character
            lowercase = True
         elif char.isdigit():   # found a digit
            digit = True
         elif char in "!@#$%":  # found a special character
            special = True
         else:                  # an invalid character was found
            return False

   # if all criteria are met, password is valid, otherwise it is not
   if uppercase and lowercase and digit and special:
      return True
   else:
      return False
```

Note that we did not have to explicitly check for whitespace in our function, because it is handled by the `else` clause at the end. If the single character was a space (or other whitespace character, such as a tab), then `isupper`, `islower`, and `isdigit` would all return `False`. It should be pointed out that `isupper` and `islower` behave slightly differently than other methods like `isdigit` or `isalpha`. The latter will check if an entire string consists of a certain type of character, such as a number or a letter. On the other hand, `isupper` will check whether all letters in a given string are uppercase. The string may contain other non-alphabetic characters, which will be ignored by `isupper`. Similarly, `islower` will check only the letters in a given string to see if they are lowercase.

```
>>> "abc".isalpha()
True
>>> "a b c ABC".isalpha()
False
>>> "abc".islower()
True
>>> "a b c 123".islower()
True
```