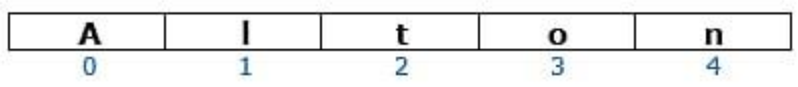
Python Programming 1

Unit 2 Module 1: Strings

Section 1 String Sequences

Accessing a Single String Character

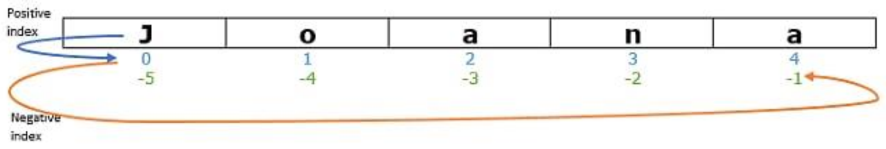
- Strings are **sequences of characters**. Another common sequence type used in this course is a **list**. Sequences index items counting from 0 for the first item.



Example:
```
# assign string to student_name
student_name = "Alton"
# first character is at index 0
student_name[0]
```

Access the end of a string using -1

- Strings assign an index number address to each string character
    - First character in a string is index 0
    - Last character in a string is index -1



- Example: To access the last character in a string:
```
student_name[-1]
```

Section 2 Index Slicing

Accessing Substrings

- String slicing returns a string section by addressing the start and stop indexes
- Syntax: Index Slicing [start:stop]
- Example:
```
  # assign string to student_name
  student_name = "Colette"
  # addressing the 3rd, 4th and 5th characters
  student_name[2:5]
```

- The slice starts at index 2 and ends at index 5 (but does not include index 5)

Accessing Substring Beginnings

- String slicing returns a string section from index 0 by addressing only the stop index
- Syntax: Index Slicing [:stop]
- Example:
```
student_name = "Colette"
# addressing the 1st, 2nd & 3rd characters
student_name[:3]
```
- default start for a slice is index 0

Accessing Substring Endings

- String slicing returns a string section including by addressing only the start index
- Syntax: Index Slicing [start:]
- Example:
```
student_name = "Colette"
# addressing the 4th, 5th and 6th characters
student_name[3:]
```
- default end index returns up to and including the last string character

Accessing Substrings by Step Size

- Syntax: Index Slicing [:], [::2]
- [:] returns the entire string
- [::2] returns the first char and then steps to every other char in the string
- [1::3] returns the second char and then steps to every third char in the string
- the number 2, in the print statement below, represents the step
- Example:
```
print(long_word[::2])
```

Stepping Backwards

- Use [::-1] to reverse a string.
- Example:
```
print(long_word[::-1])
```

Section 3 Iterating Strings

Iterate a String by Character

- Python provides powerful sequence iteration features. Below, for letter in word: loops through each letter in word.
- Syntax:
```
for letter in word:
```
- Example:
```
word = "cello"
for letter in word:
    print(letter)
```

- The variable letter is an arbitrary variable name. Any valid variable name can be used.

Iterate Substrings

- Combine string slicing and iteration:
- Example:

```
student_name = "Skye"

for letter in student_name[:3]:
    print(letter)
Iterate backwards using: student_name[::-1].
```

Section 4 String Methods

Methods for Returning String Information

- len()
    - Returns a string's length.
- .count()
    - Returns the number of times a character or substring occurs.
- .find()
    - Syntax: .find(string)
        - Returns index of first character or substring match. Returns -1 if no match found.
    - Syntax: .find(string, start index, end index)
        - Same as above .find() but searches from optional start and to optional end index.
- Example:

```
work_tip = "save your code"

# number of characters
len(work_tip)

# letter "e" occurrences
work_tip.count("e")

# find the index of the first space
work_tip.find(" ")

# find the index of "u" searching a slice work_tip[3:6]
work_tip.find("u",3,6)
```

- These methods return information that we can use to sort or manipulate strings.

Unit 2 Module 2: Lists

Section 1 List Sequences

- A simple lists contains comma separated objects enclosed in square bracket

```
    empty_list = [ ]
    sample_list = [1, 1, 2, 3, 3, 3, 3, 4, 5, 5, 5, 5, 5]
```

- List object types are not restricted so a mix of object types can be in single list

```
    mixed_list = [1, 1, "one", "two", 2.0, sample_list, "Hello"]
```

- To access a list we need to count like a computer, and that means starting with zero (0)
- Lists give an **index** number to each list item.
    - first element in a list is index 0
    - second element in a list is index 1
- To access the first item in a list use the list name, followed by square brackets containing the index number.

```
    age_survey[0]
```

Section 2 List Append

- `.append()` method adds an item to the end of a list

```
    party_list.append("Alton")
```

Section 3 List Insert

- Overwrite a specific index in a list

```
  party_list[2] = "Tobias"
```

  - **Overwrites** existing index
  - **Cannot** use to **Append** a new index to the list
- Use .insert() to define an index to insert an item¶

```
      party_list.insert(2,"Tobias")
```

  - Inserts, doesn't overwrite
  - Increases index by 1, at and above the insertion point
  - Can use to Append a new index to the end of the list

Section 3 List Delete

- Delete a specific list index
- del statement

```
     del party_list[2]
```

- .pop() method default is last item in a list

```
     last_item = party_list.pop()
     first_item = party_list.pop(0)
```

- in a conditional an empty list will evaluate False
  - This allows creating a while loop that runs until a list is empty

```
    while dog_types:
```

- .remove(object) removes the 1st item that matches

```
   dog_types.remove("Pug")
```

- ValueError occurs if the object is not available to be removed.

Unit 2 Module 3: List Iteration

Section 1 The Power of List Iteration

- Concept: Iterate through Lists using for in

```
cities = ["New York", "Shanghai", "Munich", "Tokyo", "Dubai",
"Mexico City", "São Paulo", "Hyderabad"]

for city in cities:
    print(city)
```

- Sort and Filter
    - use comparison operators while iterating lists
    - Example:

```
for bone_name in foot_bones:
   if len(bone_name) < 10:
      shorter_names += "\n" + bone_name
   else:
      longer_names += "\n" + bone_name
```

- Counting & Searching
    - use string methods while iterating lists
    - .count()

```
cities = ["New York", "Shanghai", "Munich", "Tokyo", "Dubai", "Mexico City", "São
Paulo", "Hyderabad"]
search_letter = "a"
total = 0

for city_name in cities:
   total += city_name.lower().count(search_letter)
```

- Creating a search function

```
def city_search(search_item, cities = ["New York", "Shanghai", "Munich", "Tokyo"] ):
   for city in cities:
      if city.lower() == search_item.lower():
         return True
      else:
         # go to the next item
         pass
   # no more items in list
   return False
```

Section 2 Range Iteration

- The range(stop) function creates a sequence

    - Using 1 argument with range(stop):
        - Default start: 0
        - stop: stopping integer, does not process stop number
    - Range runs from 0 through the integer before stop
    - Example

```
for count in range(10):
    print(count)
```

is the same as

```
for count in [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]:
    print(count)
```

- The range(start,stop) function creates a sequence
    - Using 2 arguments with range(start,stop):
        - start: starting integer value of a range loop
        - stop: stopping integer (second argument), does not process stop number
    - Example

```
for count in range(5,10):
    print(count)
```

- The range(start,stop,step) function creates a sequence
    - Using 3 arguments with range(start,stop,step):
        - start: starting integer value of a range loop
        - stop: stopping integer (second argument), does not process stop number
        - step: skip value for each loop
    - Example

```
for count in range(10,101,10):
   print(count)
```

Section 3 .extend(), .reverse() and .sort Methods

- Combining Lists
    - + list addition
    - .extend() list method
    - Example:

```
visited_cities = ["New York", "Shanghai", "Munich”]
wish_cities = ["Reykjavík", "Moscow", "Beijing", "Lamu"]
# combine in a new list
all_cities = visited_cities + wish_cities

# add a list to an existing list
visited_cities.extend(wish_cities)
```

- Reversing Lists
    - .reverse()
    - Example:

```
cities_1 = ["Dubai", "Mexico City", "São Paulo",
"Hyderabad"]
```

```
print("regular", cities_1)
cities_1.reverse()
print("reversed", cities_1)
```

- Sorting Lists
  - .sort() in place
    - .sort() - orders a list in place
    - Example

      ```
      quiz_scores = [20, 19, 20, 15, 20, 20, 20, 18, 18, 19]
      quiz_scores.sort()
      ```
  - sorted() copy
    - sorted() - creates an ordered list copy
    - Example

      ```
      game_points = [3, 14, 0, 8, 21, 1, 3, 8]
      sorted_points = sorted(game_points)
      ```

Section 4 Between Strings & Lists

- Converting a string to a list with .split()
  - .split() by default, splits a string at spaces (" ") to create a list
  - Example:

    ```
    tip = "Notebooks can be exported as .pdf"
    tip_words = tip.split()

    for word in tip_words:
        print(word)
    ```
- Concept: .split('-')
  - To split on characters other than " " (space), provide .split() a string argument to use as break points
  - Example:

    ```
    code_tip = "Python-uses-spaces-for-indentation"
    tip_words = code_tip.split('-')
    ```

- Build a string from a list
  - .join() is a method applied to a separator string and iterates through its argument¶
  - Example:

    ```
    tip_words = ['Notebooks', 'can', 'be', 'exported', 'as',
    '.pdf']
    " ".join(tip_words)
    ```
  - A space (" ") is the separator that gets injected between the objects in the argument (the list "tip_words").

- More Python String Tools
  - Cast a string to a list of characters
    - Example:

```
hello_letters = list("Hello")
```

- Print to the same line with multiple print statements (end=)
- Or insert any character as an end in print("String", end="+").

```
print('Hello', end = '')
print('world')
```

Unit 2 Module 4: Working with Files

Section 1 Files Import, Open and Read

- curl imports files to Jupyter session from a web address
- Below is code using curl to import poem1.txt, the code is in a command line interface syntax.

```
!curl https://raw.githubusercontent.com/MicrosoftLearning/intropython/master/poem1.txt -o poem1.txt
```

The table explains each element of the command above.

code	meaning
!	runs command interface supporting **curl**
`curl`	enables **curl** that can download files
`https://raw.githubusercontent.com/...`	is the address for data file to import
`-o`	tells **`curl`** write data to a file
`poem1.txt`	name **`curl`** will give the file

- open() creates an object that can be addressed in python code
    - "r" – read mode

MODE	Description
'r'	**read only mode**

- Others:

'w'	write - overwrites file with same name
'r+'	read and write mode
'a'	opens for appending to end of file

- read()
  - reading text
  - Example:
    poem_contents = poem_file.read()
  - .read() loads the content of the file into memory as a string, including formatting such as new line (\n)
  - .read() returns a string
    - These strings can be manipulated just like any other string
- read(n)
  - Reading a file with .read(n)
    - Where n = number of characters to read

Section 2 File .readlines() and .close() Methods

- File read as a list with .readlines()
  - Converts the lines of a file into a **list** of strings.
  - Example:

```
poem_lines = poem1.readlines()
```

- Remove newline characters from lists created using .readlines()
  - Example

```
for line in poem_lines:
    poem_lines[count] = line[:-1]
    count += 1
```

  - line[:-1] sets the end point at the last character of the string, the result is the '\n' (newline) character is omitted.

- File .close() method frees resources
  - The file.close() method removes the reference created by the file open() function.
  - Example

```
poem1.close()
```

Section 3 readline() and strip() Methods

- Use .readline() to read a line in a file as a **string**
- Each .readline() moves to the next available line in the file.
    - Example:
      ```
      poem1 = open('poem1.txt', 'r')
      poem_line1 = poem1.readline()
      poem_line2 = poem1.readline()
      poem_line3 = poem1.readline()
      ```
- while .readline()
    - while loop continues while the readline() value in poem_line returns text
    - A string value evaluates as True in the while loop
    - An empty string, '', evaluates not True in the while loop
    - When readline() reaches the end of the file, an empty string is returned
    - Example
      ```
      poem_line = poem1.readline()
      while poem_line:
          print(poem_line.capitalized())
          poem_line = poem1.readline()
      ```
- .strip() whitespace
    - .strip() removes leading and trailing whitespace, including the '\n' formatting character.
    - Example:
      ```
      poem_line = poem1.readline().strip()
      ```
- .strip() arguments
    - .strip('*\n') removes leading and training * and \n.
    - Example:
      ```
      color = rainbow_messy.readline().strip('*\n*')
      ```

Section 4 File .write() and .seek() Methods

- Writing to a file
    - write mode: 'w'
    - write mode plus read: 'w+'
    - 'w' and 'w+' modes will create a new file or overwrite existing files
    - All previous data will be lost
- Using .seek(0)
    - Using .seek() to set the read/write pointer to beginning of file
    - seek(0) sets the pointer to the beginning of the file, enabling read() to input the entire file contents
    - Setting the pointer to beginning of file
      ```
      new_file.seek(0)
      new_contents = new_file.read()
      print(new_contents)
      ```
    - new_file.seek(0) moves the file read\write pointer to the start of the file.
- Using .seek() offset and whence
    - Setting the pointer in a file with positive offset values and whence location
    - Example:
      ```
      new_file.seek(13)
      ```

```
new_contents = new_file.read()
print(new_contents)
new_file.seek(0,2)
```

- Using .seek() to set the read/write pointer in a file
    - offset values and whence arguments
    - .seek() can set the pointer to a desired index from the beginning of the file.
    - The example below moves to the character to offset 10 (the 11th character).
      `new_file.seek(10)`
    - Which is also written with an optional second argument, know as whence ("from where").
      `new_file.seek(10,0)`
    - Using 0 for whence starts the offset from the beginning of the file.
    - Note: the offset must be a positive integer in Python 3, so we cannot offset backwards from the end of the file
- file.seek(offset, whence)

- whence mode	description
0	points to the beginning of the file
1	points to the current location
2	points to the end of the file & offset is always 0

- Using whence the index can be offset from either the beginning, current location or to the end of the file (where there is no offset applied).
- Open a file in a writeable mode

MODE	Description
'r'	read only mode
'w'	write - **overwrites** file with same name
'w+'	write and read mode - **overwrites** file with same name
'r+'	read and write mode (**no** overwrite)
'a'	opens for appending to end of file (**no** overwrite)

'a+'	opens for appending to end of file (**no** overwrite) plus read

- Open a file in a writing mode, with: 'w', 'w+', 'r+', 'a', 'a+'
- Warning: 'w' and 'w+' modes will create a new file or overwrite existing files (deleting all file contents)

- Writing to a file opened in additional write modes 'r+', 'a', 'a+'
  - Writing is the same - pointers are different.
- read mode plus write 'r+' and append modes 'a', 'a+'
  - read plus mode 'r+' differs from write modes 'w', 'w+'
    - read plus doesn't blank out the file contents with an overwrite
  - append modes 'a', 'a+' differ from write modes 'w', 'w+'
    - append doesn't blank out the file contents with an overwrite
    - append pointer is set to the end of the file for every write
    - append plus (a+) is append mode, plus read mode