Strings

C-START Python PD Workshop

Special Characters

Special characters can be inserted in a string using an **escape sequence**: a backslash (\) followed by another character. Here are some common escape sequences:

`\"`	Double Quote	`\\`	Backslash
`\n`	Newline	`\t`	Horizontal Tab

Special Characters

Special characters can be inserted in a string using an **escape sequence**: a backslash (\) followed by another character. Here are some common escape sequences:

`\"`	Double Quote	`\\`	Backslash
`\n`	Newline	`\t`	Horizontal Tab

Here is an example of using some escape sequences:

```
print("Favorite Color:\n\t\"Glow in the Dark\"")
```

```
Favorite Color:
        "Glow in the Dark"
```

Single or Double Quotes: Your Choice

Strings can be written using either single or double quotes, your choice.

```
primary = 'Python'
secondary = "English"
```

Single or Double Quotes: Your Choice

Strings can be written using either single or double quotes, your choice.

```
primary = 'Python'
secondary = "English"
```

Using single quotes means no need to escape double quotes:

```
print('So you must be "the one"?')
```

Single or Double Quotes: Your Choice

Strings can be written using either single or double quotes, your choice.

```
primary = 'Python'
secondary = "English"
```

Using single quotes means no need to escape double quotes:

```
print('So you must be "the one"?')
```

Using double quotes means no need to escape single quotes:

```
print("Margaret's house is blue.")
```

Strings Are Like Lists

Strings are like lists containing characters:

```
myname = "Jack"
print(myname[0])
```

```
J
```

Strings Are Like Lists

Strings are like lists containing characters:

```
myname = "Jack"
print(myname[0])
```

```
J
```

But unlike lists, strings cannot be modified:

```
myname = "Jack"
myname[0] = "T"    # bad
```

Strings are Iterables!

```
for c in 'hello world':
    print(c)
```

```
h
e
l
l
o

w
o
r
l
d
```

.split()ting Strings

To separate a string into a list based on white spaces, call `.split()` on it. Here is an example:

```
my_str = "  Python is  really cool"
wordlist = my_str.split()
# wordlist will be ["Python", "is", ... ]
for word in wordlist:
    print(word)
```

```
Python
is
really
cool
```

.split()ting Strings

To separate a string into a list based on white spaces, call `.split()` on it. Here is an example:

```
my_str = "  Python is  really cool"
wordlist = my_str.split()
# wordlist will be ["Python", "is", ... ]
for word in wordlist:
    print(word)
```

The . Operator

The . operator used above is actually the **accessor operator**, however, most programmers simply call it the **dot operator**. It allows us to use a function which is specific to a certain data type on the object.

Splitting the Input

Remember that the `input` function returns a string contaning the *line* that the user typed. If we want to accept multiple words per line, we must split the input.

```
line = input("What is your full name? ")
words = line.split()
firstname = words[0]
lastname = words[1]
```

Splitting the Input

Remember that the `input` function returns a string contaning the *line* that the user typed. If we want to accept multiple words per line, we must split the input.

```
line = input("What is your full name? ")
words = line.split()
firstname = words[0]
lastname = words[1]
```

Useful for Kattis

Some Kattis problems require that you recieve input on a single line separated by spaces. This is an effective method to receive the input.