Strings

C-START Python PD Workshop
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:

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

Favorite Color: "Glow in the Dark"
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:

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

Favorite Color:
"Glow in the Dark"
Strings can be written using either single or double quotes, your choice.

```python
primary = 'Python'
secondary = "English"
```
Strings can be written using either single or double quotes, your choice.

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

Using single quotes means no need to escape double quotes:

```python
print('So you must be "the one"?')
```
Strings can be written using either single or double quotes, your choice.

```python
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 containing characters:

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

J
Strings are like lists containing characters:

```python
def_length(myname):
    length = 0
    for letter in myname:
        length += 1
    return length

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

But unlike lists, strings **cannot be modified**:

```python
def_length(myname):
    length = 0
    for letter in myname:
        length += 1
    return length

myname = "Jack"
myname[0] = "T"  # bad
```
Strings are Iterables!

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

```
h
e
l
l
o
w
o
r
l
d
```
To separate a string into a list based on white spaces, call .split() on it. Here is an example:

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

```python
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 containing the line that the user typed. If we want to accept multiple words per line, we must split the input.

```python
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 containing the line that the user typed. If we want to accept multiple words per line, we must split the input.

```python
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 receive input on a single line separated by spaces. This is an effective method to receive the input.