

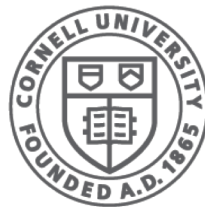
<http://www.cs.cornell.edu/courses/cs1110/2019sp>

Lecture 5: Strings

(Sections 8.1, 8.2, 8.4, 8.5,
1st paragraph of 8.9)

CS 1110

Introduction to Computing Using Python



Cornell CIS
COMPUTING AND INFORMATION SCIENCE

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

Today

- More about the `str` type
 - New ways to use strings
- More examples of functions
 - Functions with strings!
- Learn the difference between `print` and `return`

Strings are Indexed (Question 1)

- `s = 'abc d'`

0	1	2	3	4
a	b	c		d

- `t = 'Hello all'`

0	1	2	3	4	5	6	7	8
H	e	l	l	o		a	l	l

- Access characters with []

- `s[0]` is 'a'
- `s[4]` is 'd'
- `s[5]` causes an error
- `s[0:2]` is 'ab' (excludes c)
- `s[2:]` is 'c d'

- What is `t[3:6]`?

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

- Called “string slicing”

Strings are Indexed (Solution 1)

- `s = 'abc d'`

0	1	2	3	4
a	b	c		d

- `t = 'Hello all'`

0	1	2	3	4	5	6	7	8
H	e	l	l	o		a	l	l

- Access characters with `[]`

- `s[0]` is 'a'
- `s[4]` is 'd'
- `s[5]` causes an error
- `s[0:2]` is 'ab' (excludes c)
- `s[2:]` is 'c d'

- What is `t[3:6]`?

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

- Called “string slicing”

Strings are Indexed (Question 2)

- `s = 'abc d'`

0	1	2	3	4
a	b	c		d

- `t = 'Hello all'`

0	1	2	3	4	5	6	7	8
H	e	l	l	o		a	l	l

- Access characters with []

- `s[0]` is 'a'
- `s[4]` is 'd'
- `s[5]` causes an error
- `s[0:2]` is 'ab' (excludes c)
- `s[2:]` is 'c d'

- What is `t[:3]`?

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

- Called “string slicing”

Strings are Indexed (Solution 2)

- `s = 'abc d'`

0	1	2	3	4
a	b	c		d

- `t = 'Hello all'`

0	1	2	3	4	5	6	7	8
H	e	l	l	o		a	l	l

- Access characters with []

- `s[0]` is 'a'
- `s[4]` is 'd'
- `s[5]` causes an error
- `s[0:2]` is 'ab' (excludes c)
- `s[2:]` is 'c d'

- What is `t[:3]`?

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

- Called “string slicing”

Other Things We Can Do With Strings

Operator `in`: `s1 in s2`

- Tests if `s1` “a part of” (or a *substring* of) `s2`
- Evaluates to a bool

Examples:

```
>>> s = 'abracadabra'
```

```
>>> 'a' in s
```

```
True
```

```
>>> 'cad' in s
```

```
True
```

```
>>> 'foo' in s
```

```
False
```

Built-in Function `len`: `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] *Too easy!!*

>>> ~~middle('aabbcc')~~ middle_third = text[2:4] *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"""  
  
    # Get length of text  
    size = len(text)  
  
    # Start of middle third  
    start2 = size//3  
  
    # End of middle third  
    start3 = (2*size)//3  
  
    # Get the substring  
    middle_third = text[start2:start3]  
    return middle_third
```

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

- Strings have some useful *methods*
 - Like functions, but “with a string in front”
- **Format:** *<string name>.<method name>(x,y,...)*
- **Example:** `upper()` returns an upper case version

```
>>> s = 'Hello World'
```

```
>>> s.upper()
```

```
'HELLO WORLD'
```

```
>>> s
```

```
'Hello World'
```

```
>>> s[1:5].upper()
```

```
'ELLO'
```

```
>>> 'scream'.upper()
```

```
'SCREAM'
```

```
>>> 'cs1110'.upper()
```

```
'CS1110'
```

Examples of String Methods

- `s1.index(s2)`
 - Returns position of the first instance of `s2` in `s1`
 - **error** if `s2` is not in `s1`
- `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

• `s = 'abracadabra'`

0	1	2	3	4	5	6	7	8	9	10
a	b	r	a	c	a	d	a	b	r	a

- `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'

See Python Docs for more

String Extraction, Round 1

```
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'
```

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** *Think of all the corner cases*
7. Debug (if necessary) *What could possibly go wrong?*

String Extraction, Round 2

```
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'
```


String 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, mouse, lion')  
expecting: 'dog'           get: ' dog'  
  
>>> second('apple,pear , banana')  
expecting: 'pear'          get: 'pear '
```

Where is the error?

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

String Extraction 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:] → tail = thelist[start+2:]  #possible fix ??
```

```
3  end = tail.index(',')      What if there are multiple (or no!) spaces?
```

```
4  result = tail[:end] → result = tail[:end].strip()    #better fix!
```

```
5  return result
```

```
>>> second('cat, dog, mouse, lion')
```

```
expecting: 'dog'           get: ' dog'
```

```
>>> second('apple,pear , banana')
```

```
expecting: 'pear'          get: 'pear '
```

String: Text as a Value

- String are quoted characters
 - `'abc d'` (Python prefers)
 - `"abc d"` (most languages)
- How to write quotes in quotes?
 - Delineate with “other quote”
 - **Example:** `"' '"` or `'" "'`
 - What if need both `"` and `'`?
- **Solution:** escape characters
 - Format: `\` + letter
 - Special or invisible chars

Type: str

Char	Meaning
<code>\'</code>	single quote
<code>\"</code>	double quote
<code>\n</code>	new line
<code>\t</code>	tab
<code>\\</code>	backslash

Not All Functions Need a Return

```
def greet(n):
```

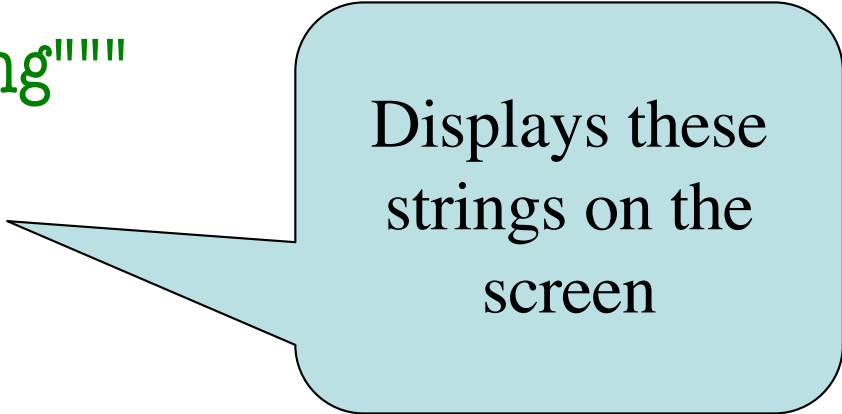
```
    """Prints a greeting to the name n
```

```
    Parameter n: name to greet
```

```
    Precondition: n is a string"""
```

```
    print('Hello '+n+'!')
```

```
    print('How are you?')
```



Displays these
strings on the
screen



No assignments or return
(returns None)

print

vs.

return

- Displays a value on screen
- Used primarily for **testing**
- Not useful for calculations

```
def print_plus(n):  
    print(n+1)
```

```
>>> print_plus(2)  
3  
>>>
```

- Sends a value from a function call frame back to the caller
- Important for **calculations**
- Does not display anything

```
def return_plus(n):  
    return n+1
```

```
>>> return_plus(2)  
3  
>>>
```

?

unexpected printing courtesy of:

Python Interactive Mode

- executes both *statements* and *expressions*
- if *expression*:
 1. *evaluates*
 2. *prints value (if one exists)*

>>> 2+2 ← *evaluates (performs addition)*

4 ← *prints value (4)*

>>> return_plus(2) ← *evaluates (makes function call, gets return value)*

3 ← *prints value (3)*

>>>

return_plus in action

```
def return_plus(n):  
    return n+1
```

call frame

return_plus	
n	2
RETURN	3

Python Interactive Mode

```
>>> return_plus(2)  
3  
>>>
```

*1. Evaluates : makes
function call, evaluates to
return value*

2. prints value

print_plus in action

```
def print_plus(n):  
    print(n+1)
```

Python Interactive Mode

```
>>> print_plus(2)
```

```
3
```

```
>>>
```

call frame

print_plus	
n	2
RETURN	NONE

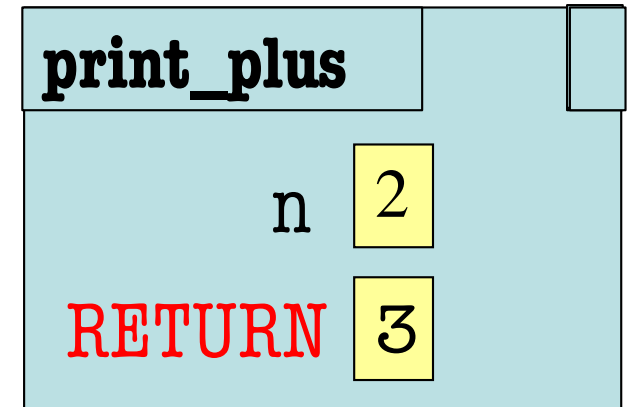
1. Evaluates : makes function call, evaluates to return value

2. does not print value b/c it's NONE

hybrid_plus in action

```
def hybrid_plus(n):  
    print(n)  
    return n+1
```

call frame



Python Interactive Mode

```
>>> print_plus(2)
```

```
2
```

```
3
```

```
>>>
```

*1. Evaluates : makes
function call, evaluates to
return value*

2. print value

See the difference in the Python Tutor

```
def print_plus(n):
```

```
    print(n+1)
```

```
def return_plus(n):
```

```
    return n+1
```

```
x1 = print_plus(2)
```

```
x2 = return_plus(2)
```

```
print(x1)
```

```
print(x2)
```

Program output:

3

None

3

Exercise 1

Module Text

```
# module.py
```

```
def foo(x):
```

```
    x = 1+2
```

```
    x = 3*x
```

Python Interactive Mode

```
>>> import module
```

```
>>> print(module.x)
```

```
...
```

What does Python
give me?

A: 9

B: 10

C: 1

D: None

E: Error

Exercise 1, Solution

Module Text

```
# module.py
```

```
def foo(x):
```

```
    x = 1+2
```

```
    x = 3*x
```

Python Interactive Mode

```
>>> import module
```

```
>>> print(module.x)
```

```
...
```

What does Python
give me?

A: 9

B: 10

C: 1

D: None

E: Error **CORRECT**

Exercise 2

Module Text

```
# module.py
```

```
def foo(x):  
    x = 1+2  
    x = 3*x
```

```
y = foo(0)
```

Python Interactive Mode

```
>>> import module
```

```
>>> print(module.y)
```

```
...
```

What does Python
give me?

A: 9

B: 10

C: 1

D: None

E: Error

Exercise 2, Solution

Module Text

```
# module.py
```

```
def foo(x):  
    x = 1+2  
    x = 3*x
```

```
x = foo(0)
```

Python Interactive Mode

```
>>> import module
```

```
>>> print(module.x)
```

```
...
```

What does Python
give me?

A: 9

B: 10

C: 1

D: None **CORRECT**

E: Error

Exercise 3

Module Text

```
# module.py
```

```
def foo(x):  
    x = 1+2  
    x = 3*x  
    return x+1
```

```
y = foo(0)
```

Python Interactive Mode

```
>>> import module
```

```
>>> module.y
```

```
...
```

What does Python
give me?

A: 9

B: 10

C: 1

D: None

E: Error

Exercise 3, Solution

Module Text

```
# module.py
```

```
def foo(x):  
    x = 1+2  
    x = 3*x  
    return x+1
```

```
y = foo(0)
```

Python Interactive Mode

```
>>> import module
```

```
>>> module.y
```

```
...
```

What does Python
give me?

A: 9

B: 10 **CORRECT**

C: 1

D: None

E: Error

Exercise 4

Function Definition

```
def foo(a,b):  
1  x = a  
2  y = b  
3  return x*y+y
```

Function Call

```
>>> x = 2  
>>> foo(3,4)  
>>> x  
...
```

What does Python
give me?

- A: 2
- B: 3
- C: 16
- D: None
- E: I do not know

Exercise 4, Solution

Function Definition

```
def foo(a,b):  
1  x = a  
2  y = b  
3  return x*y+y
```

Function Call

```
>>> x = 2  
>>> foo(3,4)  
>>> x  
...
```

What does Python
give me?

A: 2 **CORRECT**
B: 3
C: 16
D: None
E: I do not know