CS 1110:
Introduction to Computing Using Python

Lecture 5

Strings

[Andersen, Gries, Lee, Marschner, Van Loan, White]

Today

- Return to the string (str) type
  - Learn several new ways to use strings
- See more examples of functions
  - Particularly functions with strings
- Learn the difference between print and return

Strings are Indexed

- 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'
- Called “string slicing”

- What is t[3:6]?

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

Strings are Indexed

- s = 'abc d'

0	1	2	3	4
a	b	c		d

- 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'
- Called “string slicing”

- t = 'Hello all'

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

- What is t[:3]?

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

Other Things We Can Do With Strings

- **Operation** in: s_1 in s_2
  - Tests if s_1 "a part of" s_2
  - Say s_1 a *substring* of s_2
  - Evaluates to a bool
- **Examples**:
  - s = 'abracadabra'
  - 'a' in s → True
  - 'cad' in s → True
  - 'foo' in s → False

- **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

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

Defining a String Function

1. Add string parameter
2. Add return at end
   - Set to be “result” for now
3. Work in reverse
   - Set subgoals
   - Identify needed operations
   - Store results in variables
   - Assign on previous lines

```
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
    start = size/3
    # End of middle third
    end = 2*size/3
    # Get the text
    result = text[start:end]
    # Return the result
    return result
```

Defining a String Function

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

```
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
    start = size/3
    # End of middle third
    end = 2*size/3
    # Get the text
    result = text[start:end]
    # Return the result
    return result
```

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() - converts to upper case
  - s = 'Hello World'
  - s.upper() → 'HELLO WORLD'
  - s[1:5].upper() → 'ELLO'
  - 'methods'.upper() → 'METHODS'
  - 'cs1110'.upper() → 'CS1110'

Examples of String Methods

- s_1.index(s_2)
  - Position of the first instance of s_2 in s_1
- s_1.count(s_2)
  - Number of times s_2 appears inside of s_1
- s.strip()
  - A copy of s with white-space removed at ends

- 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'

See Python Docs for more

String Extraction Example

```
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(')')
    # Return the result
    return substr[:end]
```

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

HANDOUT IS WRONG!

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

    # Find the open parenthesis
    start = s.index('(')
    # Store part AFTER paren
    tail = s[start+1:]
    # Find the close parenthesis
    end = tail.index(')')
    # Return the result
    return tail[:end]
```

```
>>> 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
    and spaces.
    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')
'dog'
>>> second('apple, pear, banana')
'pear'
```

Where is the error?

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

String Extraction Puzzle

```
def second(thelist):
    """Returns: second word in a list
    of words separated by commas
    and spaces.
    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')
'dog'
>>> second('apple, pear, banana')
'pear'
```

```
tail = thelist[start+2:]
```

but what if there are *multiple* spaces?

```
result = tail[:end].strip()
```

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
\'	single quote
\"	double quote
\n	new line
\t	tab
\\	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
The call frame is **EMPTY**

Procedures vs. Fruitful Functions

Procedures

- Functions that **do** something
- Call them as a **statement**
- Example:

```
greet('Prof. Andersen')
```

Fruitful Functions

- Functions that give a **value**
- Call them in an **expression**
- Example:

```
x = round(2.56,1)
```

print vs. return

- Sometimes appear to have similar behavior

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

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

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

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

print vs. return

Print

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

Return

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

Python Interactive Shell >>>

- executes both *statements* and *expressions*
- if *expression*, prints value (if it exists)

```
>>> 2+2
4
>>>
```

prints to screen

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

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

prints to screen

So why do these behave similarly?

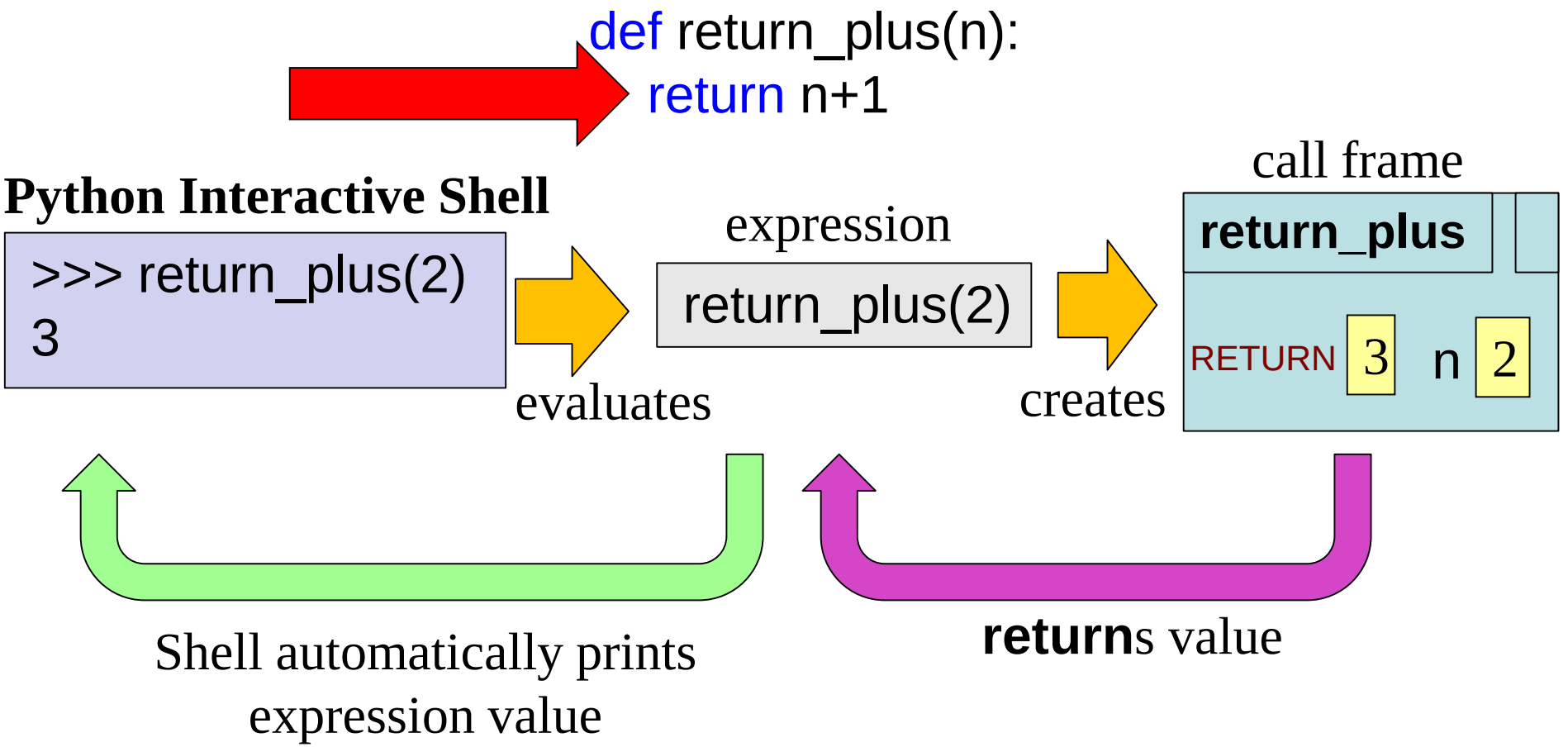
```
def print_plus(n):
    print n+1

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

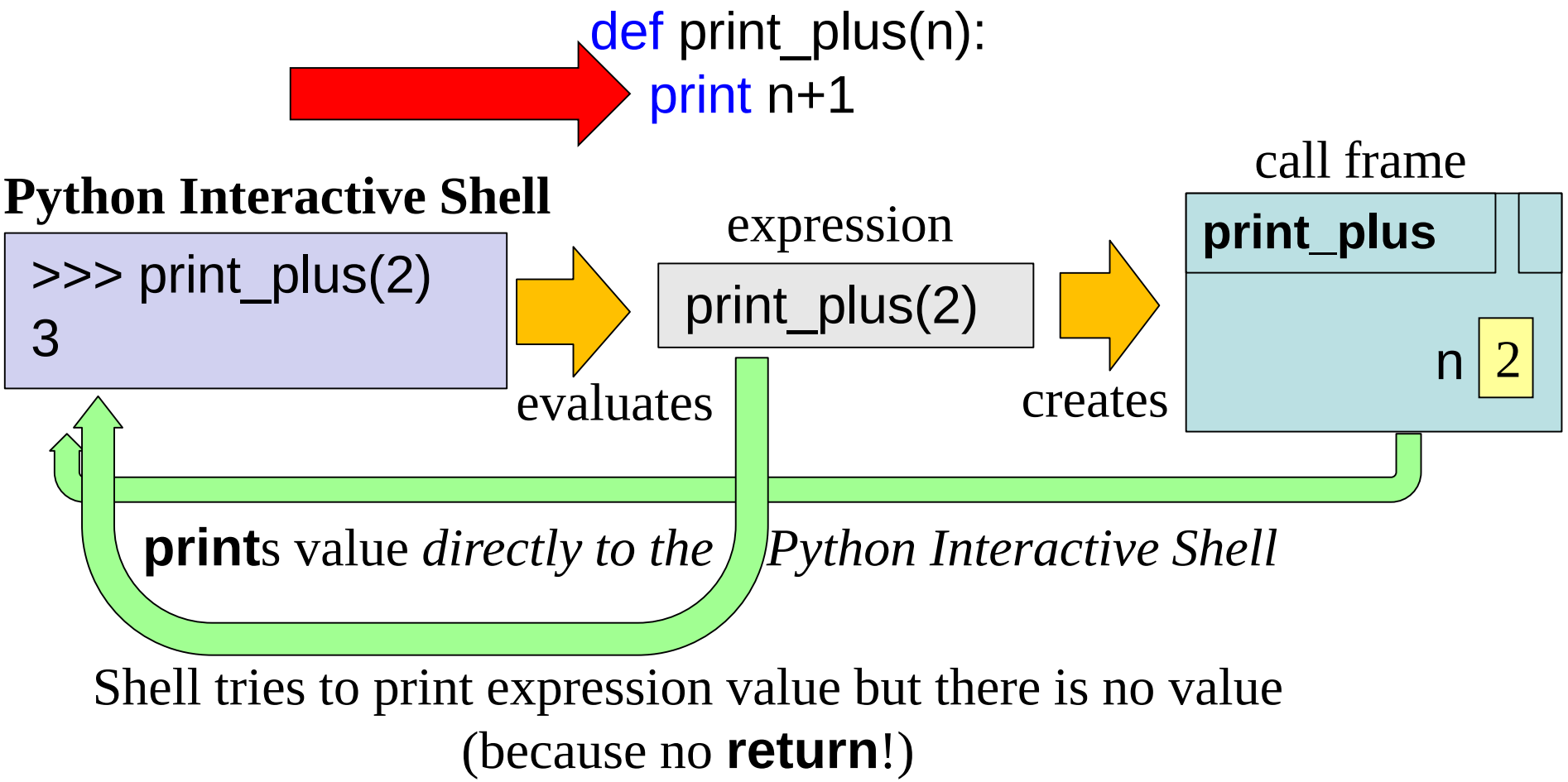
```
def return_plus(n):
    return n+1

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

return


def return_plus(n):
return n+1
Python Interactive Shell
>>> return_plus(2)
3
evaluates
expression
return_plus(2)
creates
call frame
return_plus
RETURN
3
n
2
Shell automatically prints expression value
returns value

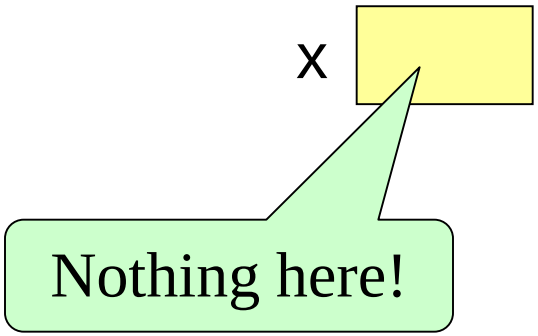
print


def print_plus(n):
print n+1
Python Interactive Shell
>>> print_plus(2)
3
evaluates
expression
print_plus(2)
creates
call frame
print_plus
n 2
prints value directly to the Python Interactive Shell
Shell tries to print expression value but there is no value
(because no return!)

print vs. return

Print

```
def print_plus(n):
    print n+1
>>> x = print_plus(2)
3
>>>
```



Return

```
def return_plus(n):
    return n+1
>>> x = return_plus(2)
>>>
```