Lists
Intro. to Functions

Computer Science 111
Boston University

Vahid Azadeh-Ranjbar, Ph.D.

Lists

Recall: A string is a sequence of characters.

'hello'

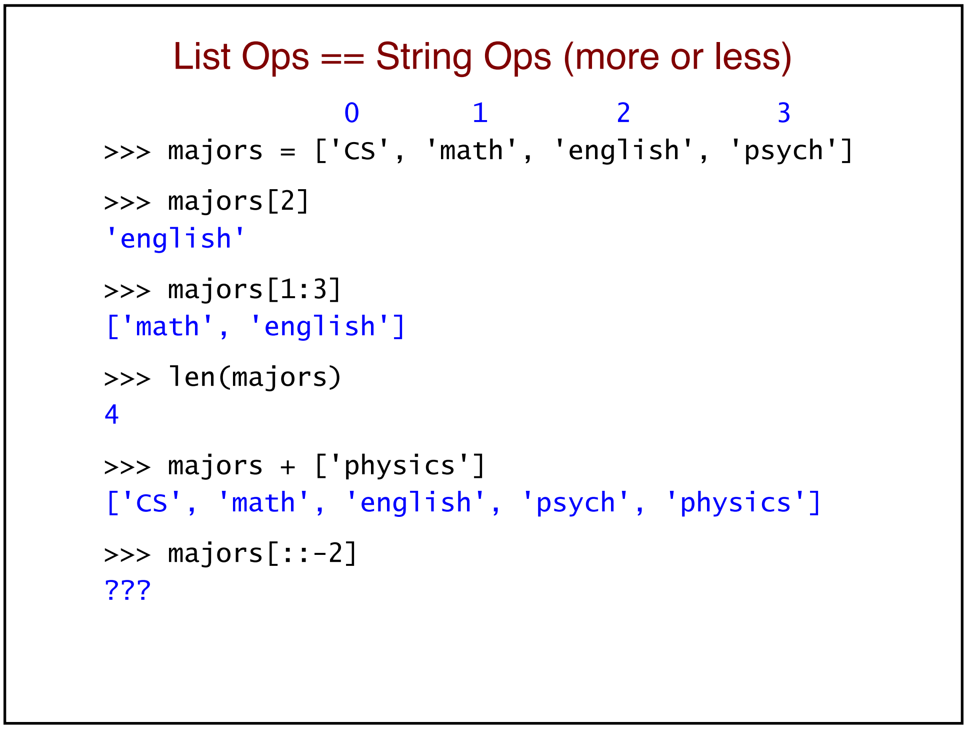
A list is a sequence of *arbitrary* values (the list's *elements*).

[2**,** 4**,** 6**,** 8**]**

['CS'**,** 'math'**,** 'english'**,** 'psych'**]**

A list can include values of different types:

['Star Wars', 1977, 'PG', [35.9, 460.9]]



List Ops == String Ops (more or less)

```
               0       1        2         3
>>> majors = ['CS', 'math', 'english', 'psych']

>>> majors[2]
'english'

>>> majors[1:3]
['math', 'english']

>>> len(majors)
4

>>> majors + ['physics']
['CS', 'math', 'english', 'psych', 'physics']

>>> majors[::-2]
???
```

List Ops == String Ops (more or less)

```
>>> majors = ['CS', 'math', 'english', 'psych']
>>> majors[2]
'english'
>>> majors[1:3]
['math', 'english']
>>> len(majors)
4
>>> majors + ['physics']
['CS', 'math', 'english', 'psych', 'physics']
>>> majors[::-2]
['psych', 'math']
```

What is the output of the following program?

```
mylist = [1, 2, [3, 4, 5]]
print(mylist[1], mylist[1:2])
```

A. `2 2 3`

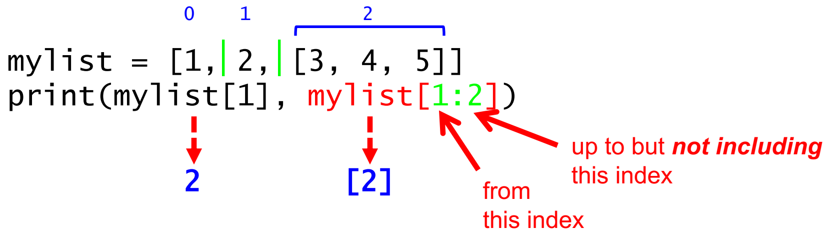
B. `2 [2, 3]`

C. `2 2`

D. `2 2 [3, 4, 5]`

E. none of these

What is the output of the following program?



```
          0   1        2
mylist = [1, 2, [3, 4, 5]]
print(mylist[1], mylist[1:2])
```

2

[2]

from
this index

up to but ***not including***
this index

A. `2 2 3`

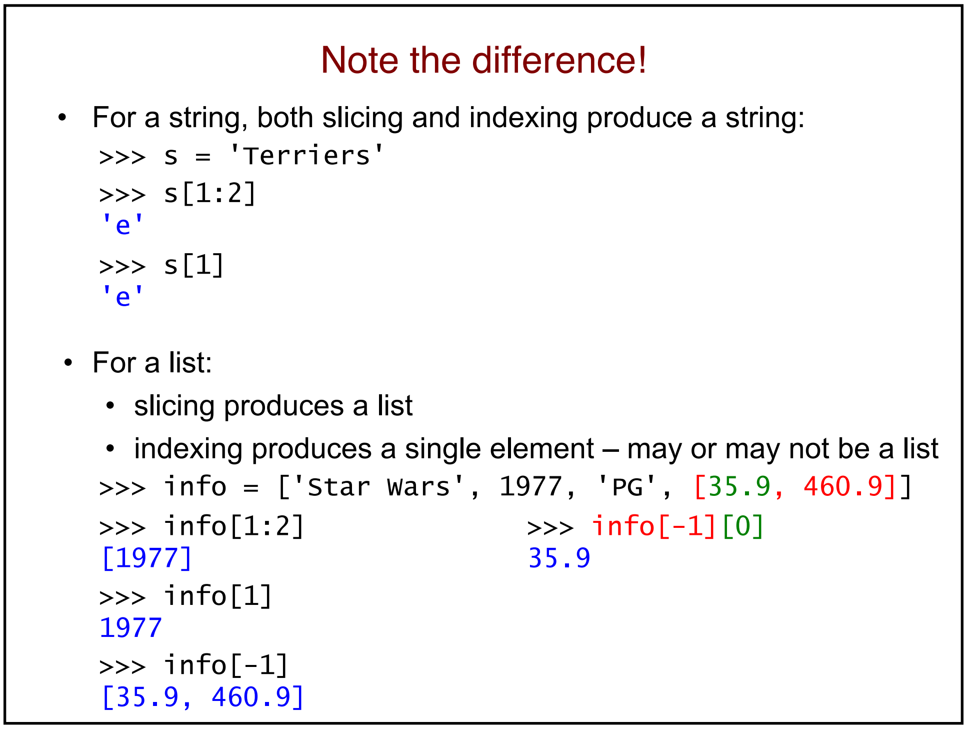
B. `2 [2, 3]`

C. `2 2`

D. `2 2 [3, 4, 5]`

E. **none of these!!** **`2 [2]`**

Slicing a list always
produces a list!



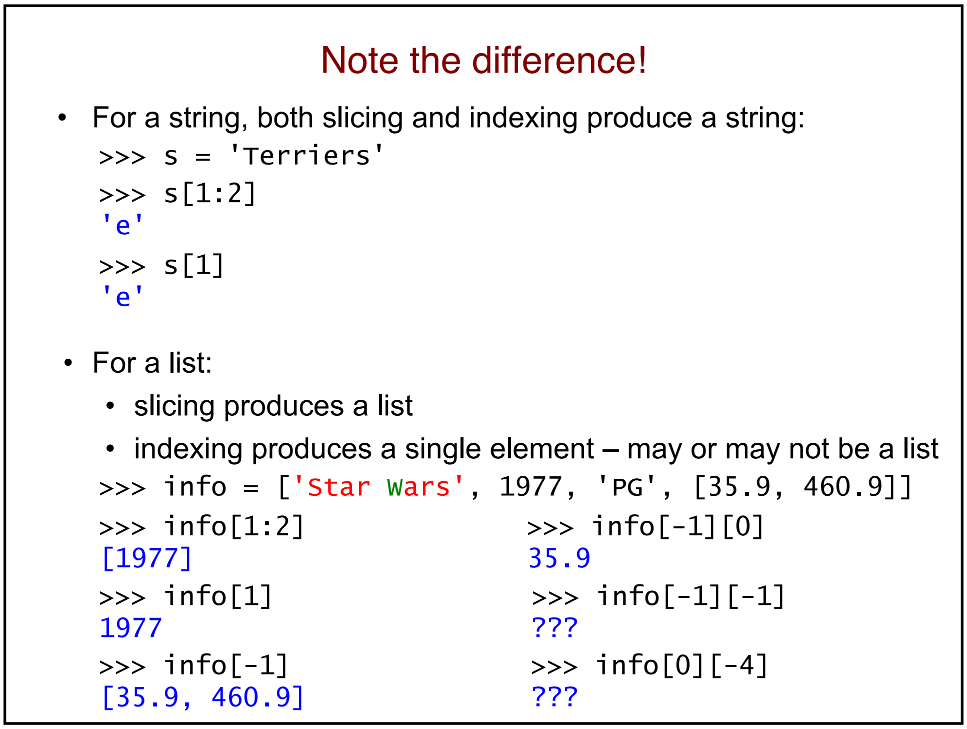
Note the difference!

- For a string, both slicing and indexing produce a string:

```
>>> s = 'Terriers'
>>> s[1:2]
'e'
>>> s[1]
'e'
```

- For a list:
  - slicing produces a list
  - indexing produces a single element – may or may not be a list

```
>>> info = ['Star Wars', 1977, 'PG', [35.9, 460.9]]
>>> info[1:2]                 >>> info[-1][0]
[1977]                        35.9
>>> info[1]
1977
>>> info[-1]
[35.9, 460.9]
```



Note the difference!

- For a string, both slicing and indexing produce a string:

```
>>> s = 'Terriers'
>>> s[1:2]
'e'
>>> s[1]
'e'
```

- For a list:
  - slicing produces a list
  - indexing produces a single element – may or may not be a list

```
>>> info = ['Star Wars', 1977, 'PG', [35.9, 460.9]]
>>> info[1:2]                 >>> info[-1][0]
[1977]                        35.9
>>> info[1]                   >>> info[-1][-1]
1977                          ???
>>> info[-1]                  >>> info[0][-4]
[35.9, 460.9]                 ???
```

Note the difference!

- For a string, both slicing and indexing produce a string:

```
>>> s = 'Terriers'
>>> s[1:2]
'e'
>>> s[1]
'e'
```

- For a list:
  - slicing produces a list
  - indexing produces a single element – may or may not be a list

```
>>> info = ['Star Wars', 1977, 'PG', [35.9, 460.9]]
>>> info[1:2]                >>> info[-1][0]
[1977]                       35.9
>>> info[1]                  >>> info[-1][-1]
1977                         460.9
>>> info[-1]                 >>> info[0][-4]
[35.9, 460.9]                ???
```

Note the difference!

- For a string, both slicing and indexing produce a string:

```
>>> s = 'Terriers'
>>> s[1:2]
'e'
>>> s[1]
'e'
```

- For a list:
  - slicing produces a list
  - indexing produces a single element – may or may not be a list

```
>>> info = ['Star Wars', 1977, 'PG', [35.9, 460.9]]
>>> info[1:2]                >>> info[-1][0]
[1977]                       35.9
>>> info[1]                  >>> info[-1][-1]
1977                         460.9
>>> info[-1]                 >>> info[0][-4]
[35.9, 460.9]                'W'
```

How could you fill in the blank to produce [103, 111]?

```
intro_cs = [101, 103, 105, 108, 109, 111]

vahid_courses = ______________________
```

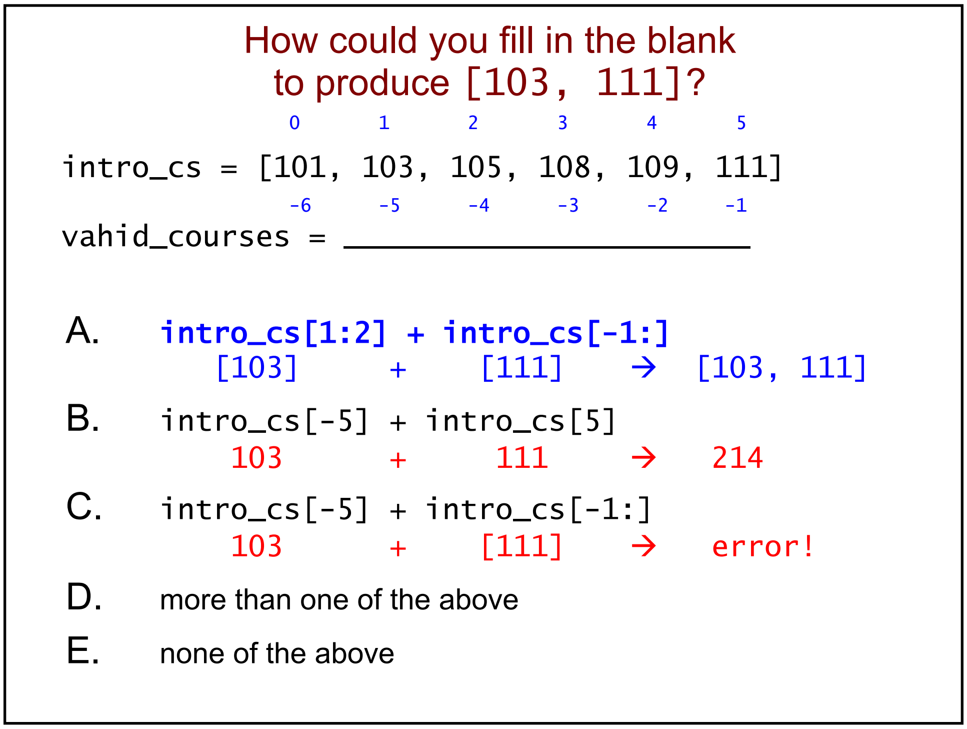
A. `intro_cs[1:2] + intro_cs[-1:]`

B. `intro_cs[-5] + intro_cs[5]`

C. `intro_cs[-5] + intro_cs[-1:]`

D. more than one of the above

E. none of the above



How could you fill in the blank to produce [103, 111]?

```
                0        1        2        3        4        5
intro_cs = [101, 103, 105, 108, 109, 111]
                -6       -5       -4       -3       -2       -1
vahid_courses = ______________________
```

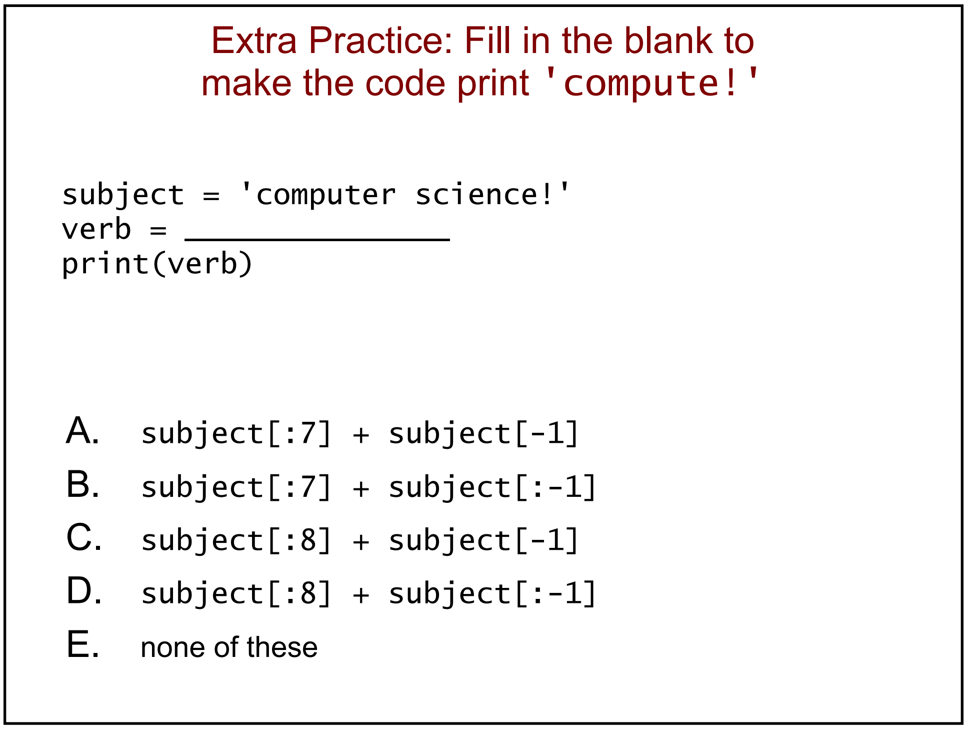
A. **intro_cs[1:2] + intro_cs[-1:]**
```
      [103]      +     [111]     →   [103, 111]
```

B. intro_cs[-5] + intro_cs[5]
```
       103       +      111      →    214
```

C. intro_cs[-5] + intro_cs[-1:]
```
       103       +     [111]     →    error!
```

D. more than one of the above

E. none of the above



Extra Practice: Fill in the blank to make the code print 'compute!'

```
subject = 'computer science!'
verb = _______________
print(verb)
```

A. subject[:7] + subject[-1]

B. subject[:7] + subject[:-1]

C. subject[:8] + subject[-1]

D. subject[:8] + subject[:-1]

E. none of these

Extra Practice: Fill in the blank to make the code print 'compute!'

```
subject = 'computer science!'
verb = _______________
print(verb)
```

A. **subject[:7] + subject[-1]**

B. subject[:7] + subject[:-1]

C. subject[:8] + subject[-1]

D. subject[:8] + subject[:-1]

E. none of these

Extra practice from the textbook authors!

```
pi = [3,1,4,1,5,9]
L = [ 'pi', "isn't", [4,2] ]
M = 'You need parentheses for chemistry !'
     0   4   8   12  16  20  24  28  32
```

Part 1

What is `len(pi)`

What is `len(L)`

What is `len(L[1])`

What is `pi[2:4]`

What slice of `pi` is `[3,1,4]`

What slice of `pi` is `[3,4,5]`

Part 2

What is `L[0]`

These two are different!

What is `L[0:1]`

What is `L[0][1]`

What slice of `M` is `'try'`?

is `'shoe'`?

What is `M[9:15]`

What is `M[::5]`

What is `M[::-5]`

Extra!

What are `pi[0]*(pi[1] + pi[2])` and `pi[0]*(pi[1:2] + pi[2:3])`?

These two are different, too…

Extra practice from the textbook authors!

```
pi = [3,1,4,1,5,9]
L = [ 'pi', "isn't", [4,2] ]
M = 'You need parentheses for chemistry !'
       0   4   8   12  16  20  24  28  32
```

Part 1

What is `len(pi)` 6

What is `len(L)` 3

What is `len(L[1])` 5

What is `pi[2:4]` [4, 1]

What slice of `pi` is `[3,1,4]` pi[:3]

What slice of `pi` is `[3,4,5]` pi[::2]

Part 2

What is `L[0]` 'pi'

What is `L[0:1]` ['pi']

These two are different!

What is `L[0][1]` 'i'

What slice of `M` is `'try'`? M[31:34]

is `'shoe'`? M[30:17:-4]

What is `M[9:15]` 'parent'

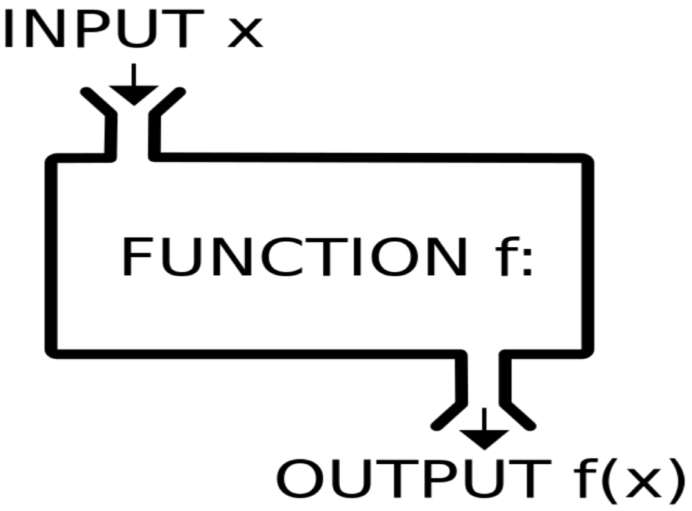
What is `M[::5]` 'Yeah cs!'

What is `M[::-5]` '!sc haeY'

Extra! What are `pi[0]*(pi[1] + pi[2])` and `pi[0]*(pi[1:2] + pi[2:3])`?

These two are different, too… 15 [1, 4, 1, 4, 1, 4]

Functions

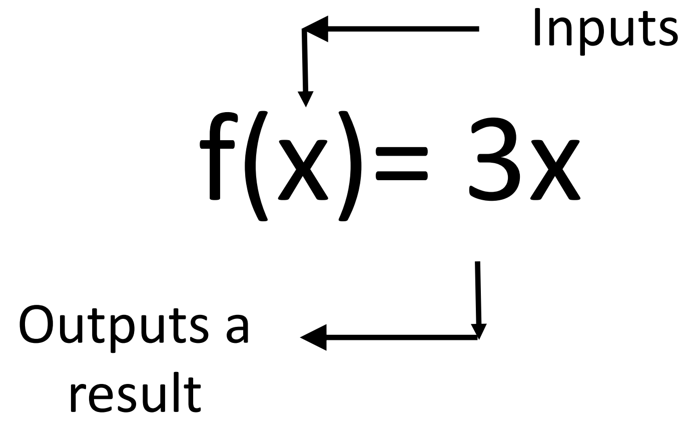


INPUT x

FUNCTION f:

OUTPUT f(x)

Algebraic Function

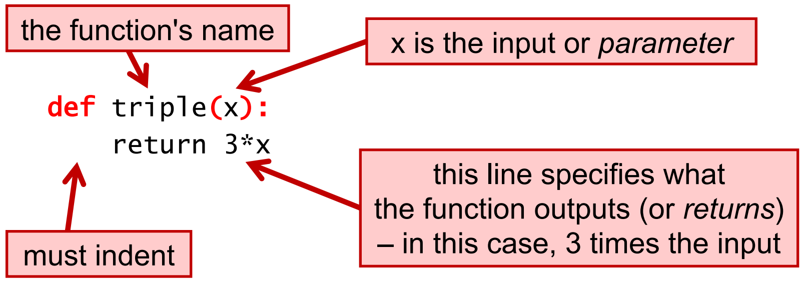


Inputs

$$f(x)= 3x$$

Outputs a result

Defining a Function



the function's name

x is the input or *parameter*

```
def triple(x):
    return 3*x
```

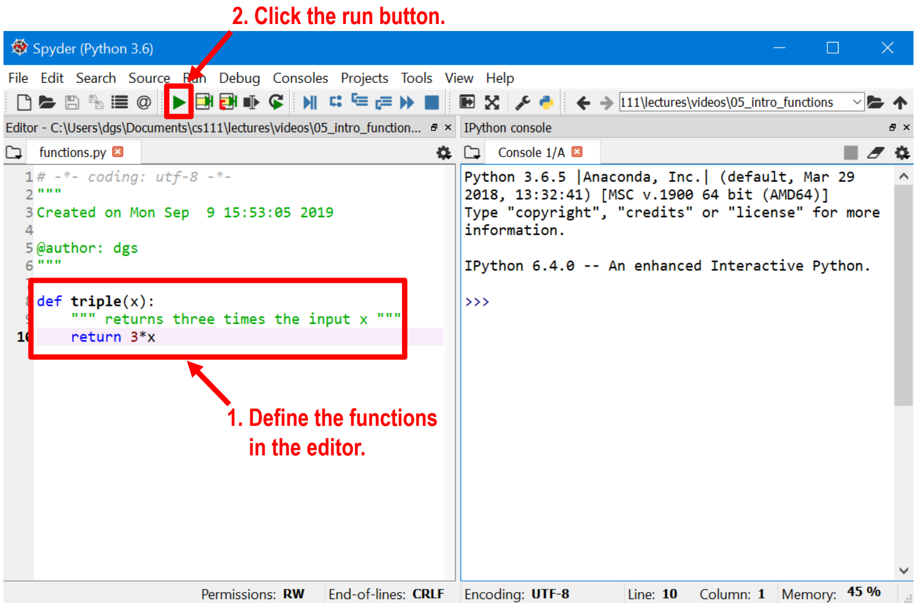
must indent

this line specifies what the function outputs (or *returns*) – in this case, 3 times the input

- Once we define a function, we can call it:

```
>>> triple(3)
9
>>> triple(10)
30
>>> triple(0.5)
1.5
```

Working with Functions in Spyder


Spyder (Python 3.6)
File Edit Search Source Run Debug Consoles Projects Tools View Help
111\lectures\videos\05_intro_functions
Editor - C:\Users\dgs\Documents\cs111\lectures\videos\05_intro_function...
functions.py
IPython console
Console 1/A
Permissions: RW End-of-lines: CRLF Encoding: UTF-8 Line: 10 Column: 1 Memory: 45 %

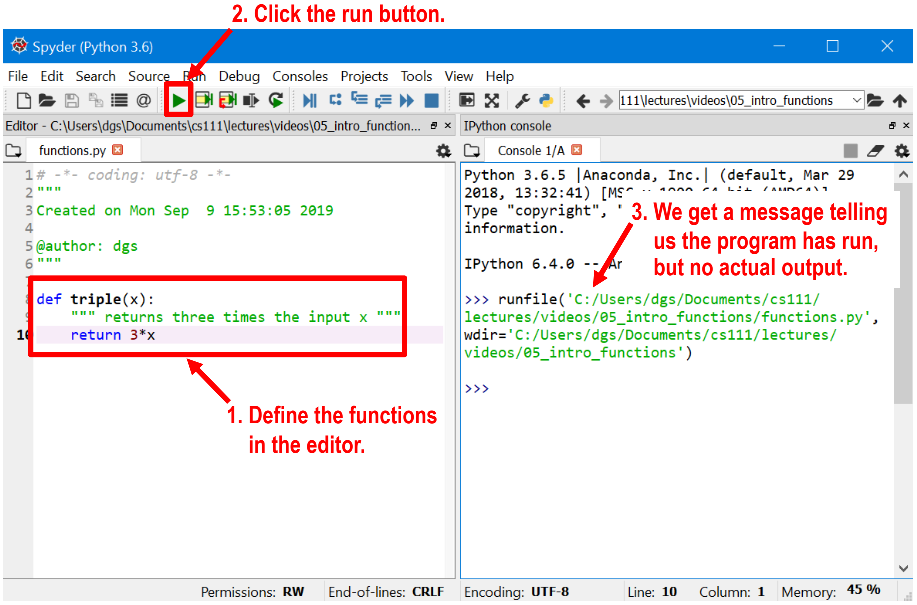
2. Click the run button.

```
# -*- coding: utf-8 -*-
"""
Created on Mon Sep  9 15:53:05 2019

@author: dgs
"""

def triple(x):
    """ returns three times the input x """
    return 3*x
```

```
Python 3.6.5 |Anaconda, Inc.| (default, Mar 29
2018, 13:32:41) [MSC v.1900 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more
information.

IPython 6.4.0 -- An enhanced Interactive Python.

>>>
```

1. Define the functions in the editor.

Working with Functions in Spyder


Spyder (Python 3.6)
File Edit Search Source Run Debug Consoles Projects Tools View Help
111\lectures\videos\05_intro_functions
Editor - C:\Users\dgs\Documents\cs111\lectures\videos\05_intro_function...
functions.py
IPython console
Console 1/A
Permissions: RW End-of-lines: CRLF Encoding: UTF-8 Line: 10 Column: 1 Memory: 45 %

2. Click the run button.

```
# -*- coding: utf-8 -*-
"""
Created on Mon Sep  9 15:53:05 2019

@author: dgs
"""

def triple(x):
    """ returns three times the input x """
    return 3*x
```

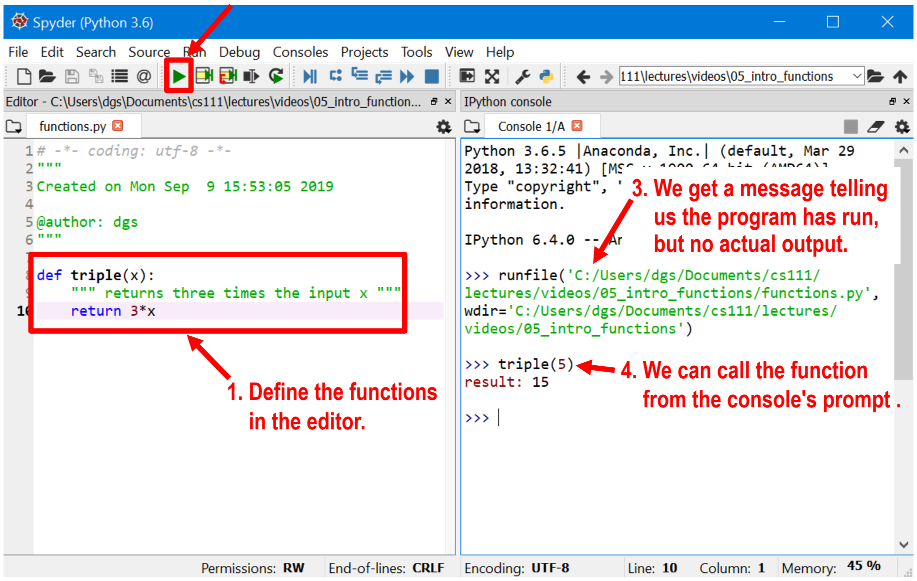
```
Python 3.6.5 |Anaconda, Inc.| (default, Mar 29
2018, 13:32:41) [MS
Type "copyright", '
information.

IPython 6.4.0 -- Ar

>>> runfile('C:/Users/dgs/Documents/cs111/
lectures/videos/05_intro_functions/functions.py',
wdir='C:/Users/dgs/Documents/cs111/lectures/
videos/05_intro_functions')

>>>
```

3. We get a message telling us the program has run, but no actual output.

1. Define the functions in the editor.

Working with Functions in Spyder

2. Click the run button.


Spyder (Python 3.6)
File Edit Search Source Run Debug Consoles Projects Tools View Help
111\lectures\videos\05_intro_functions
IPython console
functions.py
Console 1/A

Editor - C:\Users\dgs\Documents\cs111\lectures\videos\05_intro_function...

```
# -*- coding: utf-8 -*-
"""
Created on Mon Sep  9 15:53:05 2019

@author: dgs
"""

def triple(x):
    """ returns three times the input x """
    return 3*x
```

```
Python 3.6.5 |Anaconda, Inc.| (default, Mar 29
2018, 13:32:41) [MSC v.1900 64 bit (AMD64)]
Type "copyright", '
information.

IPython 6.4.0 -- Ar

>>> runfile('C:/Users/dgs/Documents/cs111/
lectures/videos/05_intro_functions/functions.py',
wdir='C:/Users/dgs/Documents/cs111/lectures/
videos/05_intro_functions')

>>> triple(5)
result: 15

>>> |
```

Permissions: RW End-of-lines: CRLF Encoding: UTF-8 Line: 10 Column: 1 Memory: 45 %

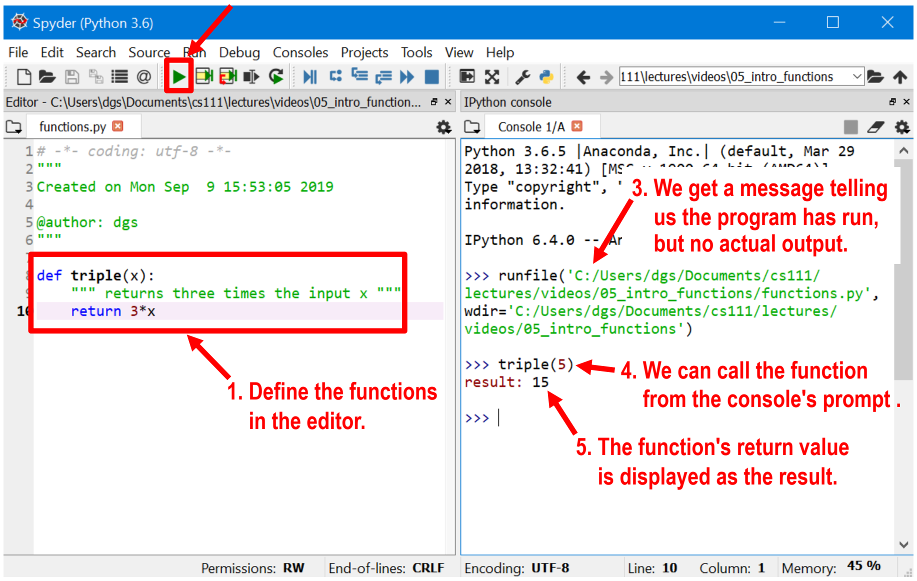
3. We get a message telling us the program has run, but no actual output.

4. We can call the function from the console's prompt .

1. Define the functions in the editor.

Working with Functions in Spyder

2. Click the run button.


Spyder (Python 3.6)
File Edit Search Source Run Debug Consoles Projects Tools View Help
111\lectures\videos\05_intro_functions
IPython console
functions.py
Console 1/A

Editor - C:\Users\dgs\Documents\cs111\lectures\videos\05_intro_function...

```
# -*- coding: utf-8 -*-
"""
Created on Mon Sep  9 15:53:05 2019

@author: dgs
"""

def triple(x):
    """ returns three times the input x """
    return 3*x
```

```
Python 3.6.5 |Anaconda, Inc.| (default, Mar 29
2018, 13:32:41) [MSC v.1900 64 bit (AMD64)]
Type "copyright", '
information.

IPython 6.4.0 -- Ar

>>> runfile('C:/Users/dgs/Documents/cs111/
lectures/videos/05_intro_functions/functions.py',
wdir='C:/Users/dgs/Documents/cs111/lectures/
videos/05_intro_functions')

>>> triple(5)
result: 15

>>> |
```

Permissions: RW End-of-lines: CRLF Encoding: UTF-8 Line: 10 Column: 1 Memory: 45 %

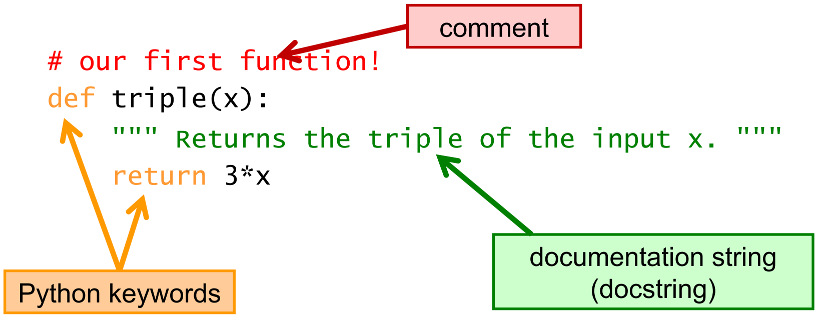
3. We get a message telling us the program has run, but no actual output.

4. We can call the function from the console's prompt .

1. Define the functions in the editor.

5. The function's return value is displayed as the result.

Other Details



```
# our first function!
def triple(x):
    """ Returns the triple of the input x. """
    return 3*x
```

comment

Python keywords

documentation string
(docstring)

- Python uses color-coding to distinguish program components.
- Always use a *docstring* to explain what the function does.
  - surrounded by triple quotes, beginning on the second line
  - `help(`*function name*`)` retrieves it
- Other (non-docstring) comments can be included as needed.

Functions With String Inputs

```
def undo(s):
    """ Adds the prefix "un" to the input s. """
    return 'un' + s

def redo(s):
    """ Adds the prefix "re" to the input s. """
    return 're' + s
```

The evil "un" people!
(from the PBS kids show *Between the Lions*)

- Examples:

```
>>> undo('plugged')
'unplugged'
>>> undo('zipped')
'unzipped'
>>> redo('submit')
'resubmit'
>>> redo(undo('zipped'))      # redo('unzipped')
'reunzipped'
```

Multiple Lines, Multiple Parameters

```
def circle_area(diam):
    """ Computes the area of a circle
        with a diameter diam.
    """
    radius = diam / 2
    area = 3.14159 * (radius**2)
    return area

def rect_perim(l, w):
    """ Computes the perimeter of a rectangle
        with length l and width w.
    """
    return 2*l + 2*w
```

- Examples:

```
>>> rect_perim(5, 7)
24
>>> circle_area(20)
314.159
```

Function and Function Call in the Same File

```
def circle_area(diam):
    """ Computes the area of a circle
        with a diameter diam.
    """
    radius = diam / 2
    area = 3.14159 * (radius**2)
    return area

def rect_perim(l, w):
    """ Computes the perimeter of a rectangle
        with length l and width w.
    """
    return 2*l + 2*w

print(rect_perim(20, 8))     # Why is print needed?
```

- Defines two functions, but only one gets called when we run the program.
- We can still call either of them from the Console after running the program.

Multiple Lines, Multiple Parameters

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


print(calculate(3, 2))
```

What is the output of this code?

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


print(calculate(3, 2))
```

x	y	a	b

On paper, make a table for the values of your variables!

A. 5

B. 9

C. 4

D. 3

E. 8

What is the output of this code?

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


print(calculate(3, 2))
```

x	y	a	b

A. 5

B. 9

C. 4

D. 3

E. 8

What is the output of this code?

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


print(calculate(3, 2))
```

x	y	a	b

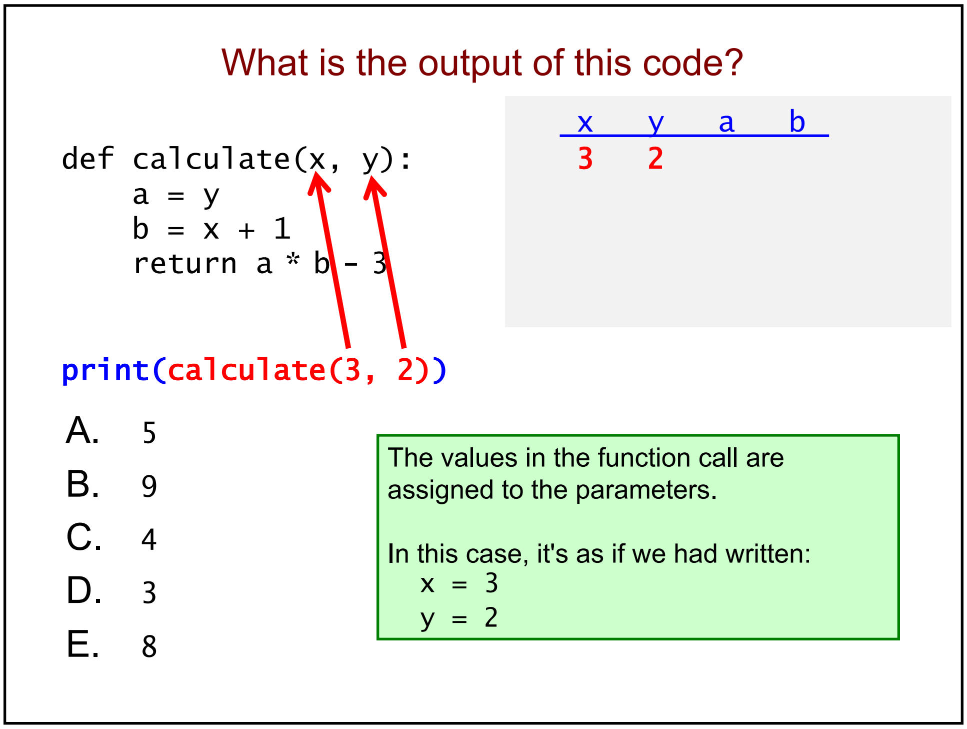
A. 5

B. 9

C. 4

D. 3

E. 8



What is the output of this code?

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


print(calculate(3, 2))
```

x	y	a	b
3	2		

A. 5

B. 9

C. 4

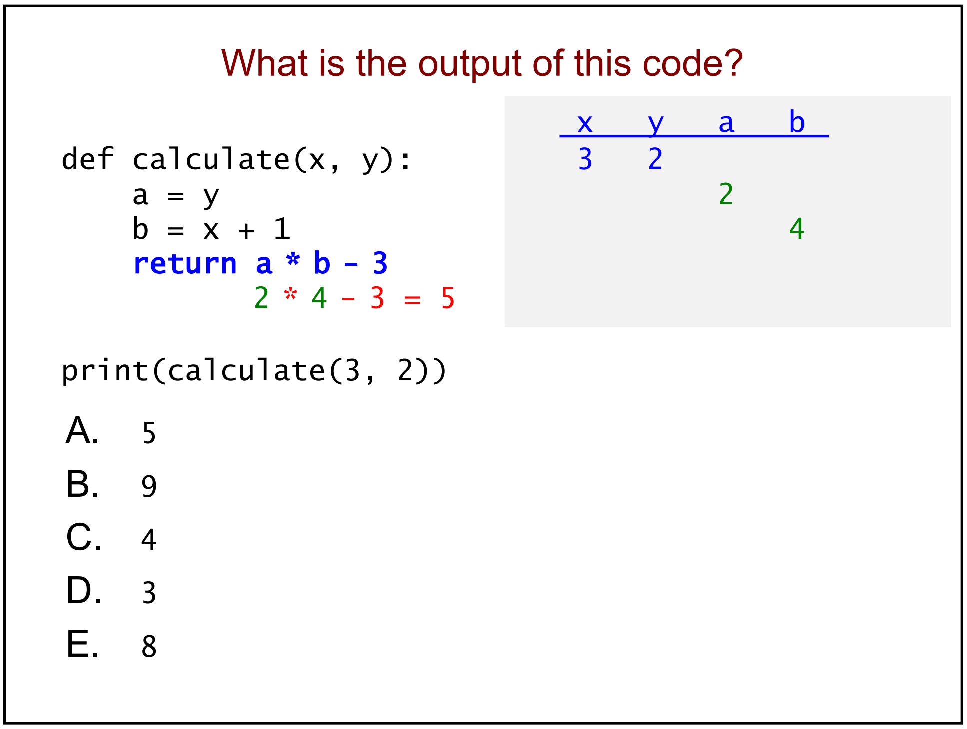
D. 3

E. 8

The values in the function call are assigned to the parameters.

In this case, it's as if we had written:

```
   x = 3
   y = 2
```



What is the output of this code?

```
def calculate(x, y):
    a = y
    b = x + 1
    return a * b - 3
           2 * 4 - 3 = 5

print(calculate(3, 2))
```

x	y	a	b
3	2		
		2	
			4

A. 5

B. 9

C. 4

D. 3

E. 8

What is the output of this code?

```
def calculate(x, y):
    a = y
    b = x + 1
    return a * b - 3
           2 * 4 - 3 = 5

print(calculate(3, 2))    # print(5)
```

x	y	a	b
3	2		
		2	
			4

A. 5

B. 9

C. 4

D. 3

E. 8

Practice Writing a Function

- Write a function `middle_elem(values)` that:
  - takes a list `values` that has at least one element
  - returns the element in the middle of the list
    - when there are two middle elements,
      return the one closer to the end
  - examples:

```
>>> middle_elem([2, 6, 3])
6
>>> middle_elem([7, 3, 1, 2, 4, 9])
2
```

Practice Writing a Function

- Write a function `middle_elem(values)` that:
  - takes a list `values` that has at least one element
  - returns the element in the middle of the list
    - when there are two middle elements, return the one closer to the end
  - examples:

```
>>> middle_elem([2, 6, 3])
6
>>> middle_elem([7, 3, 1, 2, 4, 9])
2
```

```
def middle_elem(values):
    middle_index = _____________
    return _______________
```

Practice Writing a Function

- Write a function `middle_elem(values)` that:
  - takes a list `values` that has at least one element
  - returns the element in the middle of the list
    - when there are two middle elements,
      return the one closer to the end
  - examples:

```
>>> middle_elem([2, 6, 3])
6
>>> middle_elem([7, 3, 1, 2, 4, 9])
2
```

```
def middle_elem(values):
    middle_index = len(values) // 2
    return values[middle_index]
```