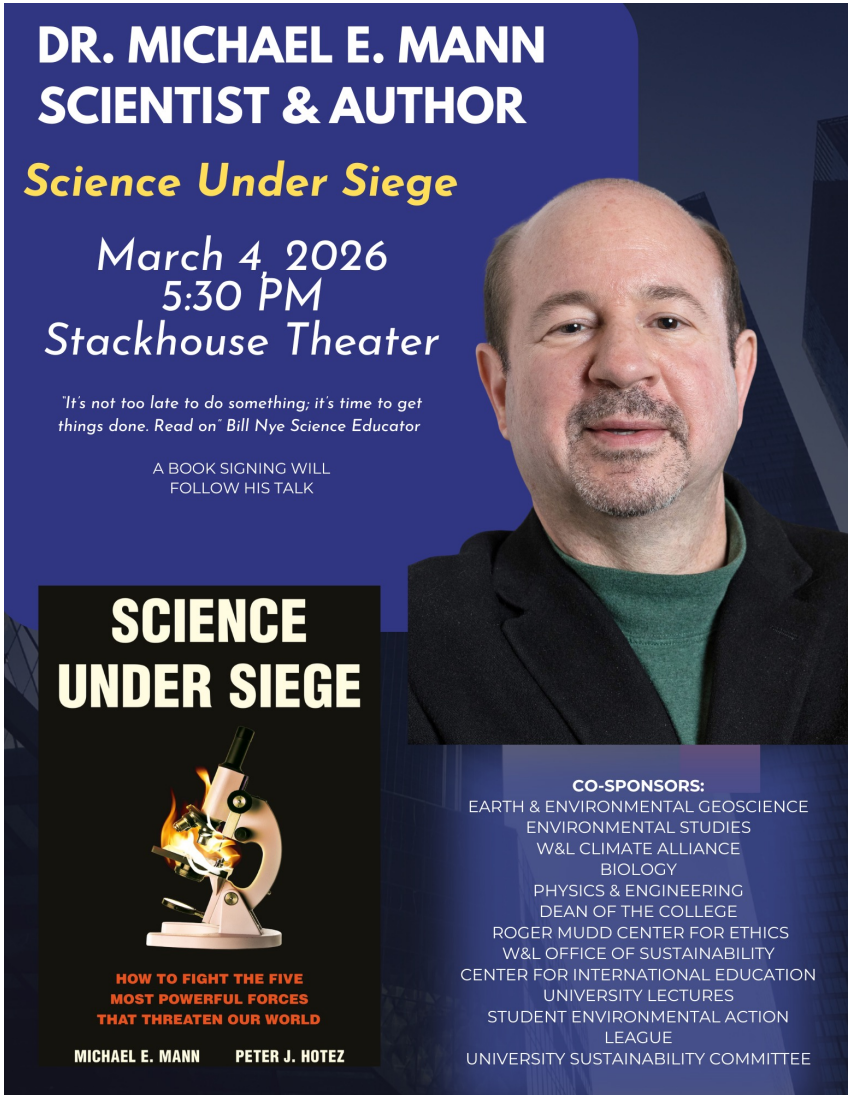
Objectives

- Continuing text processing, manipulation
  - String methods, operations, processing

Extra Credit Opportunity

- Post a response in Canvas's discussion forum "Extra Credit: Other Special Events"



Review

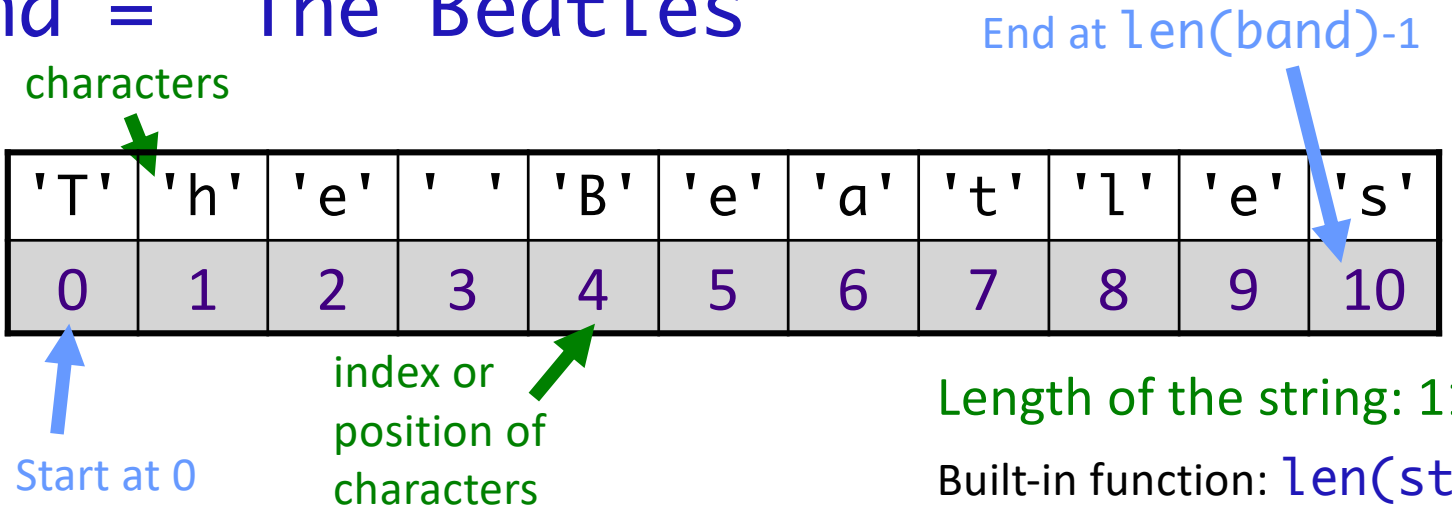
Get out your handouts from last time

- How do we represent text?
  - How can we represent reaaaaaaaally long text?
- How can we combine strings?
  - How can we combine strings multiple times?
  - What if we want to combine a string and an integer? What would we need to do?
- How can you find out how many characters are in a string?
- How do we find the character at a particular position of a string?
- Object-oriented programming:
  - What is an API?
  - What are methods?
  - How do we call methods on an object?

Review: Strings

- A *sequence* of one-character strings
  - Example:

```
band = "The Beatles"
```

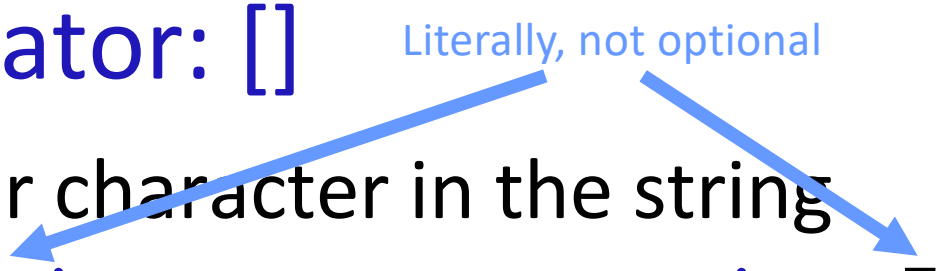


Length of the string: 11

Built-in function: `len(string)`

to find length of a string

Substrings Operator: []



- Look at a particular character in the string
  - Syntax: `string[<integer_expression>]`
  - [Positive value]: index of character
  - [Negative value]: count backwards from end
- Examples:
  - `<sequence>[0]` returns the first element/char
  - `<sequence>[-1]` returns the last element/char

We will deal with sequences beyond strings later.

USING THE STR API

`str` Methods

- `str` is a ***class*** or a ***type***
- **Methods**: available operations to perform on `str` objects
  - Provide common functionality
- To see all methods available for `str` class:
  - `help(str)`

`str` Methods

- Example method: **`find(substring)`**
  - Finds the first index where `substring` is in the string
  - Returns -1 if `substring` isn't found
- To call a method:
  - `<str_obj>.methodname([arguments])`
  - Example: `filename.find(".py")`

`find` method executed on this string

Common `str` Methods

Method	Operation
`center(width)`	Returns a copy of string centered within the given number of columns
`count(sub[, start [, end]])`	Returns # of non-overlapping occurrences of substring `sub` in the string.
`endswith(sub)` `startswith(sub)`	Returns `True` iff string ends with/starts with `sub`
`find(sub[, start [, end]])`	Returns first index where substring `sub` is found
`isalpha(), isdigit(), isspace()`	Returns `True` iff string contains letters/digits/whitespace *only*
`lower(), upper()`	Returns a copy of string converted to lowercase/uppercase

Common `str` Methods

Review: What do the square brackets in APIs mean?

Method	Operation
`center(width)`	Returns a copy of string centered within the given number of columns
`count(sub[, start [, end]])`	Returns # of non-overlapping occurrences of substring `sub` in the string.
`endswith(sub)` `startswith(sub)`	Returns `True` iff string ends with/starts with `sub`
`find(sub[, start [, end]])`	Returns first index where substring `sub` is found
`isalpha(), isdigit(), isspace()`	Returns `True` iff string contains letters/digits/whitespace *only*
`lower(), upper()`	Returns a copy of string converted to lowercase/uppercase

Common `str` Methods

Method	Operation
`replace(old, new[, count])`	Returns a copy of string with all occurrences of substring `old` replaced by substring `new`. If `count` given, only replaces first `count` instances.
`split([sep])`	Returns a list of the words in the string, using `sep` as the delimiter string. If `sep` is not specified or is None, any whitespace string is a separator.
`strip()`	Returns a copy of the string with the leading and trailing whitespace removed
`join(<sequence>)`	Returns a string which is the concatenation of the strings in the sequence with the string this is called on as the separator
`swapcase()`	Returns a copy of the string with uppercase characters converted to lowercase and vice versa.

Understanding the API: What Does This Code Do?

```
sentence = input("Enter a sentence to mangle: ")

length = len(sentence)

print("*", sentence.center(int(length*1.5)), "*")

upperSentence = sentence.upper()
print(upperSentence)
print(sentence)

print("Uppercase: ", sentence.upper())
print()
print("Lowercase: ", sentence.lower())
print()

print("Did sentence change?: ", sentence)
```

Functions vs Methods (with Strings)

Functions

- Associated with a file or module
- All input comes from arguments/parameters
- Example: `len` is a built-in function
  - Called as `len(strobj)`

Methods

- Associated with a *class* or *type*
- Input comes from arguments ***and*** the string the method was called on
- Example:
  - `strobj.upper()`

How to Use APIs

- Given a problem, break down the problem
  - Can any of the parts of the problem be solved using a method in the API?

Wheel of Fortune

- Determine how many of a certain letter are in a given phrase
- How would we solve this, regardless of case?

```
def get_num_letters( phrase, letter ):
```

Example Test Cases:

```
test.testEqual( get_num_letters("abracadabra", "a"), 5)
test.testEqual( get_num_letters("Abracadabra", "a"), 5)
test.testEqual( get_num_letters("abracadabra", "A"), 5)
```

Substrings Operator: [:]

- Select a substring (zero or more characters) using the **[ ]** and **:**
  - The values/positions must be integers
- `<sequence>[<start>:<end>]`
  - returns the subsequence from the position `start` up to and **not** including `end`
- `<sequence>[<start>:]`
  - returns the subsequence from the position `start` to the end of the sequence
- `<sequence>[:<end>]`
  - returns the subsequence from the first element up to and **not** including `end`
- `<sequence>[:]`
  - returns a copy of the entire sequence

Substrings Operator: [:]

- Select a substring (one or more characters)
- Examples: `filename = "program.py"`

p	r	o	g	r	a	m	.	p	y
0	1	2	3	4	5	6	7	8	9

Expression	Result
`filename[0:2]`	
`filename[0:]`	
`filename[:3]`	
`filename[8:]`	
`filename[-2:]`	

Substrings Operator: [:]

- Select a substring (one or more characters)
- Examples: `filename = "program.py"`

p	r	o	g	r	a	m	.	p	y
0	1	2	3	4	5	6	7	8	9

Expression	Result
`filename[0:2]`	`"pr"`
`filename[0:]`	`"program.py"`
`filename[:3]`	`"pro"`
`filename[8:]`	`"py"`
`filename[-2:]`	`"py"`

String Comparisons

- Same operations as with numbers:
  - ==, !=
  - <, <=
  - >, >=

  } Lexicographical comparison

- Use in conditions, e.g., in `if` statements

```
if course_choice == "CSCI111":
    print("Good choice!")
else:
    print("Maybe next semester")
```

Iterating Through a String

- Use a `for` loop to iterate through *characters* in a string

Loop variable represents a string of length 1

```
for char in string:
    print(char)
```

➢ Read as "for each character in the string"

Iterating Through a String

- Use a `for` loop to iterate through *characters* in a string

Name of loop variable doesn't matter
(with respect to the code's functionality)

```
for doggie in string:
    print(doggie)
```

- Read as "for each character (named by loop variable `doggie`) in the string"

Iterating Through a String

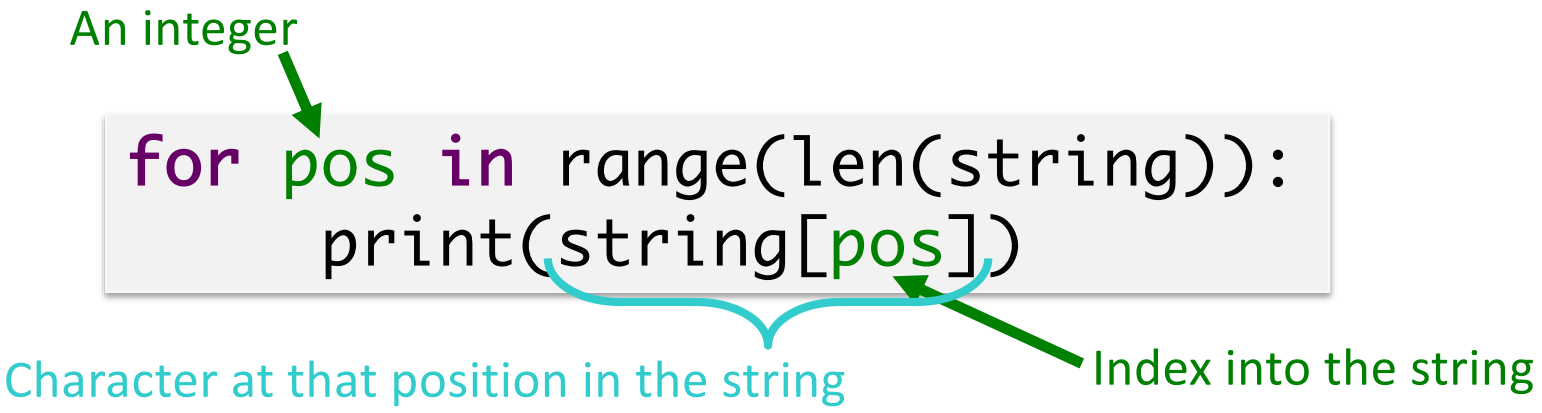
- Alternatively, can iterate through the *positions* in a string

```
for pos in range(len(string)):
    print(string[pos])
```

What values are assigned to **pos**?
What is displayed when `string = "kitty"`?

Iterating Through a String

- Alternatively, can iterate through the *positions* in a string
  - Could write as a **while** loop as well (How?)



Summary: Iterating Through a String

- For each *character* in the string

string of length 1

```
for char in mystring:
    print(char)
```

Produce the same output!

What comes after `in` determines loop's behavior

- For each *position* in the string

An integer

```
for pos in range(len(mystring)):
    print(mystring[pos])
```

Index into the string

Testing for Substrings

- Using the `in` operator
  - Used `in` before in `for` loops
- Syntax: `substring in string`
  - Evaluates to `True` or `False`
- Example: simple Web search

```
if searchTerm in documentText:
        print(document, "contains", searchTerm)
```

String Search Comparison

- What do the two `if` statements test for? (What would be descriptive, appropriate output?)

```
PYTHON_EXT = ".py"

filename = input("Enter a filename: ")

if filename[-(len(PYTHON_EXT)):] == PYTHON_EXT:
        # Appropriate output 1
if PYTHON_EXT in filename:
        # Appropriate output 2
```

Provide some examples for `filename` and state how code would execute

String Search Comparison

- What do the two `if` statements test for?

```
PYTHON_EXT = ".py"

filename = input("Enter a filename: ")

if filename[-(len(PYTHON_EXT)):] == PYTHON_EXT:
        # Appropriate output 1
if PYTHON_EXT in filename:
        # Appropriate output 2
```

How would the program execution change if the structure used an if-elif?

Looking Ahead

- Lab 6 Pre-lab due tomorrow
- Lab 6 tomorrow!
- Broader Issue: Social Media Responsibility