Strings in Python

- The `string` data type is used to represent text data.
- We’ve already been using string literals, which consist of a sequence of characters surrounded by either double or single quotes.
  ```
  "Enter a number:"
  'The sum is'
  ```
- To see the data type of these values, we can use the built-in `type()` function:
  ```
  >>> type("Enter a number: ")
  <class 'str'>
  >>> myName = "Dave"
  >>> type(myName)
  <class 'str'>
  ```
Inputting String Values

- Recall: the `input` function treats any value that the user enters as a string.

- Thus, when we want the user to enter a string, we use `input` without using `eval`.

  ```python
  >>> name = input("Enter your name: ")
  Enter your name: Perry
  >>> print(name)
  Perry
  ```

- Using `eval` doesn't work with strings that can't be evaluated.

  ```python
  >>> name = eval(input("Enter your name: "))
  Enter your name: Perry
  Traceback (most recent call last):
    File "<pyshell#47>", line 1, in <module>
      name = eval(input("Enter your name: "))
    File "<string>", line 1, in <module>
    NameError: name 'Perry' is not defined
  ```

Numbering the Characters in a String

- A string is just a sequence of characters.

- The position of a character within a string is known as its `index`.

- There are two ways of numbering characters in Python:
  - from left to right, starting from 0
    ```python
    "Perry"
    ```
  - from right to left, starting from -1
    ```python
    "Perry"
    ```

- 'P' has an index of 0 or -5
- 'y' has an index of 4 or -1
Accessing a Character in a String

- To access an individual character within a string, we specify its index as follows:

  `<string>[ <index>]`

- Examples:
  ```
  >>> name = "Perry"
  >>> name[0]
  'P'
  >>> name[4]
  'y'
  >>> name[2 - 1]
  'e'
  >>> name[-1]
  'y'
  ```

- An index that is too large or too small produces an error:
  ```
  >>> name[5]
  ...IndexError: string index out of range
  ```

Slicing: Extracting a Substring

- To extract a substring, we specify two index values:

  `<string>[ <startIndex>: <endIndex>]`

- The resulting substring/slice:
  - begins at `<startIndex>`
  - ends at `<endIndex> - 1`
  - for a substring of length N, figure out `<startIndex>` and do:

    `<string>[<startIndex>:<startIndex> + N]`

- Examples:
  ```
  >>> name = "Perry"
  >>> name[0:2]
  'Pe'
  >>> name[4:5]
  'y'
  >>> name[1:4]
  'err'
  ```
Slicing: Extracting a Substring (cont.)

• If we omit the start/end index, the substring extends to the start/end of the string:
  
  examples:
  
  >>> name = "Perry"
  >>> name[1:]
  'erry'
  >>> name[:3]
  'Per'
  >>> name[:]
  'Perry'

Slicing: Extracting a Substring (cont.)

• Given the following assignment:
  
  >>> s = "computer"

• What is the value of each of the following?
  
  >>> s[1]

  >>> s[-1]

  >>> s[2:4]

  >>> s[:3]

  >>> s[:]

  >>> s[-4:-1]
Concatenation

- To concatenate two strings, we use the + operator.
  - examples:
    ```python
    >>> word = "database"
    >>> plural = word + "s"
    >>> plural
    'databases'
    
  - note: both of the values must be strings
    ```python
    >>> print("grade: "+90)
    ...TypeError: cannot concatenate 'str' and 'int' objects
    ```
  - to convert a number to a string, use the str() function:
    ```python
    >>> print("grade: "+str(90))
    grade: 90
    ```

Other String Operations

- The * operator can be used to repeat a string.
  ```python
  >>> print("--"*10)
  ----------
  >>> print("ho! "*3)
  ho! ho! ho!
  ```
- The built-in len() function can be used to get the length of a string.
  ```python
  >>> len("Perry")
  5
  >>> len("")
  0
  ```
- The in operator allows us to test for the presence of a substring.
  ```python
  >>> "base" in "database"
  True
  >>> "case" in "database"
  False
  ```
for Statements and Strings

• Recall the syntax of a for statement:
  
  ```python
  for <variable> in <sequence>:
      <body>
  ```

  where <sequence> is a sequence of values
  <body> is one or more statements

• Example: what will the following for loop print?
  ```python
  >>> for i in [2, 4, 6, 8]:
      print(i * 3, end=" ")
  ```

• Because a string is a sequence of characters, we can use a for loop to iterate over the characters in a string:
  ```python
  >>> for i in "hello":
      print(i, end=" ")
  h e l l o
  ```

Lists

• As we saw earlier, a list is another type of sequence.
  ```python
  [2, 4, 6, 8]
  ['CS', 'math', 'english', 'psych']
  ```

• unlike a string, a list can include values of different types:
  ```python
  ['The Godfather', 1972, 'R']
  ```

• All of the string operations are really operations that can be applied to any type of sequence, including lists.
  ```python
  >>> majors = ['CS', 'math', 'english', 'psych']
  >>> majors[2]
  'english'
  >>> majors[1:3]
  ['math', 'english']
  >>> len(majors)
  4
  >>> majors + ['physics']
  ['CS', 'math', 'english', 'psych', 'physics']
  ```
Mutable vs. Immutable

• A list is **mutable**, which means that it can be changed "in place":
  
  ```python
  >>> majors = ['CS', 'math', 'english', 'psych']
  >>> majors
  ['CS', 'math', 'english', 'psych']
  >>> majors[2] = 'literature'
  >>> majors
  ['CS', 'math', 'literature', 'psych']
  ```

• A string is **immutable**, which means it can't be changed "in place."

  ```python
  >>> sentence = "a string a immutable."
  >>> sentence[0] = "A"
  TypeError: 'str' object does not support item assignment
  ```

Strings Are Objects

• In programming languages, an **object** is a construct that allows us to group together:
  
  • one or more data values
  
  • operations that can be performed on those values (special functions known as the object's methods)

• In Python, a string is an object.
  
  • data values = the characters in the string
  
  • methods = functions that operate on the string

• To use a method that belongs to an object, we use **dot notation**.
  
  • examples:
    ```python
    >>> name = "Perry"
    >>> name.upper()
    PERRY
    ```
String Methods (partial list)

- `s.lower()`: return a copy of `s` with all lowercase characters
- `s.upper()`: return a copy of `s` with all uppercase characters
- `s.find(sub)`: return the index of the first occurrence of the substring `sub` in the string `s` (-1 if not found)
- `s.count(sub)`: return the number of occurrences of the substring `sub` in the string `s` (0 if not found)
- `s.replace(target, repl)`: replace all occurrences of the substring `target` in `s` with the substring `repl`

Splitting a String

- The `split()` method breaks a string into a list of substrings.
- By default, it uses spaces to determine where the splits should occur:
  ```
  >>> name = "Martin Luther King"
  >>> name.split()
  ['Martin', 'Luther', 'King']
  ```
- You can specify a different separator:
  ```
  >>> date = "03/05/2013"
  >>> date.split("/")
  ['03', '05', '2013']
  >>> record = '11234,Alan Turing,comp sci'
  >>> record.split(',')
  ['11234', 'Alan Turing', 'comp sci']
  >>> "03/05/2013".split("/")
  ['03', '05', '2013']
  ```
Splitting a String (cont.)

- You can use simultaneous assignment to assign the slices produced by `split()` to separate variables:

```python
>>> mon, day, year = "03/05/2013".split("/")
```

```python
>>> mon
'03'
```

```python
>>> day
'05'
```

```python
>>> year
'2013'
```

Joining Together a List of Strings

- The `join()` method takes a list of strings and joins them together.
- `join()` is a `string` method, not a list method.
  - we call it using the string that we want to use as a separator

**Examples:**

```python
>>> components = ["Martin", "Luther", "King"]
```

```python
>>> " ".join(components)
'Martin Luther King'
```

```python
>>> "/".join(['03', '05', '13'])
'03/05/13'
```
Practice: Analyzing a Name

- Write a program that analyzes a person's name.

- Here's a sample run of the program:

  Enter your full name: George Alexander Louis Wales
  Your name has 28 characters (including spaces).

  Your name has 4 components.
  - first name: George
  - last name: Wales
  - other names: Alexander Louis

  Enter a letter: r
  That letter occurs 2 times in your name.
  The first occurrence is at position 3 in the name.