

# Strings

# Finding vowels

```
def num_vowels(s):  
    '''Return the number of vowels in str s  
    Do not treat 'y' as a vowel.'''  
  
    count = 0  
    for char in s:  
        if char in "aAeEiIoOuU":  
            count = count + 1  
    return count
```

# Extra Lab Hours

- TA in the lab 1:30 - 3:30 today for assignment questions
- Last chance for in-person help
- DB on the weekend

# Why the fan?



# Practice writing code

- `reverse.py`
- `remove_spaces.py`

# String comparisons

- You can use the comparison operators on strings.
- The comparison will be made alphabetically.
- An ordering is defined even for non-alphabetic characters. For example:
  - “!” > “,”
- But you only need to remember that:
  - “a” < “b” < ... < “z”
  - “A” < “B” < ... < “Z”
  - “0” < “1” < ... < “9”

# Indexing strings

`s[i]`

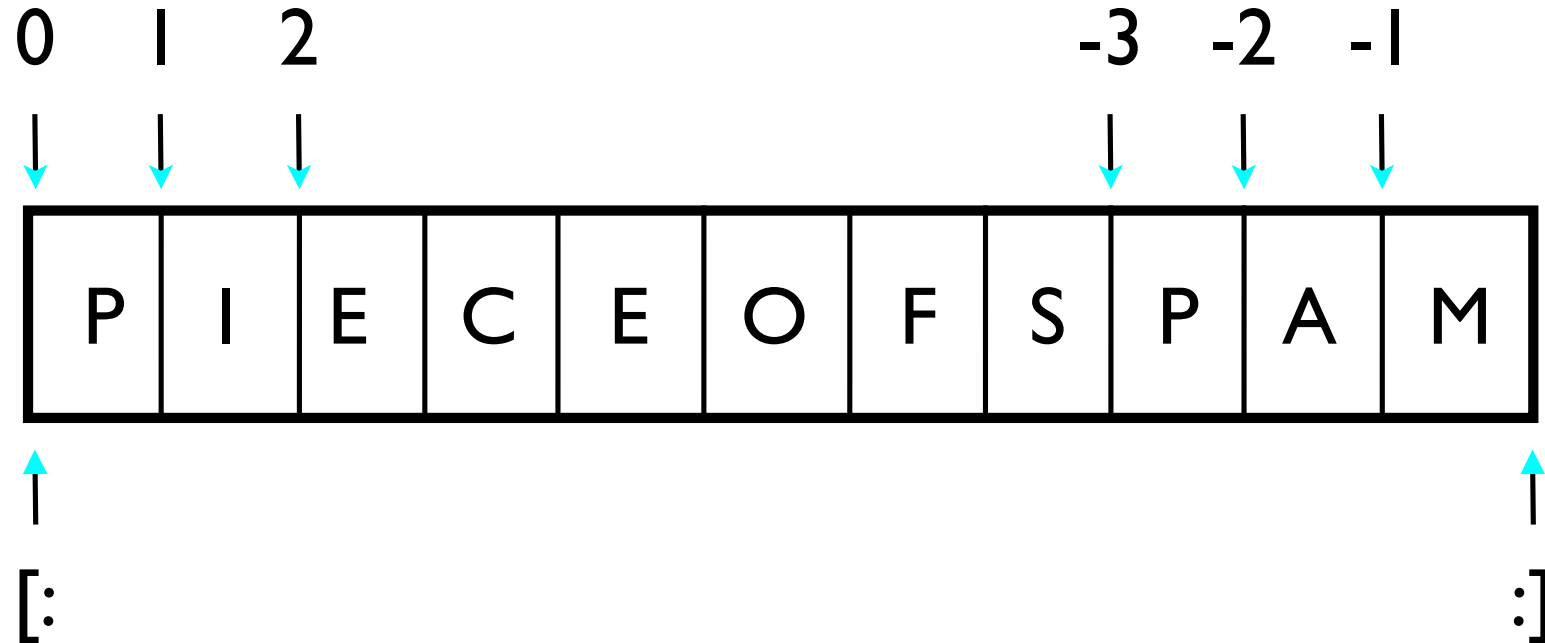
- Allows you to extract a single character.
- The first character is at index 0.
- A negative index means to count backwards from the end.
- Always returns a new object.

# Slicing strings

`s[left:right]`

- Allows you to extract a substring in one step.
- Always returns a new object.
- The lower bound is inclusive, but the upper bound is not.
- If omitted, left defaults to 0, right to `len(s)`.

Think of left and right as cutting the string.



Source: Lutz, page 134

# Methods

- We have seen several operators for strings.
- There are not enough operators for all the things one might want to do with strings.
- Instead, Python defines these using a special kind of function: a *method*.
- Strings “own” these methods, like a module owns its functions. So you call methods using the same notation as calling a function in a module:  
    “hogwarts”.capitalize()  
    villain = “malfoy”  
    villain.capitalize()

# Calling methods

- Because you provide the string using dot notation, you don't need to pass it as an argument:

`villain.capitalize(villain)` # Redundant!

In fact, this would cause an error.

- A method may have parameters if it needs *additional* information:  
`villain.startswith("mal")`

# Calling methods vs calling functions

- You need to know whether you are calling a method or a function.

- Example: `len` is a function, so:

|                      |                      |
|----------------------|----------------------|
| <code>len(s)</code>  | <code># Fine</code>  |
| <code>s.len()</code> | <code># Error</code> |

- Example: `lower` is a method, so:

|                        |                      |
|------------------------|----------------------|
| <code>lower(s)</code>  | <code># Error</code> |
| <code>s.lower()</code> | <code># Fine</code>  |

# Some string methods

- `S.replace(old, new)`: return a string, same as `S` but with all occurrences of `old` replaced by `new`. Does not change `S`.
- `S.count(substring)`: return the number of times `substring` occurs in `S`.
- `S.find(substring)`: return the index of the first occurrence of `substring` in `S`, starting from the left.
- `S.startswith(substring)`: return `True` iff `S` begins with the `substring`.
- And a very useful *function*: `len(string)`

# Why are some things methods and other things functions?

- Python could have defined len (and find etc.) as a method or a function.
- Programmers can define their own new kinds of objects too. (We'll learn how later.)
- The same decision has to be made for every operation you want to define for your new type of object object: method or function? There is always a choice.

# So how does one decide?

- One guideline: if the operation is only relevant to one type of object, make it a method defined for that type of object.
  - Eg: converting to lowercase.
- But if the operation is relevant to other types of object, make it a function, so it can be called with all of those types of object.
  - Eg: finding the length of something.
- You won't be asked to make these decisions in csc108. But the issue may have been bugging you.