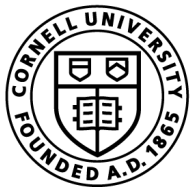
Lecture 4: Strings (Sections 8.1, 8.2, 8.4, 8.5, 1st paragraph of 8.9)

CS 1110

Introduction to Computing Using Python



Cornell Bowers CIS
Computer Science

[E. Andersen, A. Bracy, D. Gries, L. Lee, S. Marschner, C. Van Loan, W. White]

Clicker Question

Module Text File

```
# my_module.py


"""This is a simple module.
It shows how modules work"""


x = 1+2
x = 3*x
x
```

Command Line

```
C:\> python my_module.py
```

After you hit "Return" here what will be printed next?

(A) `C:\>`

(B) `9`
`C:\>`

(C) an error message

(D) The text of my_module.py

(E) Sorry, no clue.

Today

- More about the `str` type
  - This is where Ptyhon ***SHINES***
  - New ways to use strings
- More examples of functions
  - Functions with strings!

Strings

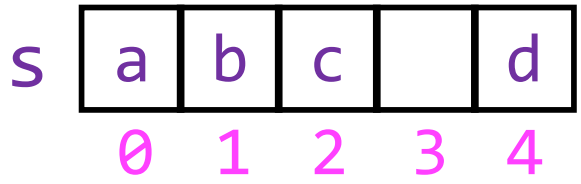
- Strings are indexed
- Access characters with [] — called "string slicing"

```
>>> s = "abc d"
>>> s[0]
'a'
>>> s[4]
'd'
>>> s[0:2]
'ab'
>>> s[2:]
'c d'
>>> s[5]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: string index out of range
```

excludes c

Two ways of drawing:

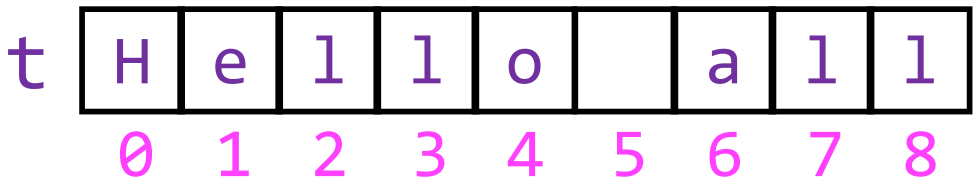
or



Question 1

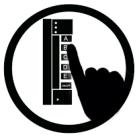
```
>>> t = 'Hello all'
>>> t[3:6]
...
```

Global Space



A: `'lo a'`
B: `'lo'`
C: `'lo '`
D: `'o '`
E: I do not know

What does this expression evaluate to?



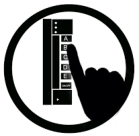
Question 2

```
>>> t = 'Hello all'
>>> t[:3]
...
```

Global Space

A: `'all'`
B: `'l'`
C: `'Hel'`
D: Error!
E: I do not know

What does this expression evaluate to?



Other Things We Can Do With Strings

Operator s_1 **in** s_2

- Tests if s_1 "a part of" s_2

  (or a *substring* of)

- Evaluates to a `bool`

Examples:

```
>>> s = 'abracadabra'
>>> 'a' in s
True
>>>  'cad' in s
True
>>>  'foo' in s
False
```

Built-in Function `len(s)`

- Value is # of chars in `s`
- Evaluates to an `int`

Examples:

```
>>> s = 'abracadabra'
>>> len(s)
11
>>> len(s[1:5])
4
>>> s[1:len(s)-1]
'bracadabr'
>>>
```

Defining a String Function

Want to write function `middle`, which returns the middle 3rd of a string (length divisible by 3).

How we want it to behave:

```
>>> middle('abc')
'b'
>>> middle('aabbcc')
'bb'
>>> middle('aaabbbccc')
'bbb'
```

Important Questions:

1. What are the parameters?
2. What is the return value?
3. What goes in the body?

```
def middle(text):

    ???

    return middle_third
```

Steps to writing a program

1. Work an instance yourself
2. Write down exactly what you just did
3. Generalize your steps from 2
4. Test your steps
5. Translate to Code
6. Test program
7. Debug (if necessary)

Steps to writing a program

1. Work an instance yourself
2. Write down exactly what you just did
3. Generalize your steps from 2
4. Test your steps
5. Translate to Code

```
>>> middle('abc')        middle_third = text[1]
>>> middle('aabbcc')     middle_third = text[2:4]
```

Too easy!!

Still too easy!!

```
>>> middle('It was the best of times, it was the worst of times, it was the age of wisdom, it was the age of foolishness, it was the epoch of belief, it was the epoch of incredulity, it was the season of Light, it was the season of Darkness, it was the spring of hope, it was the winter of despair, we had everything before us, we had nothing before us, we were all going direct to Heaven, we were all going direct the other way…')
```

Definition of **middle**

```
def middle(text):
    """Returns: middle 3rd of text
    Param text: a string with length divisible by 3"""
```

IMPORTANT:
Precondition requires that arguments to **`middle`** have length divisible by 3.

If not? Bad things could happen, and we blame the user (not the author) of the function.

Advanced String Features: Method Calls

Format: <*string name*>.<*method name*>(*x*,*y*,…)

`s1.index(s2)`

- returns position of the first instance of `s2` in `s1`
- `error if s2` is not in `s1`

See Python Docs for more

`s1.count(s2)`

- returns number of times `s2` appears inside of `s1`

`s.strip()`

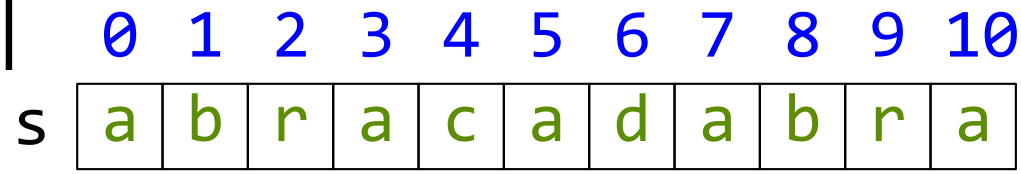
- returns a copy of `s` with white-space removed at ends

`s1.upper()`

- returns an upper case version

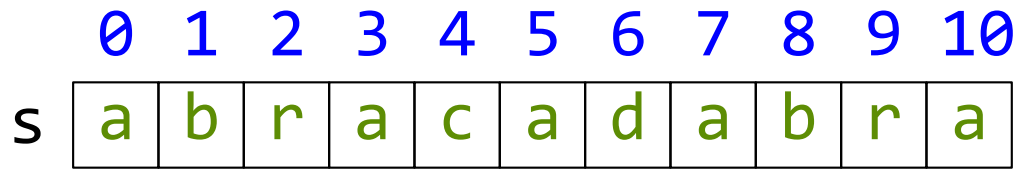
String Methods index, count, strip

```
>>> s = 'abracadabra'
>>> s.index('a')
0
>>> s.index('rac')
2
>>> s.count('a')
5
>>> s.count('b')
2
>>> s.count('x')
0
>>> '  a b '.strip()
'a b'
```



Why not just *<method name>*() ?

```
>>> s = 'abracadabra’
>>> index(s,5)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'index' is not defined
```



`index` *is not directly known to Python.*

This is a **string** method. Need to access it via a string.

(More details on this when we discuss classes.)

String Extraction Example

```
def firstparens(text):
    """Returns: substring in ()
    Uses the first set of parens
    Param text: a string with ()"""
```

```
>>> s = 'One (Two) Three'
>>> firstparens(s)
'Two'
>>> t = '(A) B (C) D'
>>> firstparens(t)
'A'
```

Steps to writing a program

1. Work an instance yourself
2. Write down exactly what you just did
3. Generalize your steps from 2
4. Test your steps
5. Translate to Code
6. **Test program**
7. Debug (if necessary)

Think of all the corner cases

What could possibly go wrong?

String Extraction, Testing reveals a problem

```
def firstparens(text):
    """Returns: substring in ()
    Uses the first set of parens
    Param text: a string with ()"""

    # Find the open parenthesis
    start = text.index('(')

    # Find the close parenthesis
    end = text.index(')')

    inside = text[start+1:end]

    return inside
```

```
>>> s = 'One (Two) Three'
>>> firstparens(s)
'Two'
>>> t = '(A) B (C) D'
>>> firstparens(t)
'A'
```

Works! Are we done?

```
>>> v = 'A) B (C) D'
>>> firstparens(v)
```

Uh oh….

We assumed the first close paren would come *after* the first open paren, but technically it doesn't have to….

String Extraction, a better version

```
def firstparens(text):
    """Returns: substring in ()
    Uses the first set of parens
    Param text: a string with ()"""

    # Find the open parenthesis
    start = text.index('(')


    # Store part AFTER paren
    substr = text[start+1:]


    # Find the close parenthesis
    end = substr.index(')')


    inside = substr[:end]
    return inside
```

```
>>> s = 'One (Two) Three'
>>> firstparens(s)
'Two'
>>> t = '(A) B (C) D'
>>> firstparens(t)
'A'



>>> v = 'A) B (C) D'
>>> firstparens(v)
```

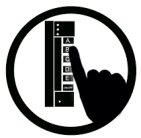
Extraction Puzzle

```
def second(thelist):
    """Returns: second word in a list
    of words separated by commas, with
    any leading or trailing spaces
    from the second word removed
    Ex: second('A, B, C') => 'B'
    Param thelist: a list of words
    with at least two commas """

1   start = thelist.index(',')
2   tail = thelist[start+1:]
3   end = tail.index(',')
4   result = tail[:end]
5   return result
```

Is there an error?

A: Yes, Line 1
B: Yes, Line 2
C: Yes, Line 3
D: Yes, Line 4
E: There is no error



Extraction Puzzle

```
def second(thelist):
    """Returns: second word in a list
    of words separated by commas, with
    any leading or trailing spaces
    from the second word removed
    Ex: second('A, B, C') => 'B'
    Param thelist: a list of words
    with at least two commas """

1   start = thelist.index(',')
2   tail = thelist[start+1:]
3   end = tail.index(',')
4   result = tail[:end]
5   return result
```

```
>>> second('cat, dog, pig, lion
expecting: 'dog'
get: ' dog'

>>> second('apple, pear, banana
expecting: 'pear'
get: ' pear'
```

Is there an error?

A: Yes, Line 1
B: Yes, Line 2
C: Yes, Line 3
D: Yes, Line 4
E: There is no error

Extraction Fix #1

```
def second(thelist):
    """Returns: second word in a list
    of words separated by commas, with
    any leading or trailing spaces
    from the second word removed
    Ex: second('A, B, C') => 'B'
    Param thelist: a list of words
    with at least two commas """
```

```
>>> second('cat, dog, pig, lion
expecting: 'dog'
get: ' dog'

>>> second('apple, pear, banana
expecting: 'pear'
get: ' pear'
```

```
1   start = thelist.index(',')
2   tail = thelist[start+1:]
3   end = tail.index(',')
4   result = tail[:end]
5   return result
```

→ `tail = thelist[start+2:]`

What if there are *multiple* (*or no!*) spaces?

Extraction Fix #2 (the better fix)

```
def second(thelist):
    """Returns: second word in a list
    of words separated by commas, with
    any leading or trailing spaces
    from the second word removed
    Ex: second('A, B, C') => 'B'
    Param thelist: a list of words
    with at least two commas """

1   start = thelist.index(',')
2   tail = thelist[start+1:]
3   end = tail.index(',')
4   result = tail[:end]
5   return result
```

```
>>> second('cat, dog, pig, lion
expecting: 'dog'
get: ' dog'

>>> second('apple, pear, banana
expecting: 'pear'
get: ' pear'
```

→ `result = tail[:end].strip()`