Operations on Strings, Lists
Introduction to Functions

CS 8: Introduction to Computer Science, Spring 2019
Lecture #4

Ziad Matni, Ph.D.
Dept. of Computer Science, UCSB

Administrative

- Hw02 – due Tuesday in class
- Lab01 – due on Sunday by midnight (11:59 pm) on **Gradescope**!

Lecture Outline

- Operations on Strings
- Intro to Lists & Tuple

Yellow Band = Class Demonstration! ☺

Strings

- These are all ok to use:

```
S = 'hello!'
S = "hello!"
S = "I said \"hello\"!"
S = 'I said "hello"!'
S = "I said 'hello'!"
```

Note the alternate use of “ and ‘

Adding a Newline Character

- If you want to print a string with a “newline” character in it…
  - i.e. equivalent to hitting the “Return” key

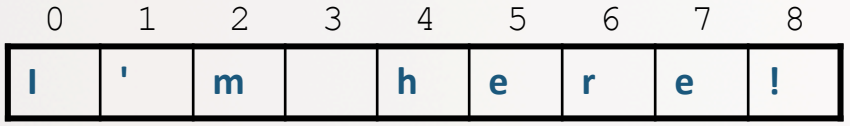
```
print("Hello!\nMy name is Zed")
```

<u>This will print out:</u>

```
Hello!
My name is Zed
```

Recall: Indexing

- Every character in a string has an index associated with it



- In Python, indexing always starts at **0**.
  - So the 1st character in the string is character #0
  - Indexing is called out with square brackets [*n*]

Indices and Slices

- To slice a string into a smaller string, use **[*i* : *j*]**
  - Where *i* = starting index, *j* = ending index (NOT included)
  - Example: `"Gaucho"[2:4]` is `"uc"`

- Combinations are possible!
  - Example, what does this spell out?

```
( ("o" + "Gaucho"[2:5] + "  " ) * 3 ) + "!"
```

Negative Indices in Strings

- If `s = "hello"`
- Then `s[-1] = "o"`
  `s[-2] = "l"` , etc…

- In the example above, `s[-2:] = "lo"`
  etc…

Exercise 1

- What is the value of s after the following code runs?

```
s = 'abc'
s = 'd' * 3 + s
s = s + e* 2
```

A. ‘abcd3e2’
B. ‘abcdddabc’
C. ‘dddabcee’
D. ‘abcdddabce2’
E. Error

Exercise 2

- What is the value of s after the following code runs?

```
s = 'abc'
s = 'd' * 3 + s
s = s + 'e'* 2
```

A. ‘abcd3e2’
B. ‘abcdddabc’
C. ‘dddabcee’
D. ‘abcdddabce2’
E. Error

Some Operations on Strings

- Given a string S, for example, **`"Tunneling"`**:

	`len(S)`	Length of string	e.g. 9
These are called ***methods***	S.upper()	Make string all upper-case	e.g. “TUNNELING”
	S.lower()	Make string all lower-case	e.g. “tunneling”
	S.find('n')	Find the 1st occurrence of	e.g. 2
	S.find('z')	*if not found…*	e.g. -1

More String Methods

Assume: `name = 'Bubba'`

- `name.count('b')` is `2` ← counts how many times 'b' occurs
- `name.count('ubb')` is `1` ← counts how many times 'ubb' occurs

- `name.center(9)` is `'  Bubba  '` ← centers w/ spaces on each side
- `name.ljust(9)` is `'Bubba    '` ← left justifies name in 9 spaces
- `name.rjust(9)` is `'    Bubba'` ← right justifies name in 9 spaces

- `name.replace('bb','dd')` is `'Budda'` ← Replaces one sub-string for another

More (Fun)ctions we can use with Strings!

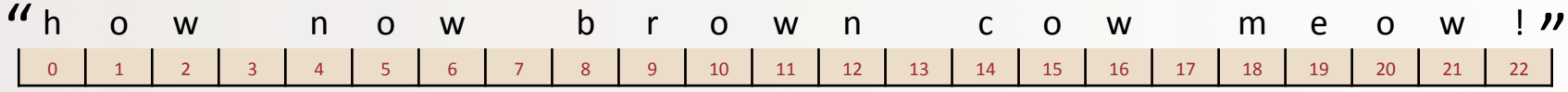
- Boolean operators `in` and `not in` are great ways to check if a sub-string is found inside a longer string

Examples:

- `"fun" in "functions"` = `True`
- `"fun" in "Functions"` = `False`
- `"Fan" not in "Functions"` = `True`

Example

Assume string **s** = “how now brown cow meow!”



What is:

- `s.find('m')` = 18
- `s.find('r')` = 9
- `s.find('ow')` = 1
- `s.find('s')` = -1
- `s.replace(' meow', 'moo?')` = “how now brown cowmoo?!”
- `'n c' in s` = True

*note: one space before **meow***
note: space between n and c

Lists

- A **list** is a collection of multiple values
  - Similar to how a str is a collection of characters

- Note: In Python, lists can be of ***heterogenous***
  - Of ***different types*** (i.e. ints or strings or etc…)

- Lists can also have duplicate values
- Lists are ***mutable*** : elements of a list can be **modified**

Example of Lists

```
NameList = [“Abby”, “Bruce”, “Chris”]
Student = [“Jill Jillson”, 19, 3.7, “F”]
```

`NameList` and **`Student`** are variables of type **list**

- You can call up list elements by indexing the list

Example: `NameList[0] = “Abby”`

Some Operations on Lists

- Given a list L, for example, **`[1, 2, -5, 9, 0, 1]`**:

`len(L)`	Length of list	e.g. 6
`max(L)`	Max value in a list	e.g. 9
`min(L)`	Min value in a list	e.g. -5
`sum(L)`	Sum of all values in a list	e.g. 8

Mostly used on lists of ***numbers***

Tuples

- Tuples are a variable type that's very similar to lists
  - Except they are *immutable*!
  - That is, once they're set, they cannot change

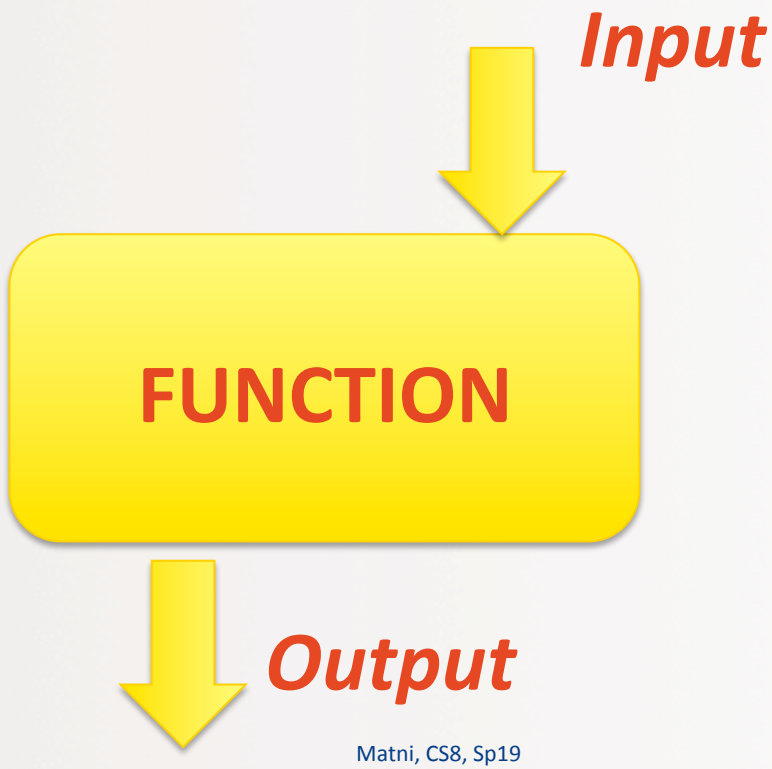
- Example of a tuple:

```
collection =  (1, 2, "buckle my shoe", 3, 4)
```

- You can call up list elements by indexing the list

Example: `collection[1] = 2`

Functions



Procedural Abstraction: The Function

- A “black box” – a piece of code that can take inputs and gives me some expected output

- A **function**, for example, is a kind of procedural abstraction

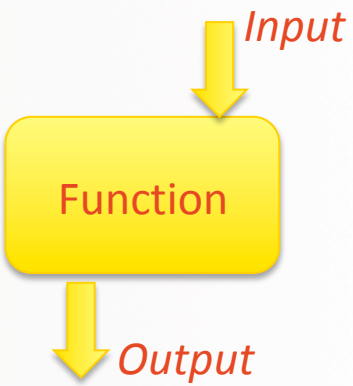
  25 → Square Root Function → 5

  - What’s happening inside the function?
  - Doesn’t always matter!... As long as it works!!

Programmed Function

- Does "something" to **input(s)** and sends back **output(s)**
  - Always has ***parentheses*** to "carry" the inputs
  - These inputs are called the ***function arguments***

- Example: the **sqrt()** function (find the square root)
  - With an input of 25, I expect an output of 5
  - That is, **sqrt(25)** will give me (**RETURNS** to me) **5**

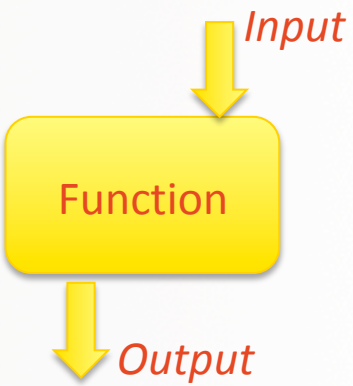


More About Functions

- Definition:
  "Self contained" modules of code that accomplish a specific task.

- The function often (although not always) **"returns" an output** (result)
- The "returned" output is linked to the function name *(examples coming…)*
- Sometimes the function does not return anything…

- Functions can be "***called from***" the main block of a program
  - Or from inside other functions!



More About Functions

- A function can be used over and over again.

- Example:
  Consider a function called "***distance***" that returns the value of the distance between a point w/ coordinates (a, b) and the Cartesian origin (0, 0)

distance (a, b) = square root of (a^2 + b^2)

We can "reuse" this function with different values for a and b!

```
distance(2, 4)        distance(92, -41)        distance(distance(1,1), 4)
```

Defining Your Own Function

- To define a function in Python, the **necessary** syntax is:

```
def functionName (parameters):
    # a block of statements appear here
    # all of them must be indented (with tabs)
```

  - **`def`** – a mandatory keyword that **defines a function**
  - **`functionName`** – any legal Python identifier (e.g. myLittleFunction)
  - **`( ):`** – mandatory set of parentheses **and** colon
  - ***`parameters`*** – object names (can be none, 1 param., or multiple params.)

Example Definition

```
# My first function! Yay!
def dbl(x):
    """This function returns double its input x"""
    print("I'm doubling the number to:", 2*x)
    return 2*x    # I need to "return" the result
```

Let's try it out!

FUNCTION RULES!

```
# My first function! Yay!
def dbl(x):
    """This function returns double its input x"""
    print("I'm doubling the number to:", 2*x)
    return 2*x    # I need to "return" the result
```

Function header
*x is the input **parameter** (also called **argument**)*

Function body

***Indentation**: VERY IMPORTANT*
Achieved with a tab character or just spaces
*All the lines in the function body are indented from the function header, and all to the **same** degree*

***docstring**: a comment that becomes part of Python's built-in help system!*
With each function be sure to include one that:
a) describes overall what the function does, and
b) explains what the inputs mean/are

More Example Definitions

```
# This function calculates the distance between (a,b) and (0,0)
def distance(a, b):
      x = a**2          # Note the tab indent!!!
      y = b**2          # Recall ** means "to the power of"
      z = (x + y) ** 0.5
      return z          # I need to "return" the result
```

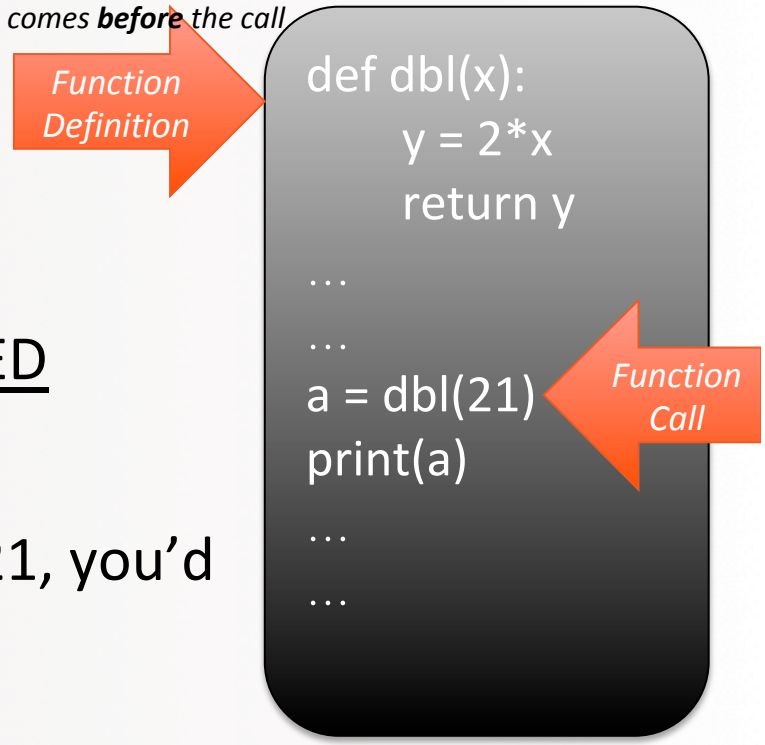
!!! Alternatively, I can also do this !!!

```
def distance(a, b):
      return ( (a**2) + (b**2) ) ** 0.5
```

Let's try it out!

Flow of Execution of a Function

- **DEFINING vs. CALLING** a function
- Calling is how you get to "run" it from another place in the code
- Use its name and arguments AS DEFINED
- Example:
  to call the **dbl** function for an input of 21, you'd have to call it like this: **dbl(21)**



What if There are Multiple Parameters??

- **When you call a function, the values you put in parenthesis have to be in the order in which they are listed in the definition!**

- Example:

```
def subtract(m, n):
    return m - n
```

When you call this function to do a subtraction of 5 – 99, then:
m has to be 5 and **n has to be 99**
So, it's called as:
`subtract(5, 99)`
i.e. **not** `subtract(99, 5)`

What About… NO Parameters?!

- Sure, you can do that!

- Example:

```
def fortyTwo():
    return 42
```

All this function does is return the number 42 to whoever called it!

Which way should we call it?

fortyTwo

fortyTwo()

Wow. Functions are Cool.
Can They CALL EACH OTHER????

Yes!!!!!!!!!!!!!!! Be careful that you get the order correct…!

```
def halve( x ):
""" returns half its input, x """
    return div(x, 2)

def div( y, x ):
""" returns y / x """
    return y / x
```

What happens when I say:

`>>> halve( 85 )`

A. I get 42
B. I get 42.5
C. 0
D. 0.02352 (i.e., 2 divided by 85)

Let's try it out!

YOUR TO-DOs

- ❑ Finish reading **Chapter 2**
- ❑ Start reading **Chapter 3**
- ❑ Finish up **HW2** (due **Tuesday**)
- ❑ Finish up **Lab1** (due **Sunday**)

- ❑ Remember office hours/open labs! ☺

- ❑ Eat your greens…

</LECTURE>