



# Intro To Python

Code Afrique Team



## What is *Python*?

- A language to tell a computer what to do
- Computers are stupid---they need to be told *EXACTLY* what to do
  - There are many programming languages
  - Python is good for learning to program

# Basic concept: Functions

- For example:

- $f(x) = 3x + 4$

- $g(x) = \sqrt{x}$

- $sum(x, y) = x + y$

We call  $x$  an “input parameter”

$sum$  takes two input parameters



# Functions in Python

```
def f(x): ←————— Function “header”  
    return 3 * x + 4 ←————— “Body”
```

```
def square(x): ←————— Another function header  
    return x * x ←————— Another body
```

# Function Syntax

*x* is the "input parameter"

```
def f(x):
```

← Function "header"

```
    return 3 * x + 4
```

← "Body"

```
def square(x):
```

← Another function header

```
    return x * x
```

← Another body

# Using Spyder

Step 1: Enter functions in this box

The screenshot displays the Spyder Python IDE interface. The left pane shows a code editor with two Python functions:

```
1 def f(x):  
2     return 3 * x + 4  
3  
4 def square(x):  
5     return x * x
```

The right pane shows the IPython console with the following output:

```
Python 3.7.0 (default, Jun 28 2018, 07:39:16)  
Type "copyright", "credits" or "license" for more information.  
  
IPython 7.2.0 -- An enhanced Interactive Python.  
  
In [1]: |
```

The status bar at the bottom indicates: Permissions: RW, End-of-lines: LF, Encoding: ASCII, Line: 5, Column: 17, Memory: 63 %.

# Using Spyder

Step 2: Click on the green triangle (don't forget)

The screenshot displays the Spyder Python IDE interface. The top toolbar contains various icons, including a green play button (run) which is highlighted by a blue callout bubble. The main editor window shows a Python script with the following code:

```
1 def f(x):  
2     return 3 * x + 4  
3  
4 def square(x):  
5     return x * x
```

The IPython console window on the right shows the following output:

```
Python 3.7.0 (default, Jun 28 2018, 07:39:16)  
Type "copyright", "credits" or "license" for more information.  
  
IPython 7.2.0 -- An enhanced Interactive Python.  
  
In [1]: |
```

The status bar at the bottom of the IDE displays the following information: Permissions: RW, End-of-lines: LF, Encoding: ASCII, Line: 5, Column: 17, Memory: 63 %.

# Using Spyder

Step 2: Click on the green triangle (don't forget)

The screenshot displays the Spyder Python IDE interface. The left pane shows a code editor with the following Python code:

```
1 def f(x):  
2     return 3 * x + 4  
3  
4 def square(x):  
5     return x * x
```

The right pane shows the IPython console with the following output:

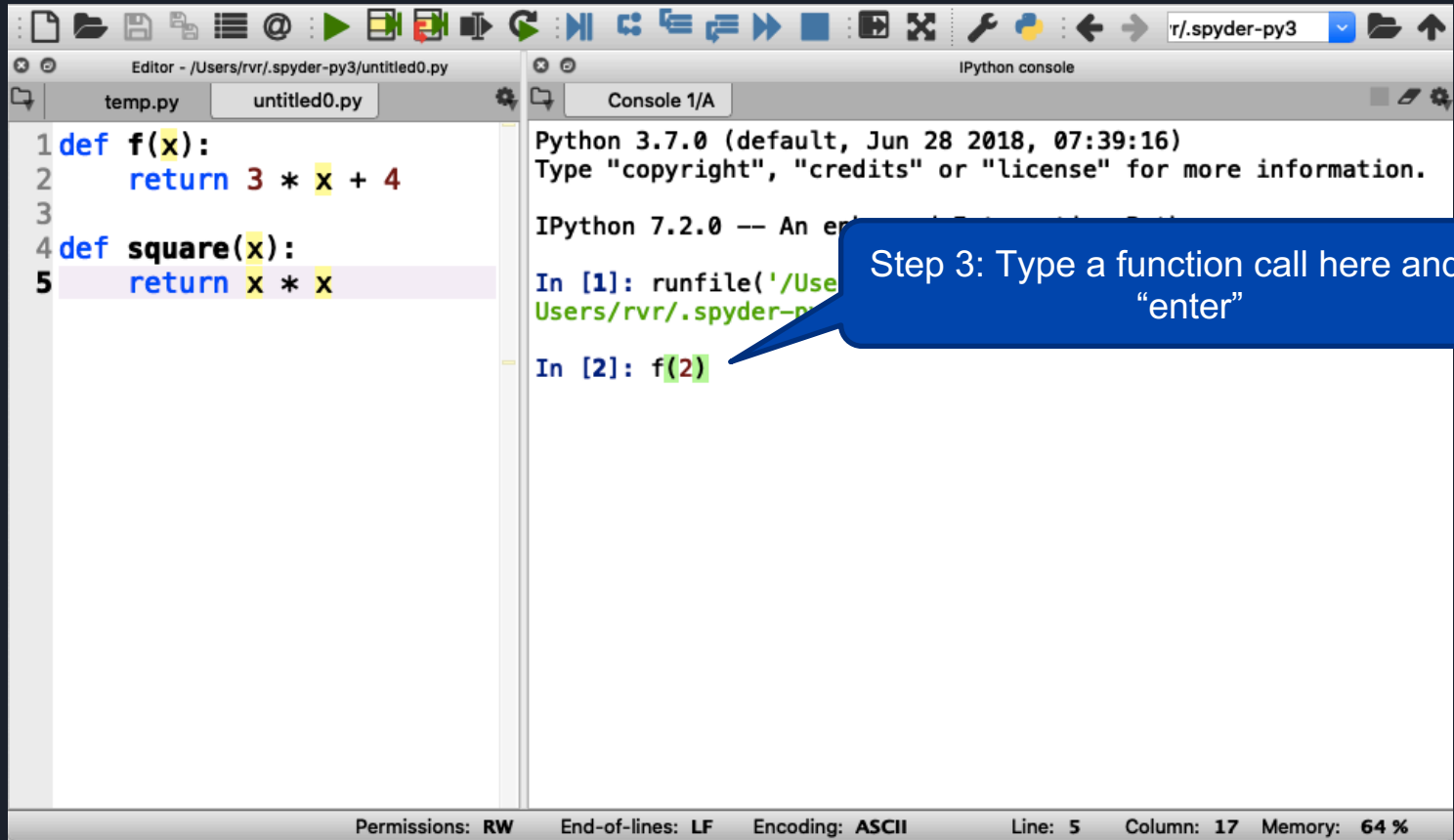
```
Python 3.7.0 (default, Jun 28 2018, 07:39:16)  
Type "copyright", "credits" or "license" for more information.  
  
IPython 7.2.0 -- An enhanced Interactive Python.  
  
In [1]: runfile('/Users/rvr/.spyder-py3/untitled0.py', wdir='/  
Users/rvr/.spyder-py3')  
  
In [2]:
```

A yellow callout bubble points to the console output with the text: "Output (and error messages) appear in 'console' box".

Permissions: RW End-of-lines: LF Encoding: ASCII Line: 5 Column: 17 Memory: 64 %



# Using Spyder



The screenshot displays the Spyder Python IDE interface. The left pane shows a code editor with the following Python code:

```
1 def f(x):  
2     return 3 * x + 4  
3  
4 def square(x):  
5     return x * x
```

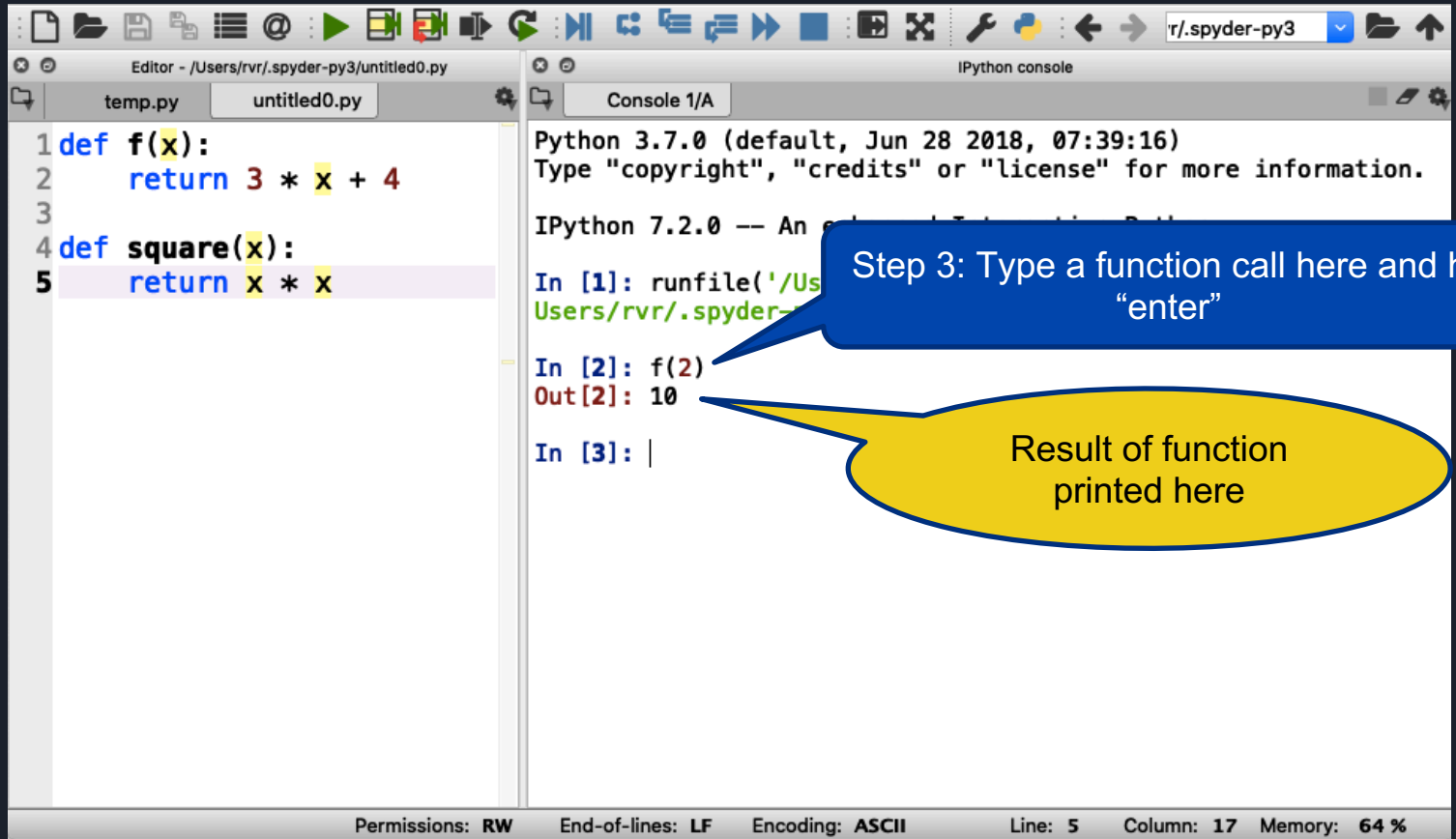
The right pane shows the IPython console with the following output:

```
Python 3.7.0 (default, Jun 28 2018, 07:39:16)  
Type "copyright", "credits" or "license" for more information.  
  
IPython 7.2.0 -- An enhanced Interactive Python.  
--> Enter '>>>' to enter the interactive shell.  
  
In [1]: runfile('/Users/rvr/.spyder-py3/untitled0.py', wdir='/Users/rvr/.spyder-py3')  
  
In [2]: f(2)
```

A blue callout box points to the console input, containing the text: "Step 3: Type a function call here and hit 'enter'".

At the bottom of the window, the status bar displays: Permissions: RW End-of-lines: LF Encoding: ASCII Line: 5 Column: 17 Memory: 64 %

# Using Spyder



The screenshot displays the Spyder Python IDE interface. The left pane shows a code editor with the following Python code:

```
1 def f(x):  
2     return 3 * x + 4  
3  
4 def square(x):  
5     return x * x
```

The right pane shows the IPython console with the following output:

```
Python 3.7.0 (default, Jun 28 2018, 07:39:16)  
Type "copyright", "credits" or "license" for more information.  
  
IPython 7.2.0 -- An enhanced Interactive Python.  
--> Enter '>>>' to enter the interactive mode and '<<<' to exit.  
  
In [1]: runfile('/Users/rvr/.spyder-py3/untitled0.py', wdir='/Users/rvr/.spyder-py3')  
Out[1]:  
  
In [2]: f(2)  
Out[2]: 10  
  
In [3]: |
```

Two callout boxes provide instructions:

- A blue callout box points to the line `In [2]: f(2)` in the console, containing the text: "Step 3: Type a function call here and hit 'enter'"
- A yellow callout box points to the line `Out[2]: 10` in the console, containing the text: "Result of function printed here"

The status bar at the bottom of the IDE shows: Permissions: RW End-of-lines: LF Encoding: ASCII Line: 5 Column: 17 Memory: 64 %



Try the following (*and take turns!*):

- $f(0)$
- $f(0.5)$
- $f(-3)$
- $square(-3)$
- $square(123456789)$
- $square(f(0))$

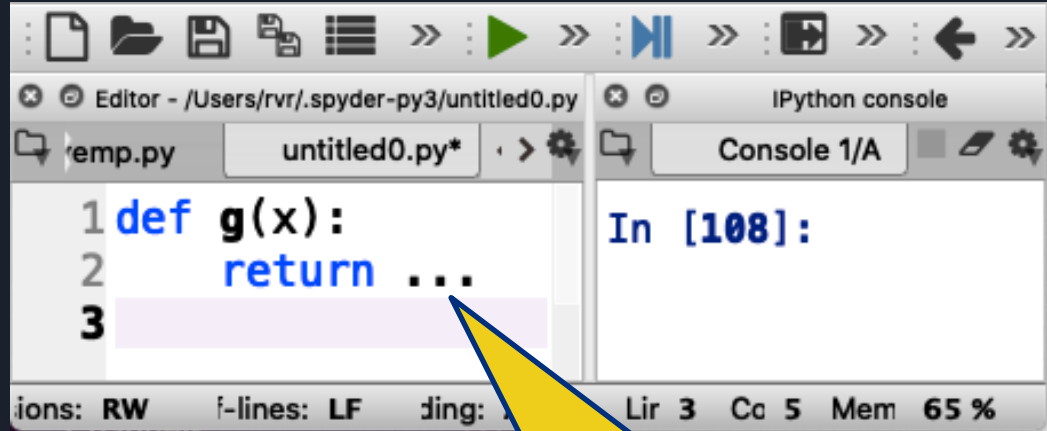
# Now define functions yourself

1. Write the following function in Python :

$$g(x) = x^2 + 2x + 3$$

2. Evaluate :

- $g(0)$
- $g(1)$
- $g(-0.5)$



```
1 def g(x):
2     return ...
3
```

In [108]:

*dot dot dot*  
Write your code here

# More time for you to play

1. Write a function  $add(x, y, z)$  that returns the sum of its three inputs (that is,  $x + y + z$ )

2. Evaluate the following:

- $add(1, 2, 3)$
- $add(1, 1, -2)$

```
def ...:
return ...
```

In [108]:

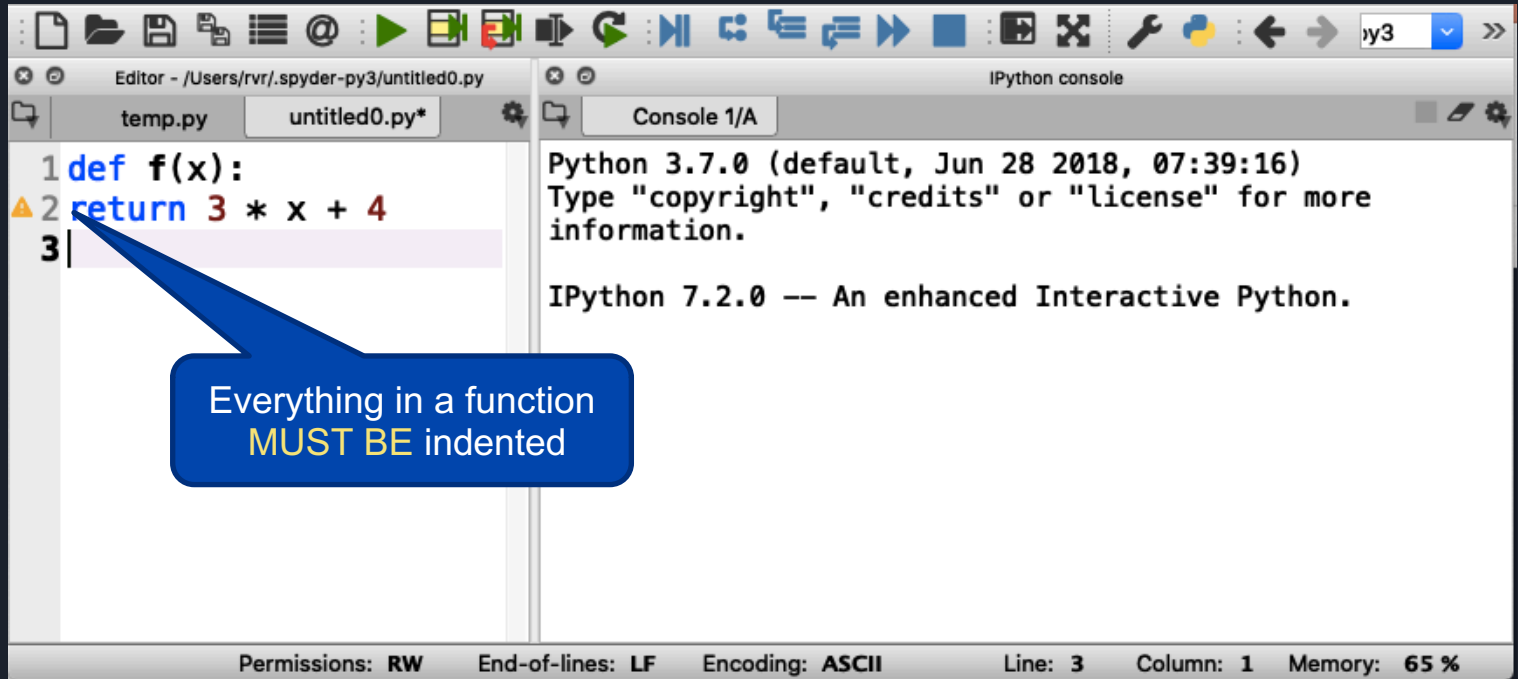
ions: RW f-lines: LF ding: ASC Lir 1 Cc 8 Mem 64%



## Errors and Bugs...

- Remember: computers are stupid
- Everything needs to be *EXACTLY* right

# Indentation error



The screenshot shows a Python IDE window with two panes. The left pane is a code editor showing a Python function definition:

```
1 def f(x):  
2 return 3 * x + 4  
3 |
```

The right pane is the IPython console, displaying the Python version and IPython version information:

```
Python 3.7.0 (default, Jun 28 2018, 07:39:16)  
Type "copyright", "credits" or "license" for more  
information.  
  
IPython 7.2.0 -- An enhanced Interactive Python.
```

A blue callout box points to the code in the editor, containing the text:

Everything in a function  
**MUST BE** indented

The status bar at the bottom of the IDE shows: Permissions: RW End-of-lines: LF Encoding: ASCII Line: 3 Column: 1 Memory: 65 %

# Indentation error

Hover mouse over triangle for information about the error

```
1 def f(x):  
2 return 3 * x + 4  
3 |
```

**Code analysis**  
expected an indented block

Python 3.7.0 (default, Jun 28 2018, 07:39:16)  
Type "copyright", "credits" or "license" for more information.

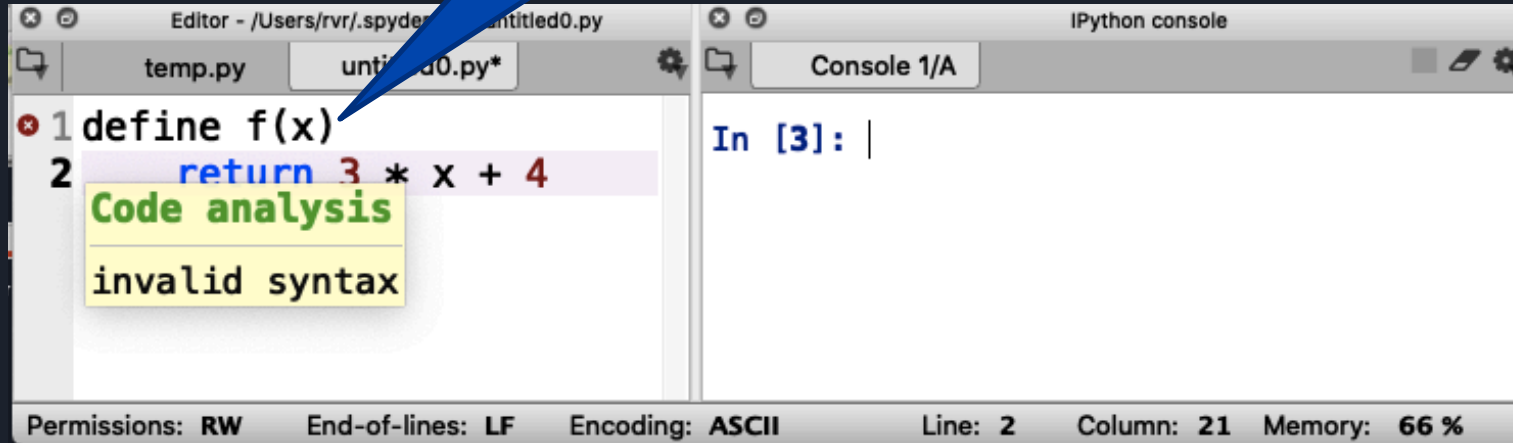
IPython 7.2.0 -- An enhanced Interactive Python.

Permissions: RW End-of-lines: LF Encoding: ASCII Line: 3 Column: 1 Memory: 65 %



# Syntax error

Missing :



The screenshot shows a code editor window with two tabs: 'temp.py' and 'untitled0.py\*'. The code in 'untitled0.py' is as follows:

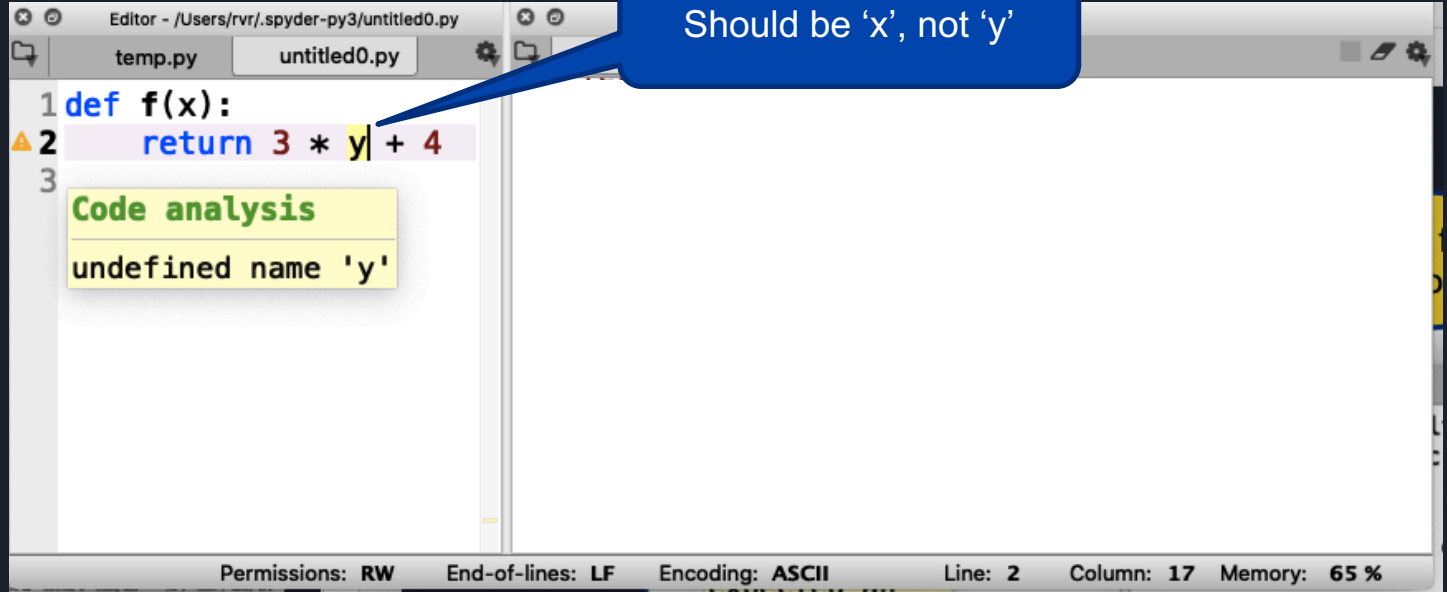
```
1 define f(x)
2   return 3 * x + 4
```

A red error icon is visible next to line 1. A blue callout bubble points to the missing colon at the end of line 1. A yellow tooltip box is overlaid on the code, containing the text:

```
Code analysis
invalid syntax
```

The right-hand side of the editor shows an IPython console window with the prompt 'In [3]: |'. The status bar at the bottom of the editor displays the following information: 'Permissions: RW', 'End-of-lines: LF', 'Encoding: ASCII', 'Line: 2', 'Column: 21', and 'Memory: 66 %'.

# Undefined name warning



The screenshot shows a code editor window with the following content:

```
1 def f(x):  
2     return 3 * y + 4  
3
```

A yellow warning box is displayed below the code, containing the text:

**Code analysis**  
undefined name 'y'

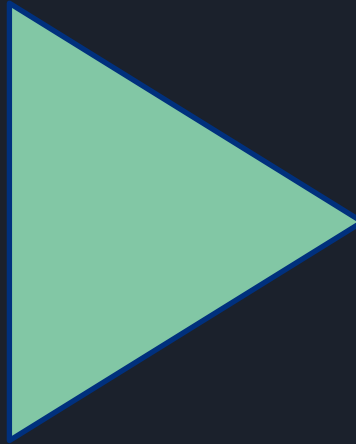
A blue callout bubble points to the variable 'y' in the code, containing the text: "Should be 'x', not 'y'".

The status bar at the bottom of the editor shows: Permissions: RW End-of-lines: LF Encoding: ASCII Line: 2 Column: 17 Memory: 65 %



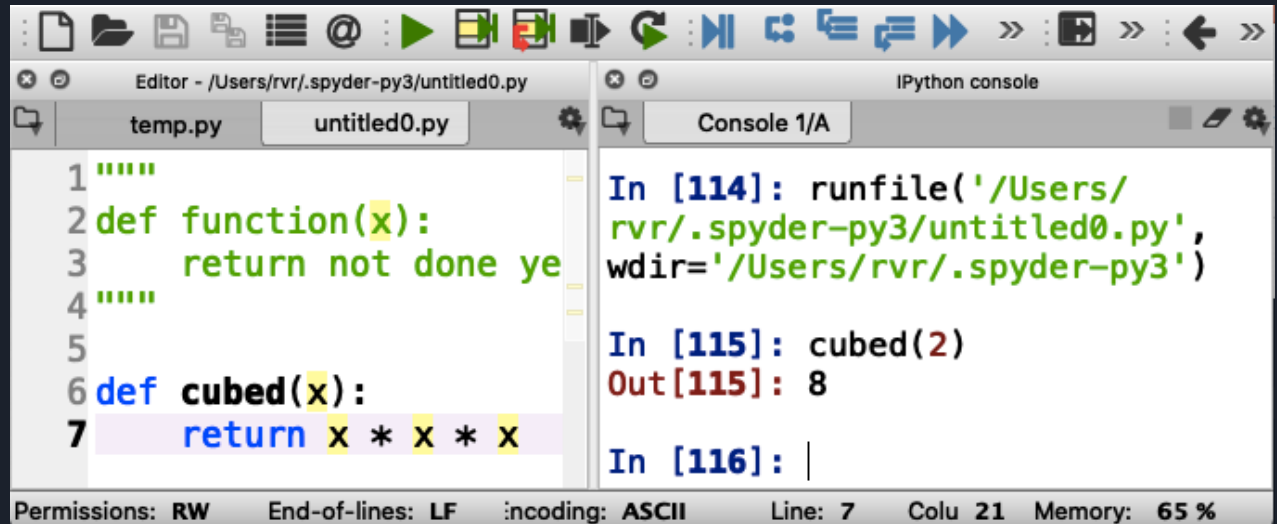
Remember: Keep hitting that green triangle

*You need to do that each time you change or fix your code!*



## Tip: “commenting out functions”

- If you want to save a function you are working on but it does not quite work yet, you can temporarily “comment it out”.
- To do this, place three quotes before and after the function at the beginning of the line



The screenshot shows the Spyder Python IDE interface. The left pane is the code editor, and the right pane is the IPython console.

```
1 """  
2 def function(x):  
3     return not done ye  
4 """  
5  
6 def cubed(x):  
7     return x * x * x
```

The console shows the following output:

```
In [114]: runfile('/Users/rvr/.spyder-py3/untitled0.py',  
           wdir='/Users/rvr/.spyder-py3')  
  
In [115]: cubed(2)  
Out[115]: 8  
  
In [116]: |
```

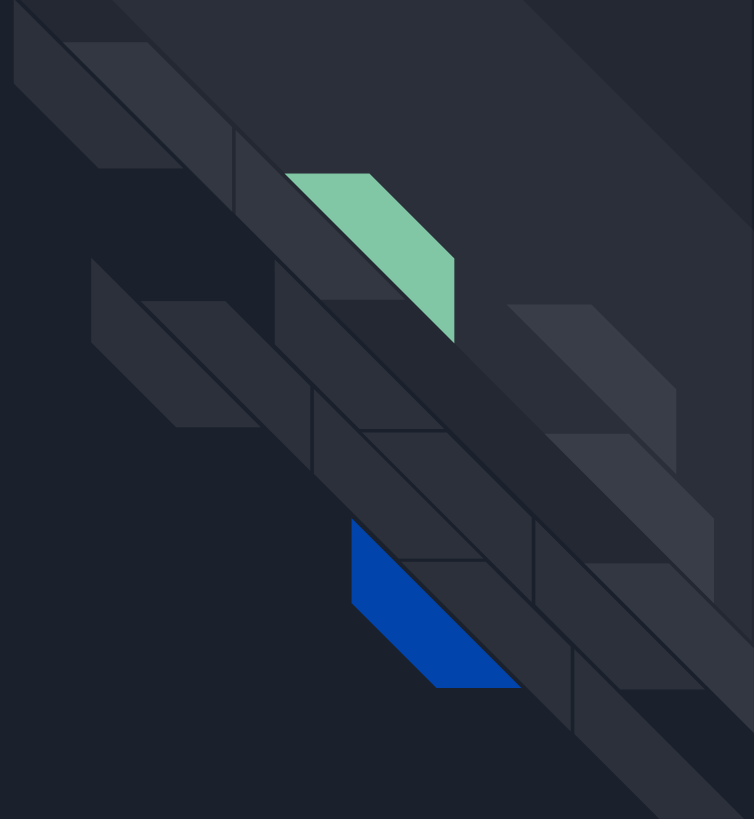
At the bottom of the editor, the status bar shows: Permissions: RW End-of-lines: LF encoding: ASCII Line: 7 Colu 21 Memory: 65 %



# What have we learned so far?

- You can define functions in *Python*
- In *Spyder*, you enter functions in the “left box”
- You can try out functions in the “right box”
- You need to deal with errors to get things to work
- You can comment out functions temporarily

“Strings”





# Strings are fun!

- Examples of strings: "hello" "I love Code Afrique" "fdalepru"
- A string is enclosed in quotation marks ("")
- You can glue together two strings using +

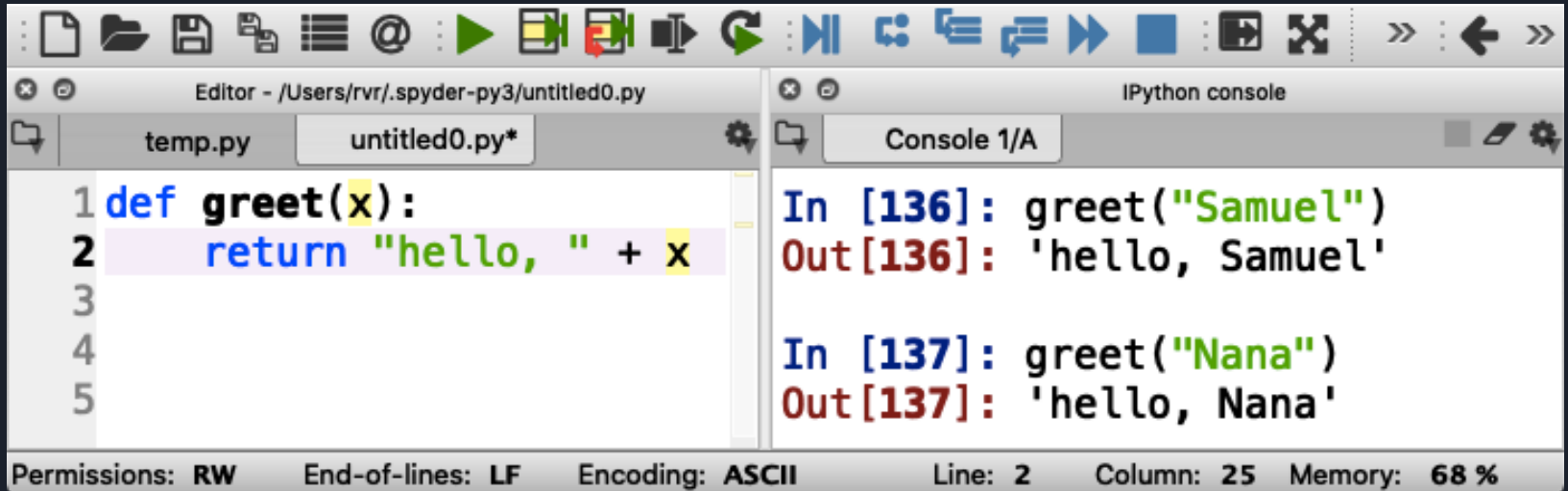
`"hello, " + "Jon"      is "hello, Jon"`

`"abcd" + "efgh"      is "abcdefghj"`

`"efgh" + "abcd"      is "efghabcd"`

# Functions with strings

Write a function `greet(x)` that returns the string "hello, x"



The screenshot shows a Python IDE with two main windows. The left window is the code editor, titled 'Editor - /Users/rvr/.spyder-py3/untitled0.py', showing a Python function definition in a file named 'untitled0.py\*'. The code is as follows:

```
1 def greet(x):
2     return "hello, " + x
3
4
5
```

The right window is the IPython console, titled 'IPython console', showing the execution of the function with two test cases:

```
In [136]: greet("Samuel")
Out[136]: 'hello, Samuel'

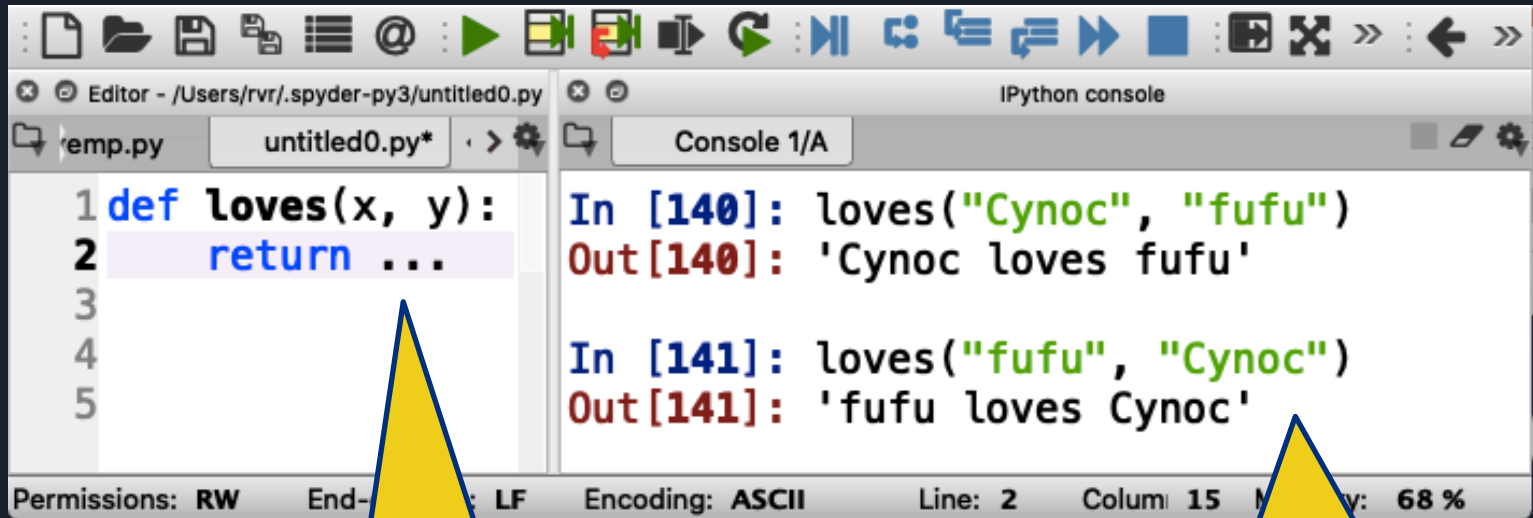
In [137]: greet("Nana")
Out[137]: 'hello, Nana'
```

At the bottom of the IDE, a status bar displays the following information: Permissions: RW, End-of-lines: LF, Encoding: ASCII, Line: 2, Column: 25, Memory: 68 %.



Now you try it (*remember, take turns*)

Write a function `loves(x, y)` that returns a string "`x loves y`"



The screenshot shows a code editor window with a file named `untitled0.py*` containing the following Python code:

```
1 def loves(x, y):  
2     return ...  
3  
4  
5
```

To the right, the IPython console shows the function being tested with two different arguments:

```
In [140]: loves("Cynoc", "fufu")  
Out[140]: 'Cynoc loves fufu'  
  
In [141]: loves("fufu", "Cynoc")  
Out[141]: 'fufu loves Cynoc'
```

The status bar at the bottom of the editor indicates: Permissions: RW, End-: LF, Encoding: ASCII, Line: 2, Column: 15, Memory: 68 %.

*dot dot dot*  
Write your code here

Be sure to test it here  
with different examples  
(and take turns!)

# String indexing

- A position in a string is called an *index*
- Indexes start counting at 0

Shoshana

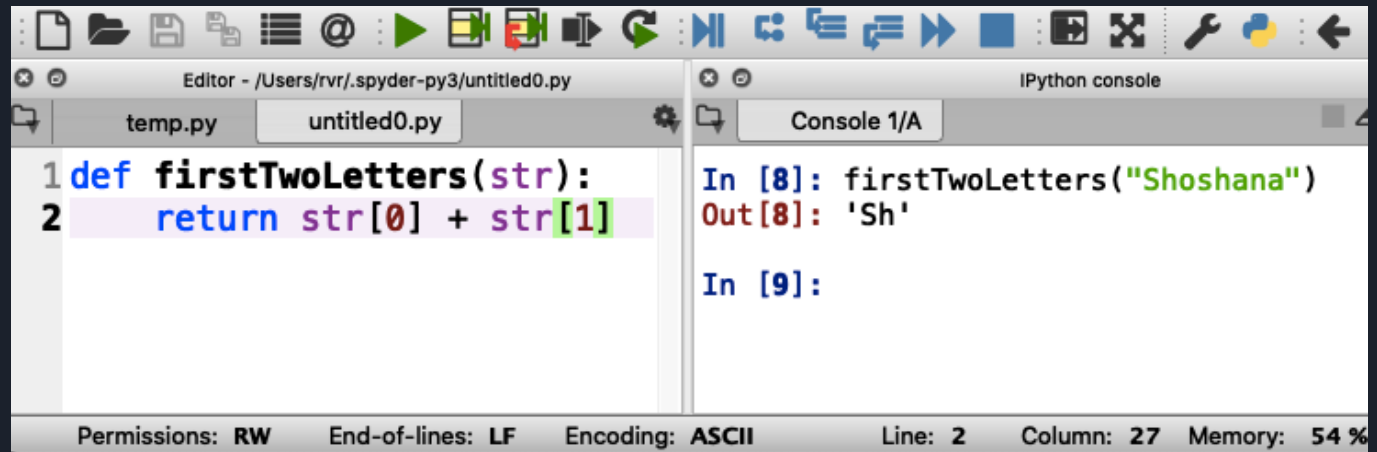
01234567



The index of 'o' is 2

# String indexing

- Write a function `firstTwoLetters(str)` that returns the first two letters in the input string `str`
- `str[x]` evaluates to the letter at index `x`



The screenshot shows a Python IDE with two main panels. The left panel is the code editor, displaying a Python function definition in `untitled0.py`:

```
1 def firstTwoLetters(str):  
2     return str[0] + str[1]
```

The right panel is the IPython console, showing the execution of the function with the string "Shoshana":

```
In [8]: firstTwoLetters("Shoshana")  
Out[8]: 'Sh'
```

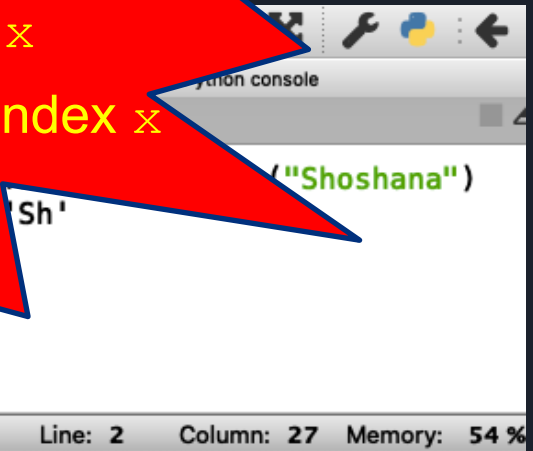
Below the console, the status bar indicates: Permissions: RW, End-of-lines: LF, Encoding: ASCII, Line: 2, Column: 27, Memory: 54 %.

# String indexing

- Write a function `f` that returns the first letter of a string `s`.

## Do not confuse:

- `f(x)` - call function `f` with input parameter `x`
- `s[x]` - get the letter at index `x` from string `s`





# Quiz



If a string has 10 letters, what is the index of the last letter?

- A) 9
- B) 10
- C) 11



## Length of a string

Built-in function `len(str)` returns the length of string `str`

### Examples:

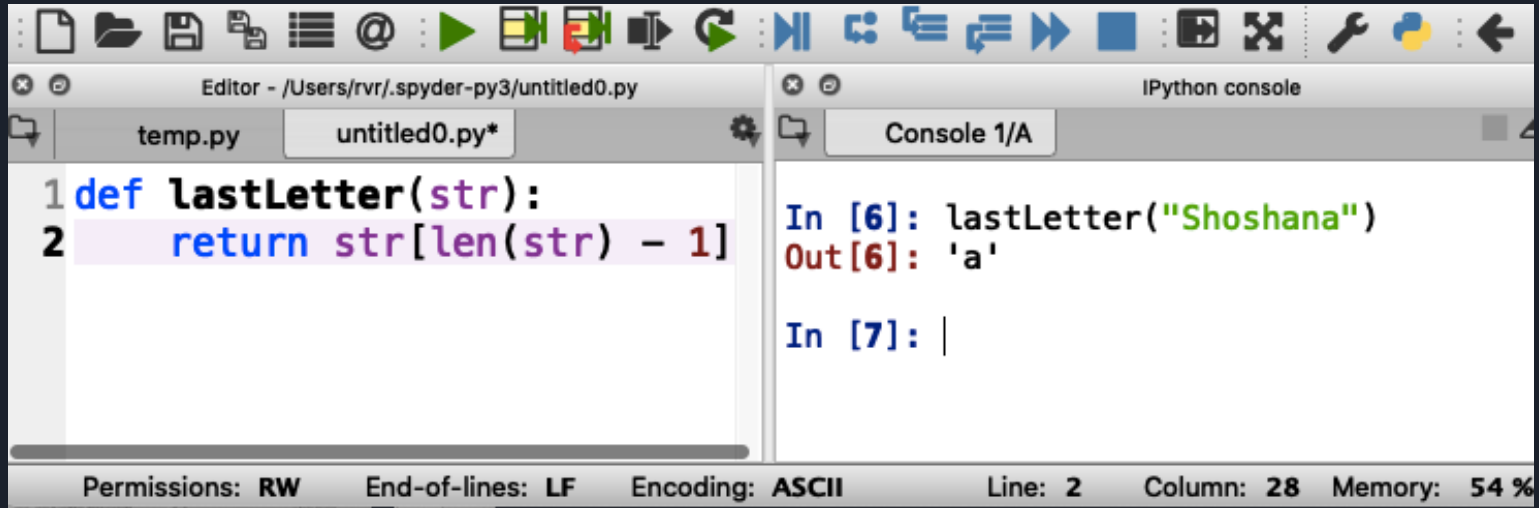
`len("hello")` is 5

`len("Lesley Greene")` is 13

`len("")` is 0

## Function using `len(str)`

- Write a function `lastLetter(str)` that returns the last letter in the input string `str`



The screenshot shows a Python IDE with two main panels. The left panel is a code editor showing a Python function definition:

```
1 def lastLetter(str):  
2     return str[len(str) - 1]
```

The right panel is an IPython console showing the function being called with the string "Shoshana":

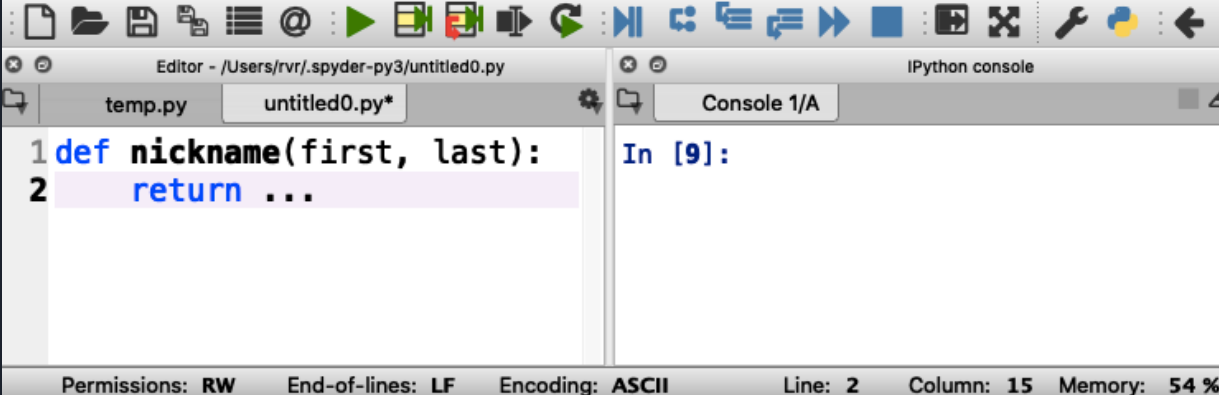
```
In [6]: lastLetter("Shoshana")  
Out[6]: 'a'  
  
In [7]: |
```

At the bottom of the IDE, a status bar displays the following information: Permissions: RW, End-of-lines: LF, Encoding: ASCII, Line: 2, Column: 28, Memory: 54 %.

# What is your Code Afrique nickname?

Write a function `nickname(first, last)` that consists of the first three letters of your given name (first name) followed by the last three letters of your surname (last name), followed by "ca" (for Code Afrique)

For example: `nickname("Hakim", "Weatherspoon")` is "Hakoonca"



The screenshot shows a code editor window with two panes. The left pane is titled "Editor - /Users/rvr/.spyder-py3/untitled0.py" and contains the following Python code:

```
1 def nickname(first, last):  
2     return ...
```

The right pane is titled "IPython console" and contains the prompt:

```
In [9]:
```

The status bar at the bottom of the editor shows: "Permissions: RW End-of-lines: LF Encoding: ASCII Line: 2 Column: 15 Memory: 54 %"





# What have we learned about strings?

- A string is a list of letters, enclosed by quotes
- We can glue two strings together with `+`
- We can get the length of string `s` with the function `len(s)`
- Letters in a string are indexed from `0`
- To get the letter of string `s` at index `x`, use `s[x]` (square brackets!)



# Lists



## Examples of lists

- [ 1, -5, 3 ]
- [ "dogs", "cats" ]
- []
- [4]

a list of numbers

a list of strings


an empty list

a list with one number



## Examples of lists

- [ 1, -5, 3 ]
- [ "dogs", "cats" ]
- []
- [4]



Lists use "square brackets" []

a list of numbers

a list of strings

an empty list

a list with one number

## Examples of lists

Lists use “square brackets” []

- [ 1, -5, 3 ] a list of numbers
- [ “dogs”, “cats” ] a list of strings
- [] an empty list
- [4] a list with one number

**Fun fact: a string is a list of letters, and everything you can do with strings you can also do with other lists**

For example:

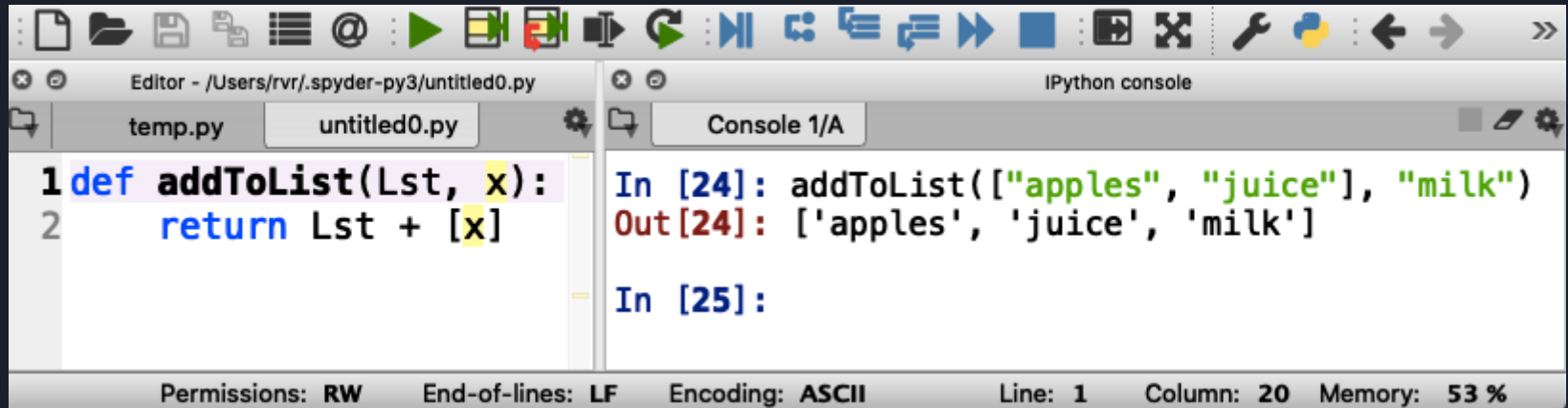
```
len( [ “dogs”, “cats”, “mice” ] ) is 3
```

```
[ “dogs” ] + [ “cats” ] is [ “dogs”, “cats” ]
```

# Example function with lists

Write a function `addToList(Lst, x)` that returns a new list consisting of `Lst` and `x`

For example `addToList(["apples", "juice"], "milk")` becomes `["apples", "juice", "milk"]`



The screenshot shows a Python IDE with two panes. The left pane is a code editor showing a function definition in `temp.py`:

```
1 def addToList(Lst, x):  
2     return Lst + [x]
```

The right pane is the IPython console, showing the execution of the function:

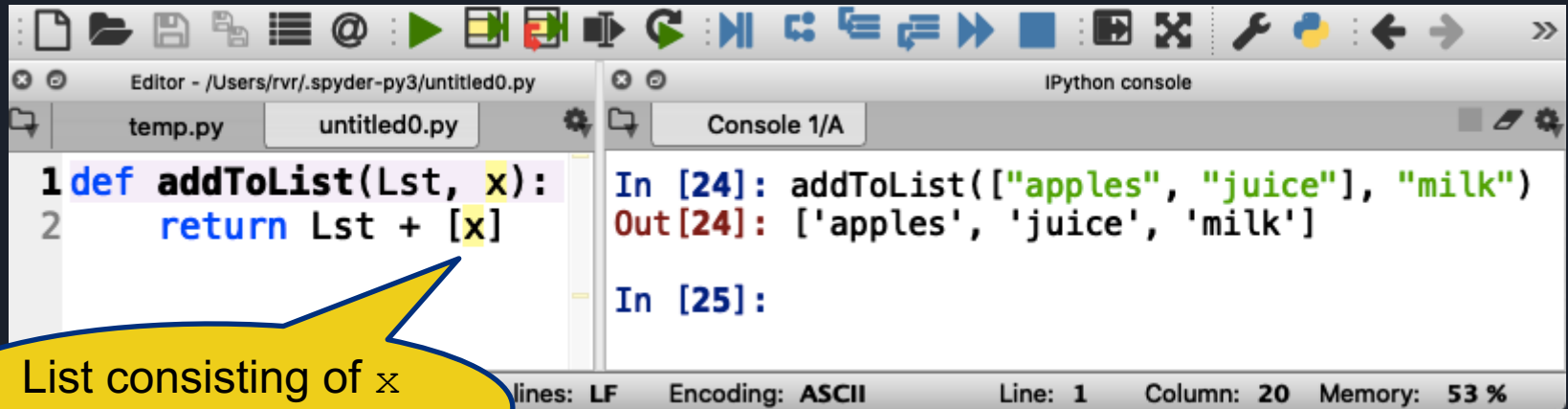
```
In [24]: addToList(["apples", "juice"], "milk")  
Out[24]: ['apples', 'juice', 'milk']  
  
In [25]:
```

The status bar at the bottom indicates: Permissions: RW, End-of-lines: LF, Encoding: ASCII, Line: 1, Column: 20, Memory: 53 %.

## Example function with lists

Write a function `addToList(Lst, x)` that returns a new list consisting of `Lst` and `x`

For example `addToList(["apples", "juice"], "milk")` becomes `["apples", "juice", "milk"]`



The screenshot shows a Python IDE with two panes. The left pane is a code editor showing a function definition:

```
1 def addToList(Lst, x):  
2     return Lst + [x]
```

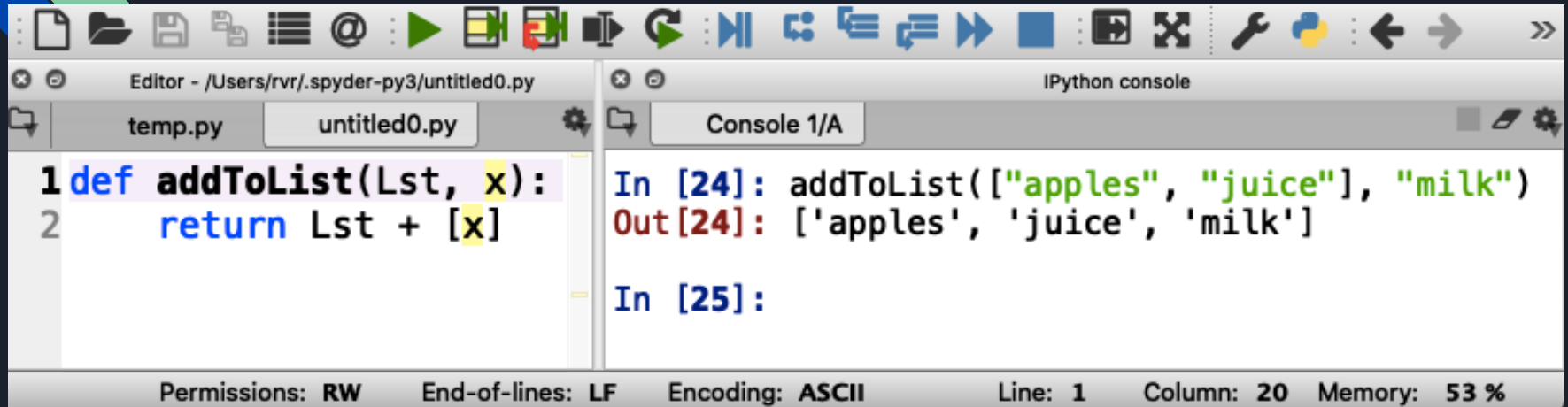
The right pane is the IPython console showing the function being called and the output:

```
In [24]: addToList(["apples", "juice"], "milk")  
Out[24]: ['apples', 'juice', 'milk']  
  
In [25]:
```

The status bar at the bottom indicates the current line and column: "Line: 1 Column: 20 Memory: 53 %".

List consisting of `x`  
(just one element)

Try it out (and take turns!)



The image shows a screenshot of the Spyder Python IDE interface. The top part of the window contains a toolbar with various icons for file operations, execution, and navigation. Below the toolbar, there are two main panels. The left panel is the code editor, showing a Python function definition in a file named 'untitled0.py'. The function is named 'addToList' and takes two arguments: 'Lst' and 'x'. The function body consists of a single line: 'return Lst + [x]'. The right panel is the IPython console, showing the execution of the function. The input is 'addToList(["apples", "juice"], "milk")' and the output is '['apples', 'juice', 'milk']'. Below the console, there is a status bar with the following information: Permissions: RW, End-of-lines: LF, Encoding: ASCII, Line: 1, Column: 20, Memory: 53 %.

```
1 def addToList(Lst, x):  
2     return Lst + [x]
```

In [24]: addToList(["apples", "juice"], "milk")  
Out[24]: ['apples', 'juice', 'milk']  
  
In [25]:

Permissions: RW End-of-lines: LF Encoding: ASCII Line: 1 Column: 20 Memory: 53 %





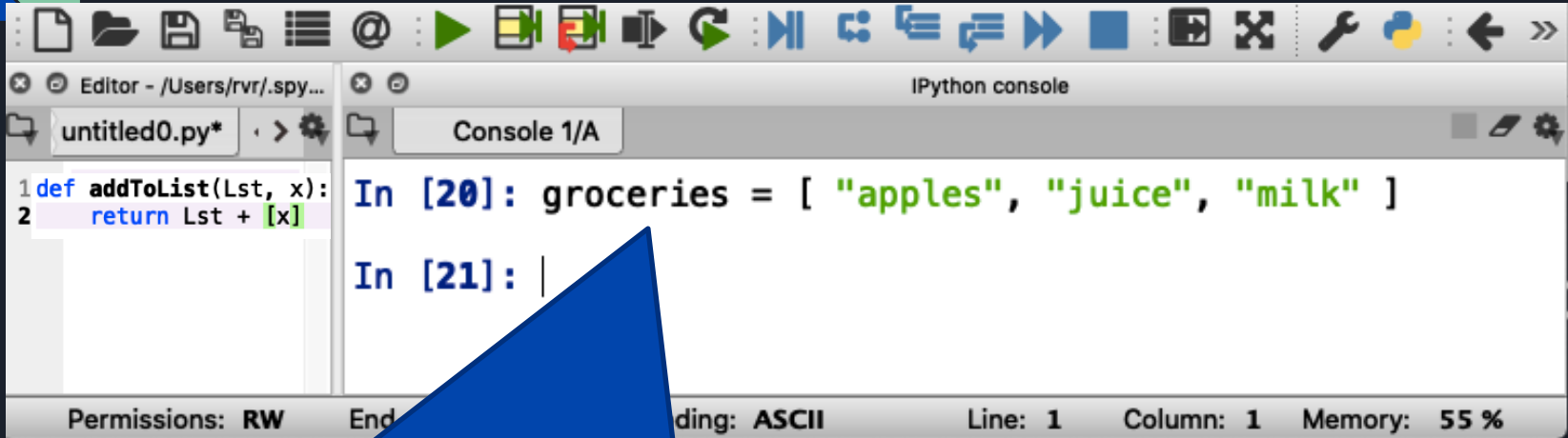
# What have we learned about lists?

- You can put anything in a list
- They are much like strings
- We can glue two lists together with `+`
- We can get the number of elements in list `Lst` using `len(Lst)`
- Elements of a list are indexed from `0`
- To get the element of string `Lst` at index `x`, use `Lst[x]`



# Variables

# How to keep a shopping list?



The screenshot shows a code editor window on the left with a file named 'untitled0.py\*' containing the following Python code:

```
1 def addToList(Lst, x):  
2     return Lst + [x]
```

On the right is an IPython console window titled 'Console 1/A' showing the execution of the code:

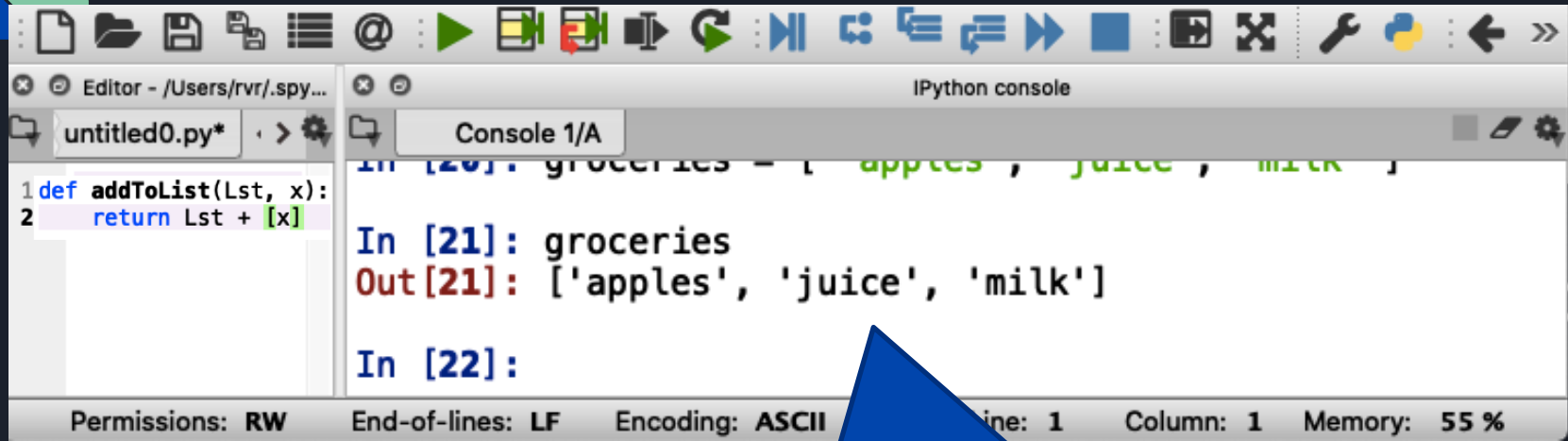
```
In [20]: groceries = [ "apples", "juice", "milk" ]  
  
In [21]: |
```

At the bottom of the console window, the status bar displays: 'Permissions: RW', 'End', 'Encoding: ASCII', 'Line: 1', 'Column: 1', and 'Memory: 55 %'.

groceries is a “variable”

**Variables remember values so you can use them later**

# How to keep a shopping list?



The screenshot shows a Python IDE with two panes. The left pane is an editor showing a function definition in a file named 'untitled0.py\*':

```
1 def addToList(Lst, x):  
2     return Lst + [x]
```

The right pane is an IPython console showing the execution of the function. The first input is:

```
In [20]: groceries = ['apples', 'juice', 'milk']
```

The output is:

```
Out[21]: ['apples', 'juice', 'milk']
```

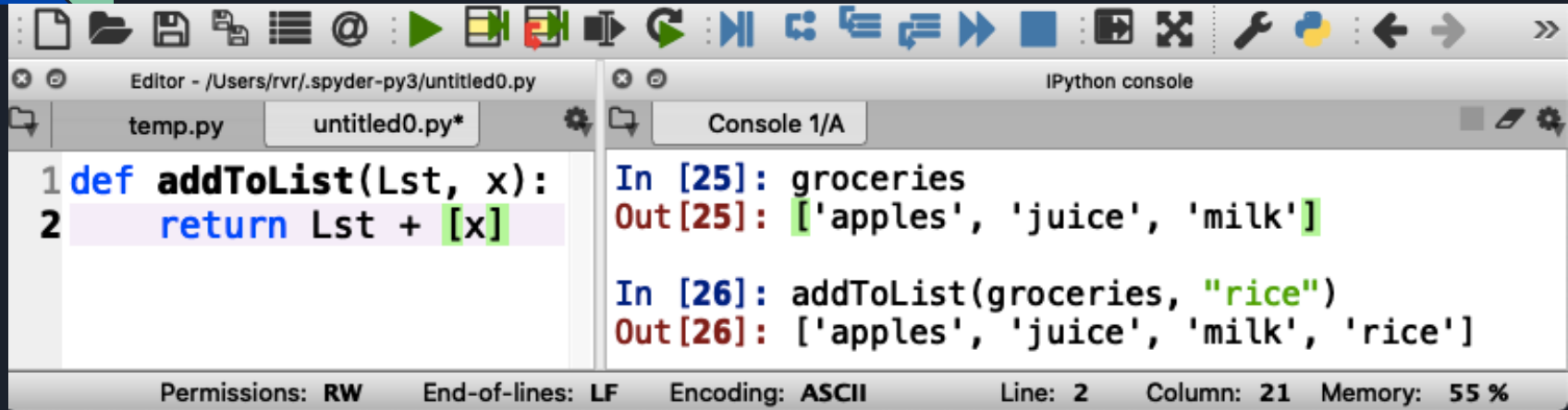
The second input is:

```
In [22]:
```

The status bar at the bottom of the console shows: Permissions: RW, End-of-lines: LF, Encoding: ASCII, Line: 1, Column: 1, Memory: 55 %.

Python can recall the value of `groceries`

# Example: adding "rice" to the shopping list



The screenshot shows the Spyder Python IDE interface. The left pane is the code editor, and the right pane is the IPython console. The code editor shows a function definition for `addToList`. The console shows the execution of the function with a list of groceries and the addition of 'rice'.

```
1 def addToList(Lst, x):  
2     return Lst + [x]
```

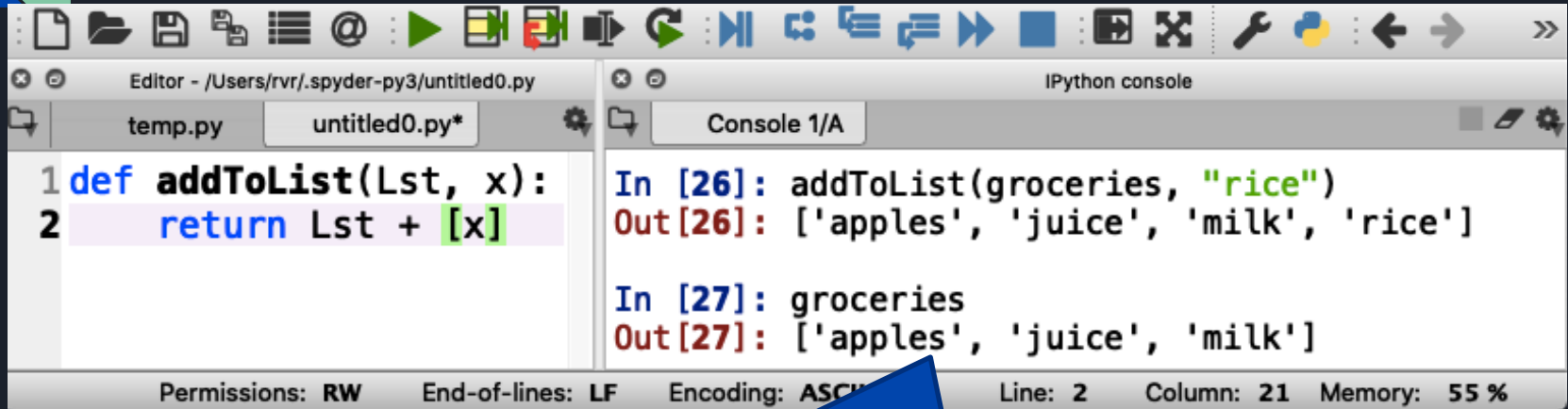
In [25]: groceries  
Out[25]: ['apples', 'juice', 'milk']

In [26]: addToList(groceries, "rice")  
Out[26]: ['apples', 'juice', 'milk', 'rice']

Permissions: RW End-of-lines: LF Encoding: ASCII Line: 2 Column: 21 Memory: 55 %

Does it work?

# Example: adding "rice" to the shopping list



The screenshot shows a Python IDE with two panes. The left pane is a code editor showing a function definition:

```
1 def addToList(Lst, x):  
2     return Lst + [x]
```

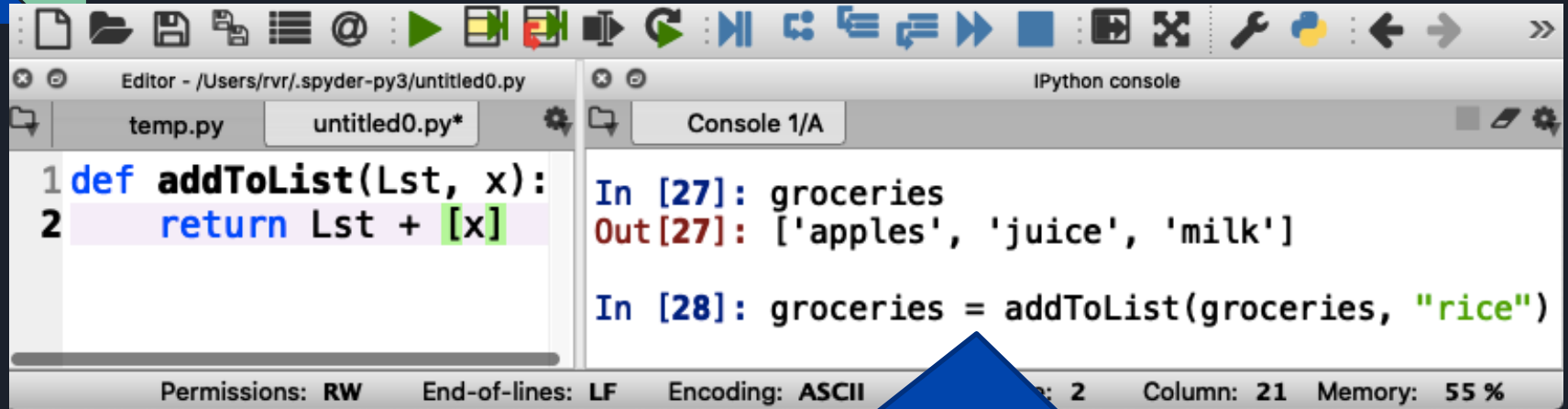
The right pane is the IPython console showing the execution of the function:

```
In [26]: addToList(groceries, "rice")  
Out[26]: ['apples', 'juice', 'milk', 'rice']  
  
In [27]: groceries  
Out[27]: ['apples', 'juice', 'milk']
```

At the bottom of the console, the status bar shows: Permissions: RW, End-of-lines: LF, Encoding: ASCII, Line: 2, Column: 21, Memory: 55 %.

The value of `groceries` has *not* changed...  
(because we didn't tell Python to do that)

# Example: adding "rice" to the shopping list



The screenshot shows a Python IDE with two main windows: an editor and an IPython console. The editor window displays a function definition in a file named `temp.py`:

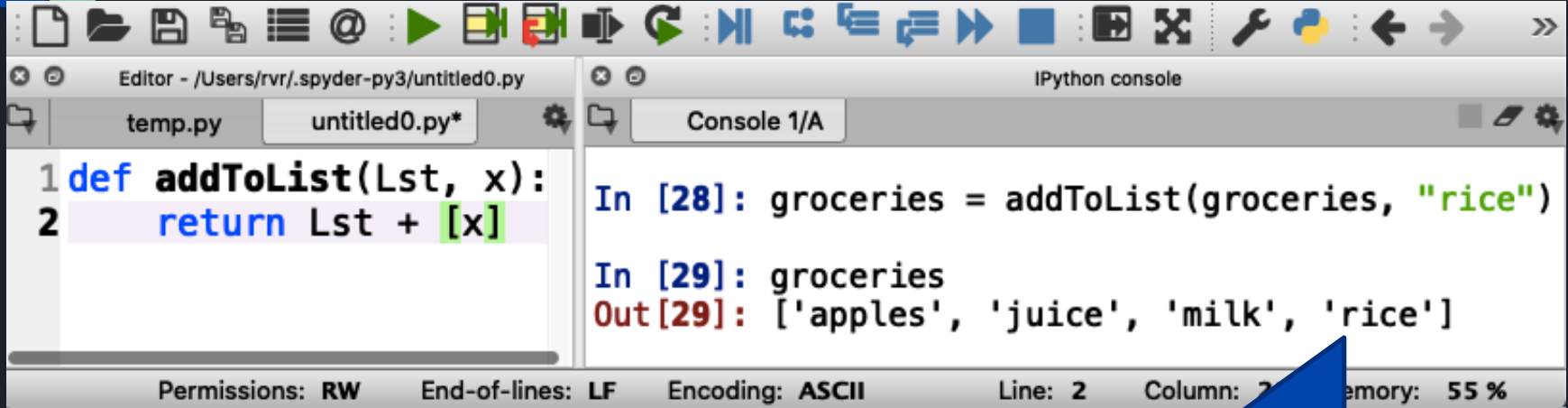
```
1 def addToList(Lst, x):  
2     return Lst + [x]
```

The IPython console window shows the execution of the function. The first input is `groceries`, which returns `['apples', 'juice', 'milk']`. The second input is `groceries = addToList(groceries, "rice")`, which updates the `groceries` variable.

At the bottom of the IDE, the status bar shows: Permissions: RW, End-of-lines: LF, Encoding: ASCII, Line: 2, Column: 21, Memory: 55 %.

Here we tell Python to store a new value for `groceries`

# Example: adding "rice" to the shopping list



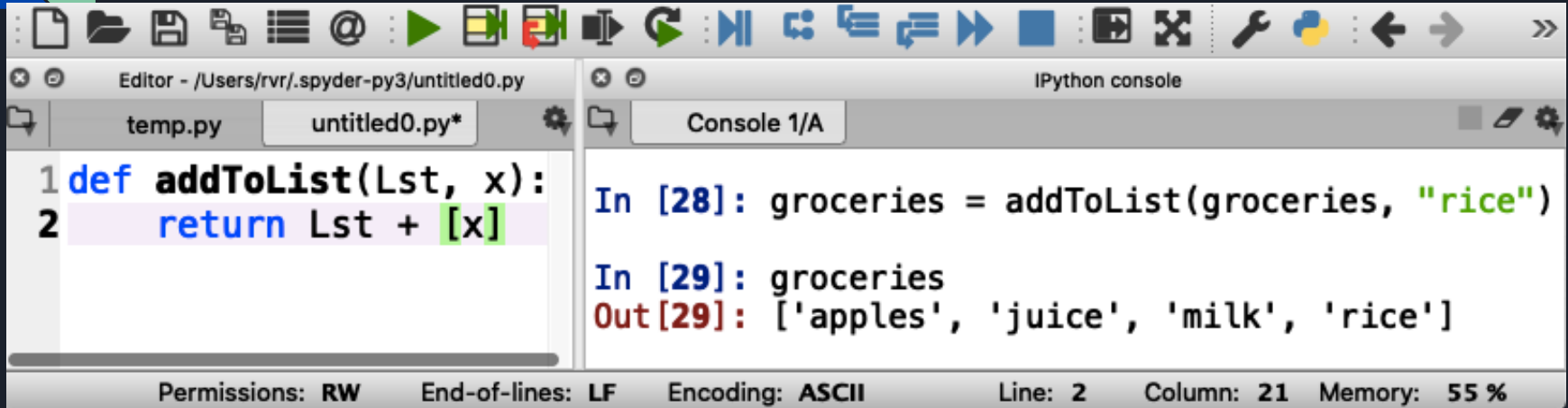
The screenshot shows a Python IDE with two main panes. The left pane is a code editor showing a function definition in a file named 'untitled0.py\*'. The function is named 'addToList' and takes two arguments: 'Lst' and 'x'. The function body consists of a single line: 'return Lst + [x]'. The right pane is an IPython console showing the execution of the function. The first input is 'groceries = addToList(groceries, "rice")'. The second input is 'groceries', which results in the output: ['apples', 'juice', 'milk', 'rice']. The status bar at the bottom of the IDE shows 'Permissions: RW', 'End-of-lines: LF', 'Encoding: ASCII', 'Line: 2', 'Column: 2', and 'Memory: 55 %'.

```
1 def addToList(Lst, x):  
2     return Lst + [x]  
  
In [28]: groceries = addToList(groceries, "rice")  
  
In [29]: groceries  
Out[29]: ['apples', 'juice', 'milk', 'rice']
```

Now "rice" is in groceries



# Example: adding "rice" to the shopping list



The screenshot shows the Spyder Python IDE interface. The left pane is the code editor, and the right pane is the IPython console. The code editor shows a function definition for `addToList`. The console shows the function being called with `groceries` and `"rice"`, and the resulting list output.

```
1 def addToList(Lst, x):  
2     return Lst + [x]
```

```
In [28]: groceries = addToList(groceries, "rice")  
  
In [29]: groceries  
Out[29]: ['apples', 'juice', 'milk', 'rice']
```

Permissions: RW End-of-lines: LF Encoding: ASCII Line: 2 Column: 21 Memory: 55 %

Now practice adding some more items to `groceries`



# What have we learned about variables?

- Each variable has a name
- You can store any value in a variable
- The notation for this in Python is `x = value`
- You can change the value of a variable the same way

# Loops






## Doing something with each element of a list

Write a function `product(Lst)` that returns the product of all elements in a list of numbers

- *How would you do this writing on a piece of paper?*



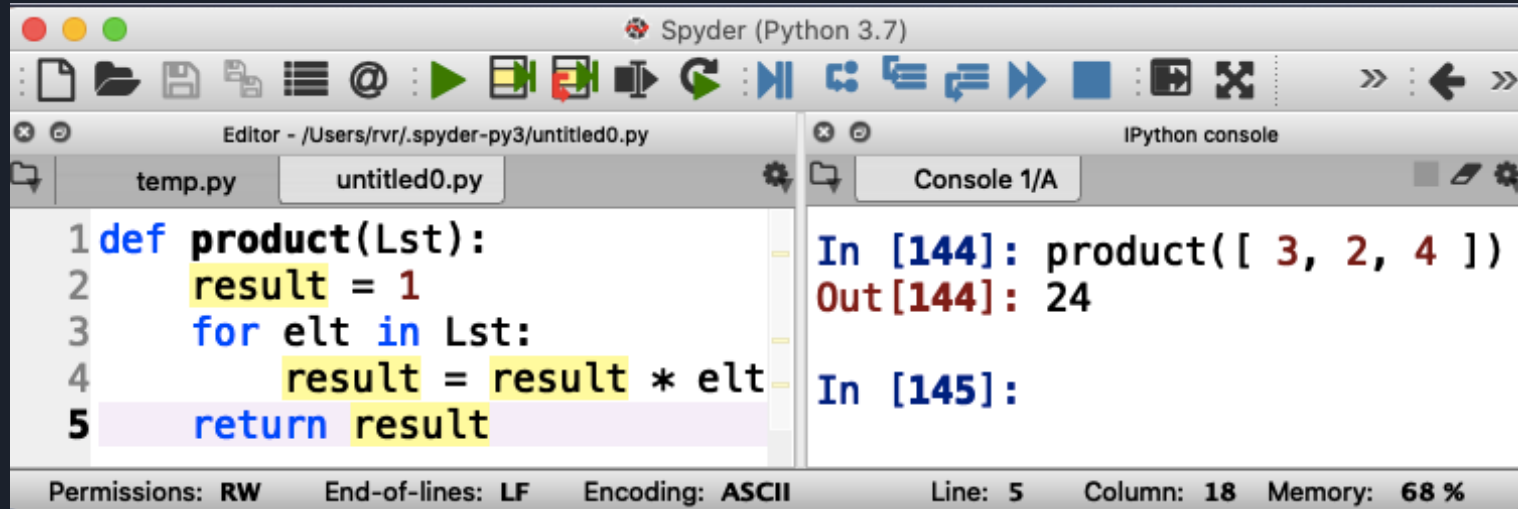
## Doing something with each element of a list

Write a function `product(Lst)` that returns the product of all elements in a list of numbers

- *How would you do this writing on a piece of paper?*
  1. Start with the first number
  2. Multiply with the next number
  3. Repeat until done with all numbers

**This is called a “loop”**

# Loops in Python



The screenshot shows the Spyder Python IDE interface. The editor window displays a Python function named `product` that iterates over a list and calculates the product of its elements. The IPython console shows the function being called with the list `[3, 2, 4]` and returning the value `24`.

```
1 def product(Lst):
2     result = 1
3     for elt in Lst:
4         result = result * elt
5     return result
```

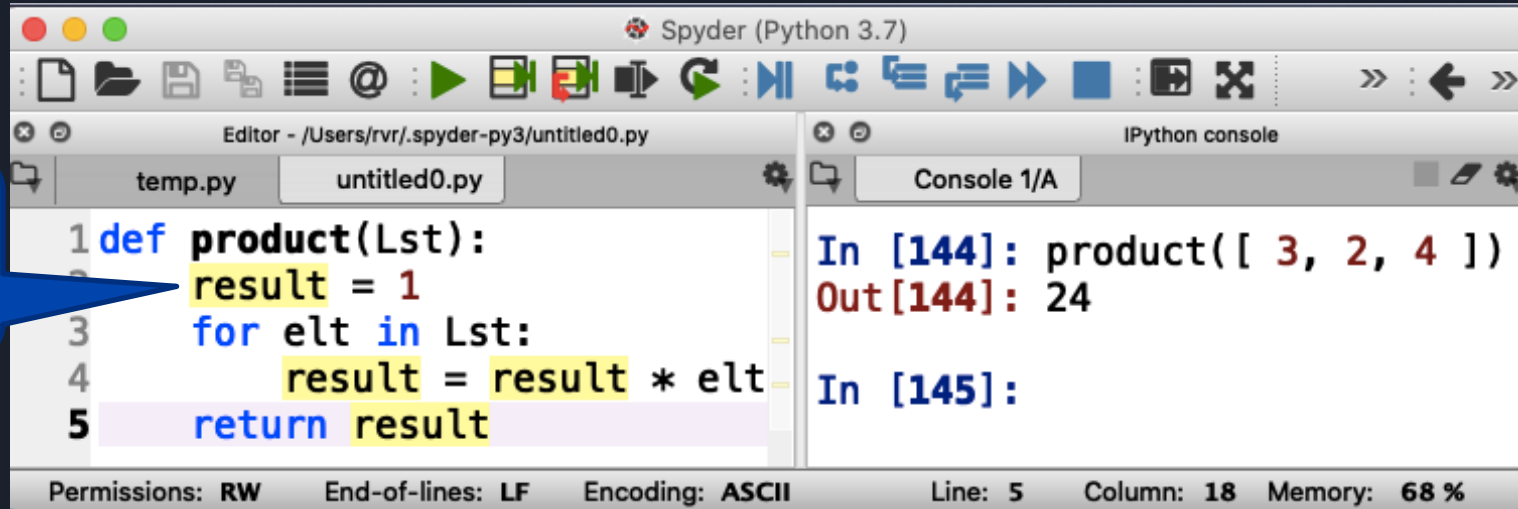
In [144]: `product([ 3, 2, 4 ])`  
Out[144]: 24

In [145]:

Permissions: RW End-of-lines: LF Encoding: ASCII Line: 5 Column: 18 Memory: 68 %

# Loops in Python

result is a  
variable



The screenshot shows the Spyder Python IDE interface. The editor window displays a Python function named `product` that takes a list `Lst` as input and returns the product of its elements. The function is defined as follows:

```
1 def product(Lst):  
2     result = 1  
3     for elt in Lst:  
4         result = result * elt  
5     return result
```

The IPython console on the right shows the execution of the function with the input `[3, 2, 4]`, resulting in the output `24`.

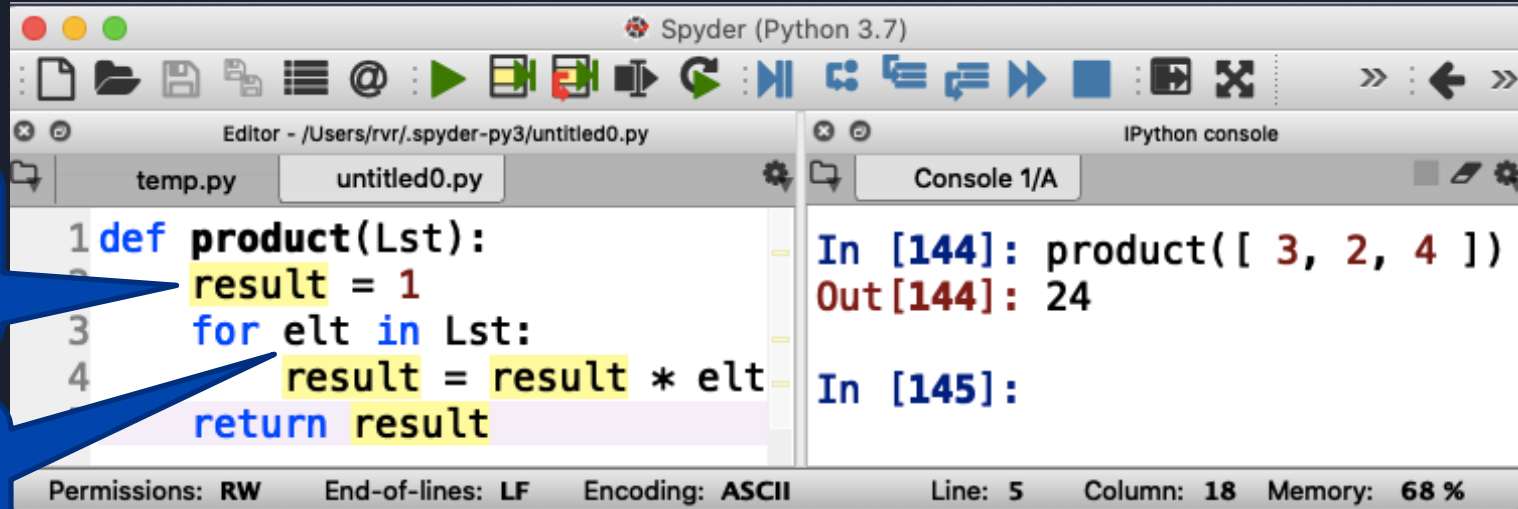
```
In [144]: product([ 3, 2, 4 ])  
Out[144]: 24  
  
In [145]:
```

The status bar at the bottom indicates the file permissions are `RW`, end-of-lines are `LF`, encoding is `ASCII`, and the current cursor position is `Line: 5 Column: 18` with `Memory: 68 %` usage.

# Loops in Python

result is a  
variable

elt is an  
element in  
Lst



The screenshot shows the Spyder Python IDE interface. The editor window displays a Python function named `product` that takes a list `Lst` as input and returns the product of its elements. The function is defined as follows:

```
1 def product(Lst):  
2     result = 1  
3     for elt in Lst:  
4         result = result * elt  
5     return result
```

The IPython console on the right shows the execution of the function with the input `[3, 2, 4]`, resulting in the output `24`.

```
In [144]: product([ 3, 2, 4 ])  
Out[144]: 24  
  
In [145]:
```

The status bar at the bottom indicates the file permissions (RW), end-of-lines (LF), encoding (ASCII), and the current cursor position (Line: 5, Column: 18) along with memory usage (68%).

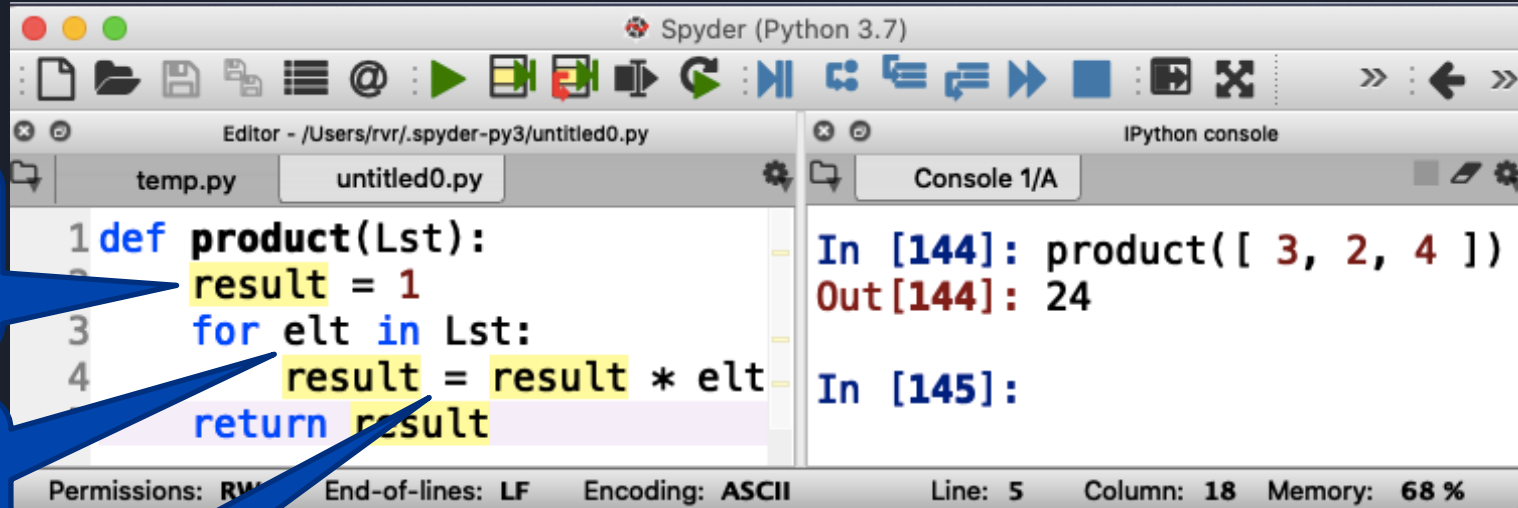


# Loops in Python

result is a variable

elt is an element in Lst

result is updated for each element



```
def product(Lst):  
    result = 1  
    for elt in Lst:  
        result = result * elt  
    return result
```

```
In [144]: product([ 3, 2, 4 ])  
Out[144]: 24  
  
In [145]:
```

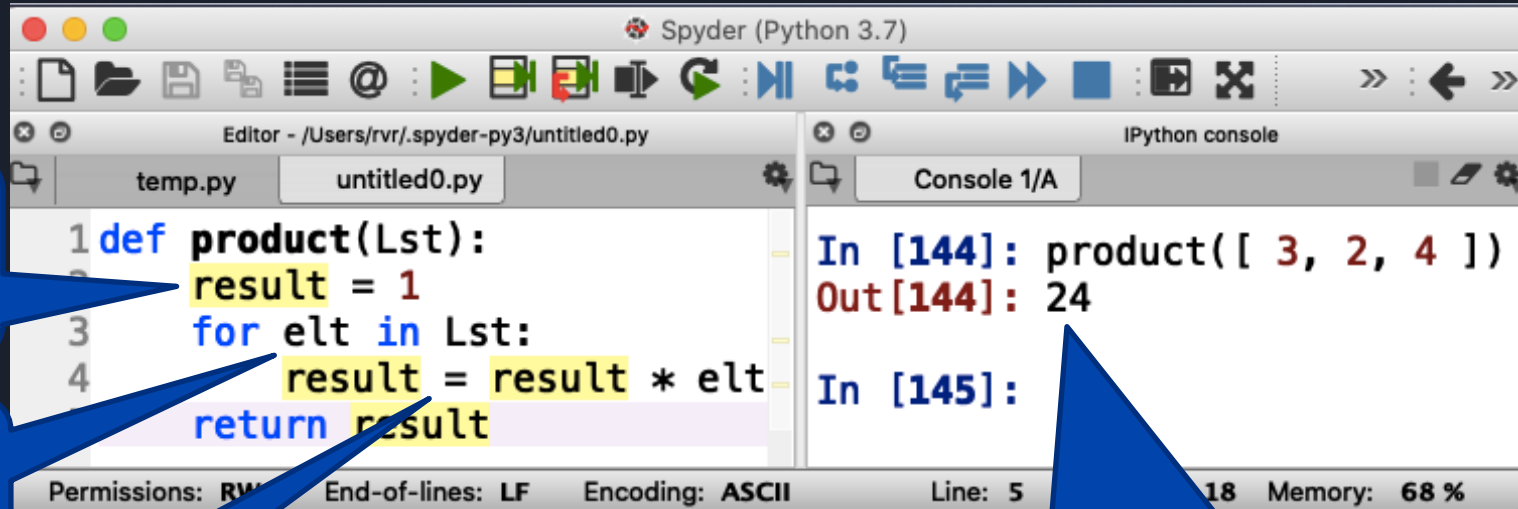
Permissions: RW End-of-lines: LF Encoding: ASCII Line: 5 Column: 18 Memory: 68 %

# Loops in Python

result is a variable

elt is an element in Lst

result is updated for each element



```
def product(Lst):  
    result = 1  
    for elt in Lst:  
        result = result * elt  
    return result
```

```
In [144]: product([ 3, 2, 4 ])  
Out[144]: 24  
  
In [145]:
```

Here, result is  $1 * 3 * 2 * 4$

# Executing a loop

elt



product([ 3, 2, 4 ])

```
def product(Lst):  
    result = 1  
    for elt in Lst:  
        result = result * elt  
    return result
```

elt	old result	new result
3	1	???

## Executing a loop

```
def product(Lst):  
    result = 1  
    for elt in Lst:  
        result = result * elt  
    return result
```

elt



product([ 3, 2, 4 ])

elt	old result	new result
3	1	3

# Executing a loop

elt



product([ 3, 2, 4 ])

```
def product(Lst):  
    result = 1  
    for elt in Lst:  
        result = result * elt  
    return result
```

elt	old result	new result
3	1	3
2	3	???

# Executing a loop

```
def product(Lst):  
    result = 1  
    for elt in Lst:  
        result = result * elt  
    return result
```

product([ 3, 2, 4 ])



elt	old result	new result
3	1	3
2	3	6

# Executing a loop

```
def product(Lst):  
    result = 1  
    for elt in Lst:  
        result = result * elt  
    return result
```

elt  
↓  
product([ 3, 2, 4 ])

elt	old result	new result
3	1	3
2	3	6
4	6	???

# Executing a loop

```
def product(Lst):  
    result = 1  
    for elt in Lst:  
        result = result * elt  
    return result
```

elt  
↓  
product([ 3, 2, 4 ])

elt	old result	new result
3	1	3
2	3	6
4	6	24



# Executing a loop

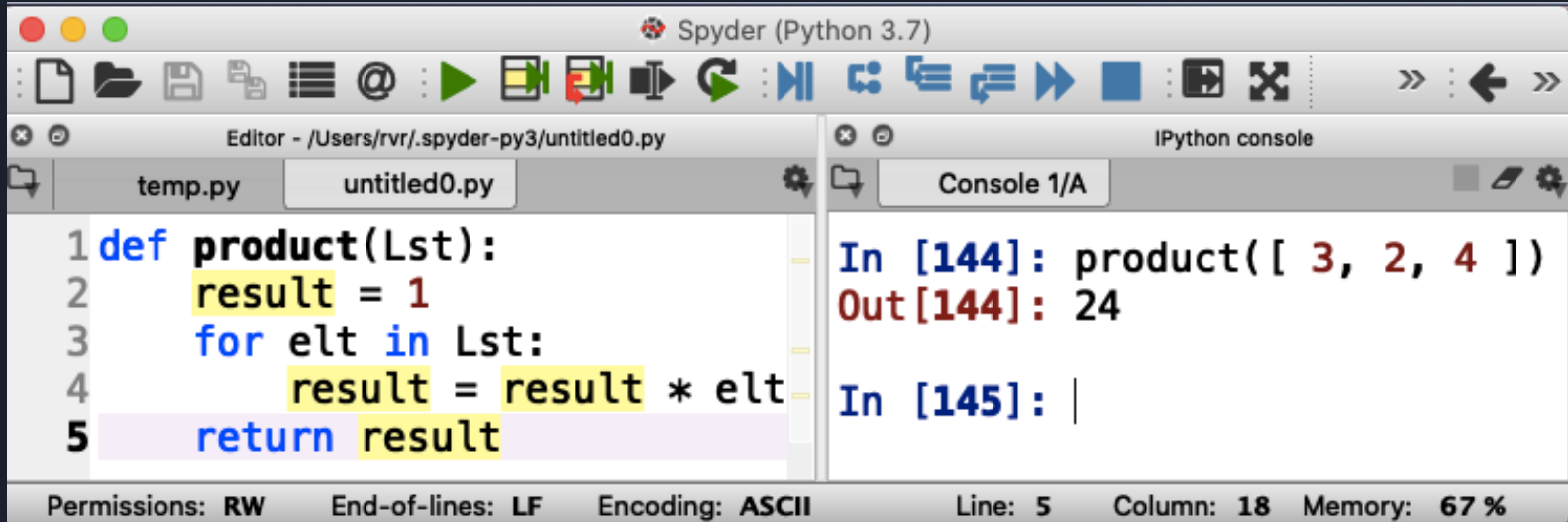
```
product([ 3, 2, 4 ])
```

```
def product(Lst):  
    result = 1  
    for elt in Lst:  
        result = result * elt  
    return result
```

elt	old result	new result
3	1	3
2	3	6
4	6	24

The final result of the function

Write your own: below change `product` to `sum`  
That is, add all the items in the list together



The screenshot shows the Spyder Python IDE interface. The top toolbar contains various icons for file operations and execution. The main window is split into two panes. The left pane is the code editor, showing a Python function named `product` defined in `untitled0.py`. The function takes a list `Lst` as input and returns the product of its elements. The right pane is the IPython console, showing the execution of the function with the input `[3, 2, 4]` and the output `24`. The status bar at the bottom indicates the file permissions, end-of-lines, encoding, and current cursor position.

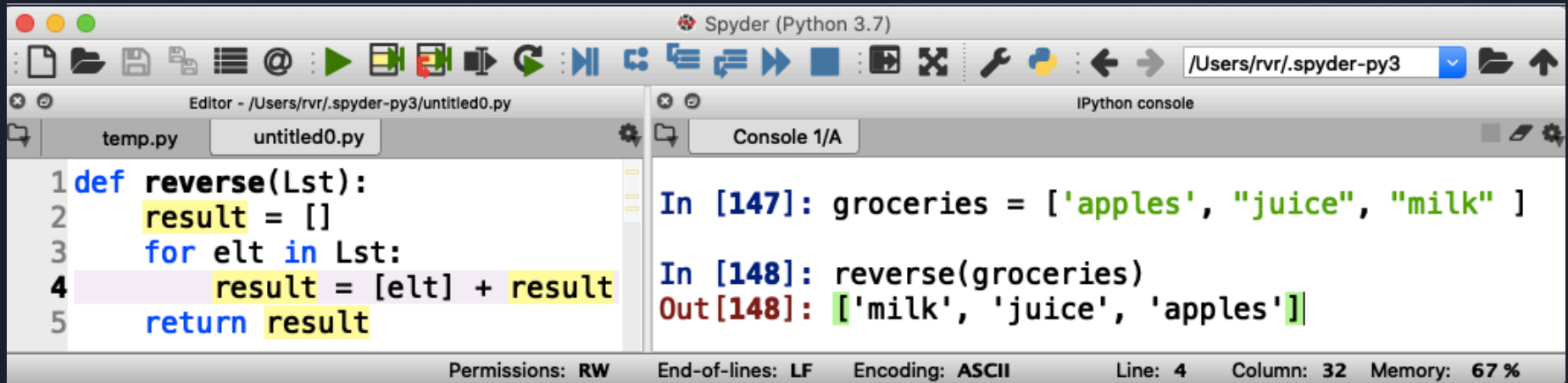
```
1 def product(Lst):
2     result = 1
3     for elt in Lst:
4         result = result * elt
5     return result
```

In [144]: `product([ 3, 2, 4 ])`  
Out[144]: 24

In [145]: |

Permissions: RW End-of-lines: LF Encoding: ASCII Line: 5 Column: 18 Memory: 67 %

Example: write a function `reverse` (Lst) that reverses the elements in a list



The screenshot shows the Spyder Python IDE interface. The top toolbar contains various icons for file operations and execution. The main window is split into two panes: an editor on the left and an IPython console on the right. The editor pane shows a Python script named `untitled0.py` with the following code:

```
1 def reverse(Lst):
2     result = []
3     for elt in Lst:
4         result = [elt] + result
5     return result
```

The IPython console pane shows the execution of the code:

```
In [147]: groceries = ['apples', "juice", "milk" ]
In [148]: reverse(groceries)
Out[148]: ['milk', 'juice', 'apples']
```

The status bar at the bottom of the window displays the following information: Permissions: RW, End-of-lines: LF, Encoding: ASCII, Line: 4, Column: 32, Memory: 67 %.

reverse([ "apples", "juice", "milk" ])



elt

```
def reverse(Lst):  
    result = []  
    for elt in Lst:  
        result = [elt] + result  
    return result
```

elt	old result	new result
"apples"	[]	???

reverse([ "apples", "juice", "milk" ])



elt

```
def reverse(Lst):  
    result = []  
    for elt in Lst:  
        result = [elt] + result  
    return result
```

elt	old result	new result
"apples"	[]	["apples"]

reverse([ "apples", "juice", "milk" ])



elt

```
def reverse(Lst):  
    result = []  
    for elt in Lst:  
        result = [elt] + result  
    return result
```

elt	old result	new result
"apples"	[]	["apples"]
"juice"	["apples"]	???

reverse([ "apples", "juice", "milk" ])



```
def reverse(Lst):  
    result = []  
    for elt in Lst:  
        result = [elt] + result  
    return result
```

elt	old result	new result
"apples"	[]	["apples"]
"juice"	["apples"]	["juice", "apples"]

reverse([ "apples", "juice", "milk" ])

 elt

```
def reverse(Lst):  
    result = []  
    for elt in Lst:  
        result = [elt] + result  
    return result
```

elt	old result	new result
"apples"	[]	["apples"]
"juice"	["apples"]	["juice", "apples"]
"milk"	["apples", "juice"]	???



reverse([ "apples", "juice", "milk" ])



```
def reverse(Lst):  
    result = []  
    for elt in Lst:  
        result = [elt] + result  
    return result
```

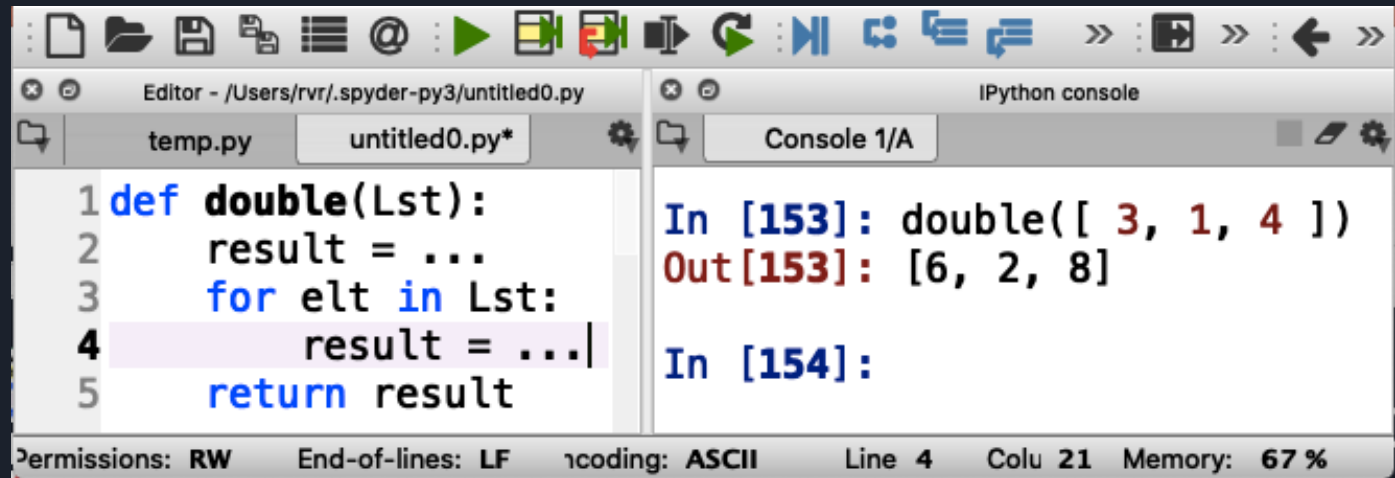
elt	old result	new result
"apples"	[]	["apples"]
"juice"	["apples"]	["juice", "apples"]
"milk"	["apples", "juice"]	["milk", "juice", "apples"]

# More practice with loops

(remember: everybody gets a turn!)

Write a function `double(Lst)` that returns a list consisting of all the numbers in `Lst`, but then doubled

- For example, `double([ 3, 1, 4 ])` should return `[ 6, 2, 8 ]`



The screenshot shows a Python IDE with two main panes. The left pane is a code editor showing a function definition for `double(Lst)`. The right pane is an IPython console showing the function being called with `[3, 1, 4]` and returning `[6, 2, 8]`. The status bar at the bottom indicates file permissions, line endings, encoding, and memory usage.

```
def double(Lst):
    result = ...
    for elt in Lst:
        result = ...
    return result
```

```
In [153]: double([ 3, 1, 4 ])
Out[153]: [6, 2, 8]
```

```
In [154]:
```

Permissions: RW End-of-lines: LF encoding: ASCII Line 4 Colu 21 Memory: 67 %



## Working with ranges

Write a function `squares(start, end)` that returns a list of all the squares of the numbers `start` (inclusive) through `end` (exclusive)

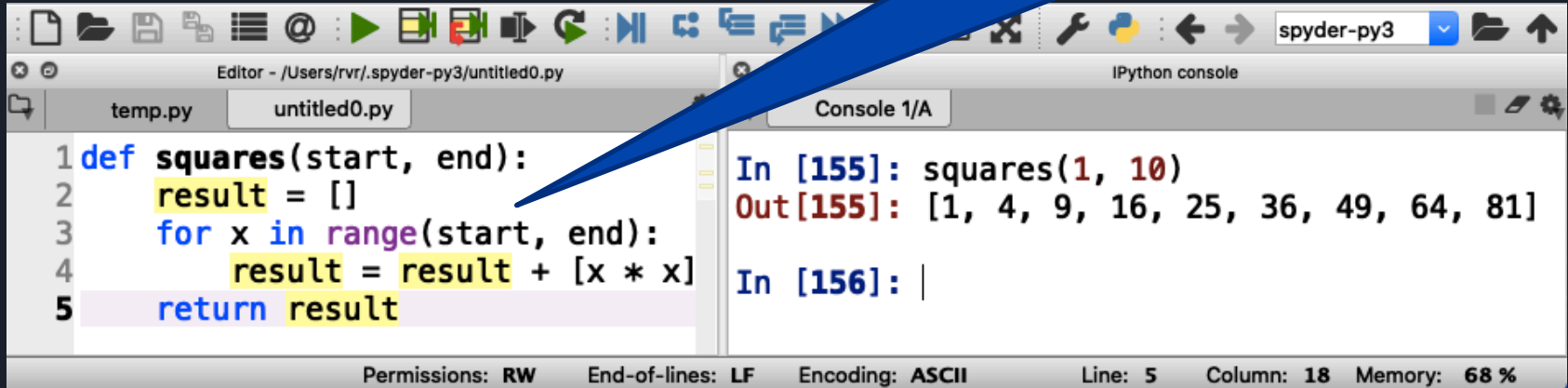
(in Computer Science, “inclusive” means that the value is included, and “exclusive” means that the value is excluded)

For example, `squares(1, 10)` should return

```
[ 1, 4, 9, 16, 25, 36, 49, 64, 81 ]
```

# Working with ranges

`range(start, end)` generates the integers from `start` (inclusive) to `end` (exclusive)



The screenshot shows the Spyder Python IDE interface. The left pane is the code editor, and the right pane is the IPython console. A blue callout bubble points from the text above to the `range(start, end)` function call in the code.

```
1 def squares(start, end):
2     result = []
3     for x in range(start, end):
4         result = result + [x * x]
5     return result
```

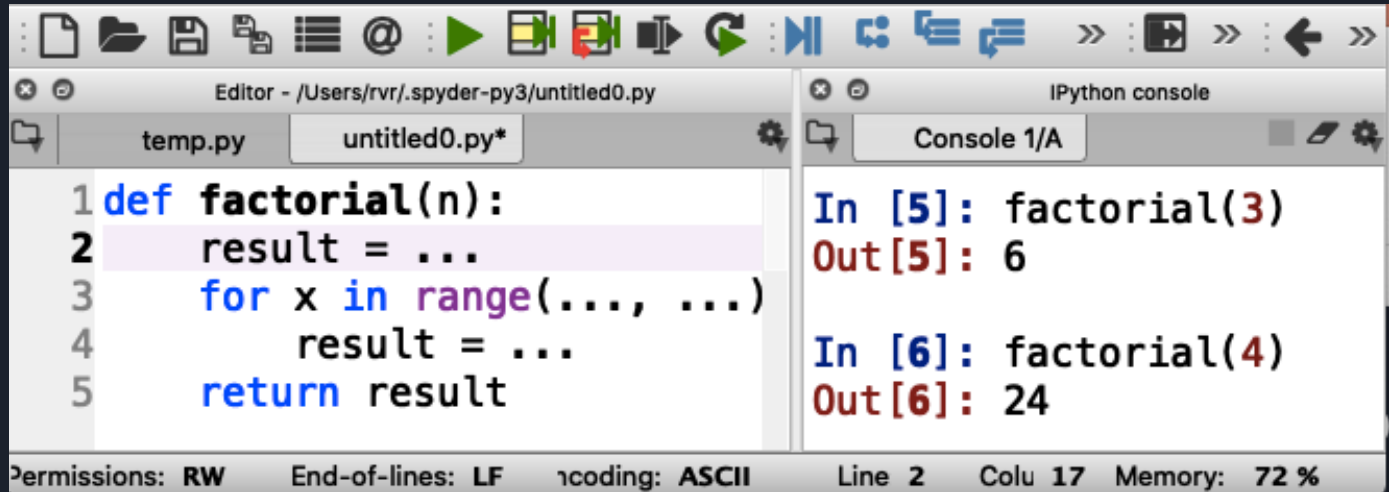
In [155]: squares(1, 10)  
Out[155]: [1, 4, 9, 16, 25, 36, 49, 64, 81]  
In [156]: |

Permissions: RW End-of-lines: LF Encoding: ASCII Line: 5 Column: 18 Memory: 68 %

# Your turn to practice with ranges

Write function `factorial(n)` that returns  $n!$

- $n!$  is defined to be the product of all the integers from 1 to  $n$
- For example, `factorial(3)` is equal to  $1 * 2 * 3$  (that is, 6)



The screenshot shows a code editor window titled "Editor - /Users/rvr/.spyder-py3/untitled0.py" with two tabs: "temp.py" and "untitled0.py\*". The code in the editor is:

```
1 def factorial(n):
2     result = ...
3     for x in range(..., ...)
4         result = ...
5     return result
```

The IPython console window titled "IPython console" shows the execution of the function:

```
In [5]: factorial(3)
Out[5]: 6

In [6]: factorial(4)
Out[6]: 24
```

The status bar at the bottom indicates: Permissions: RW, End-of-lines: LF, encoding: ASCII, Line 2, Colu 17, Memory: 72 %



# What have we learned about loops?

- You can use a loop to visit every element in a list (or string)
- Alternatively, you can use a loop to visit over a *range*
- `range(start, end)` includes `start` but not `end`
- Generic format of a function with a loop:

```
def function(Lst, x):  
    result = ...  
    for elt in Lst:  
        result = ...  
    return result
```

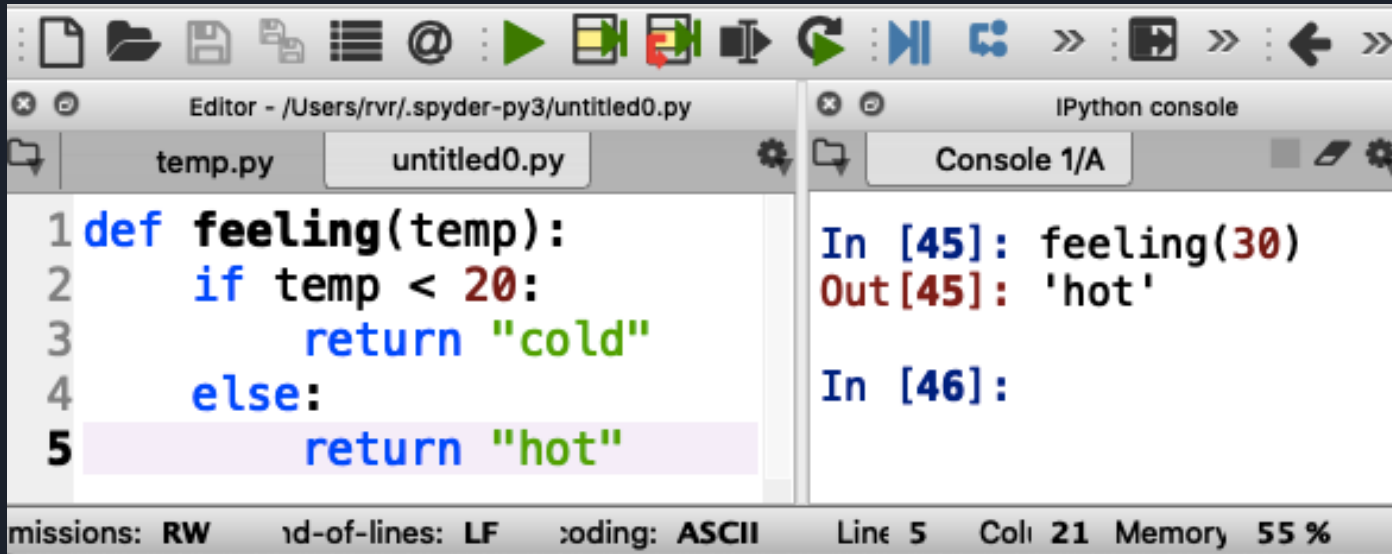


# IF STATEMENTS



What if you want to do something only sometimes?

Write function `feeling(temp)` that returns  
"cold" if `temp < 20` or "hot" otherwise



The screenshot shows a Python IDE with two panes. The left pane is the code editor, showing a function definition in `temp.py`. The right pane is the IPython console, showing the function being called with `feeling(30)` and returning the string `'hot'`. The status bar at the bottom indicates the current line and column.

```
1 def feeling(temp):  
2     if temp < 20:  
3         return "cold"  
4     else:  
5         return "hot"
```

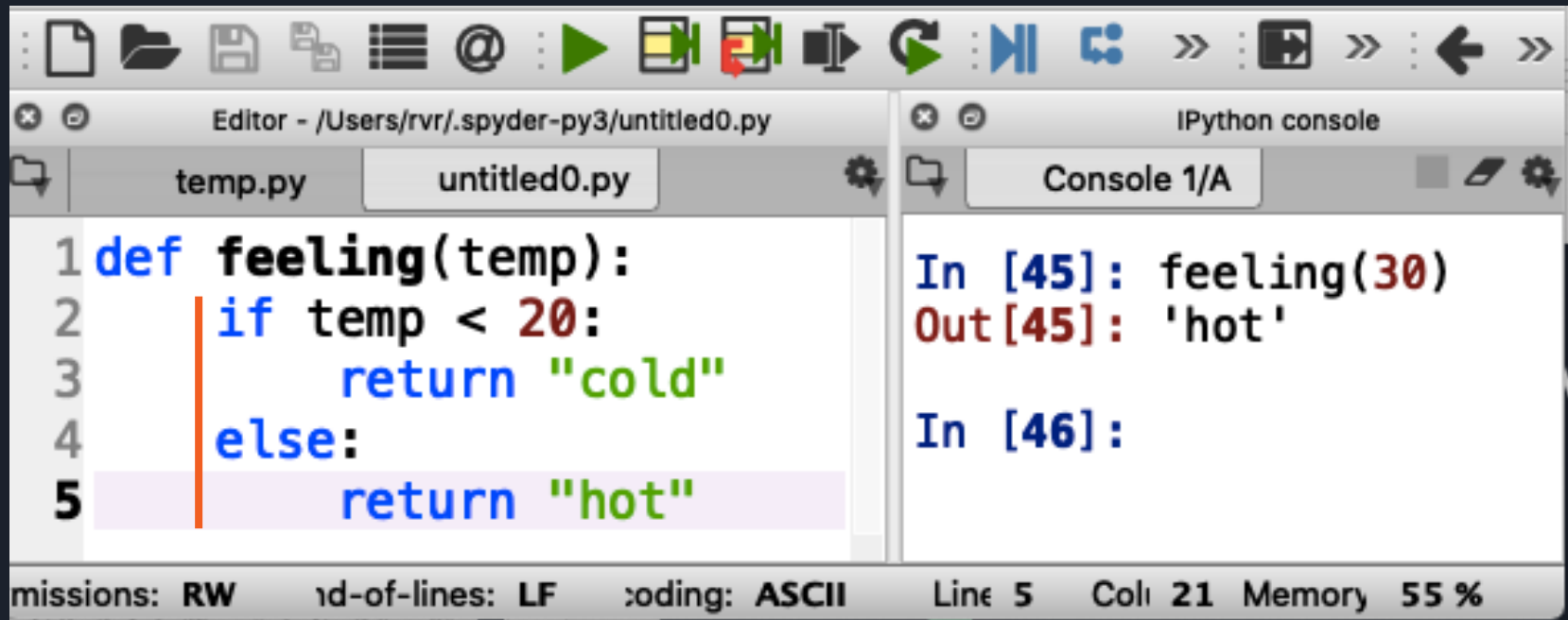
```
In [45]: feeling(30)  
Out[45]: 'hot'
```

```
In [46]:
```

missions: RW    id-of-lines: LF    coding: ASCII    Line 5    Col 21    Memory 55 %



Correct indentation is important!



The screenshot shows a Python IDE with two main panes. The left pane is a code editor showing a function definition with correct indentation. The right pane is an IPython console showing the function being called with an argument of 30, resulting in the output 'hot'. The status bar at the bottom indicates the current line and column.

```
1 def feeling(temp):  
2     if temp < 20:  
3         return "cold"  
4     else:  
5         return "hot"
```

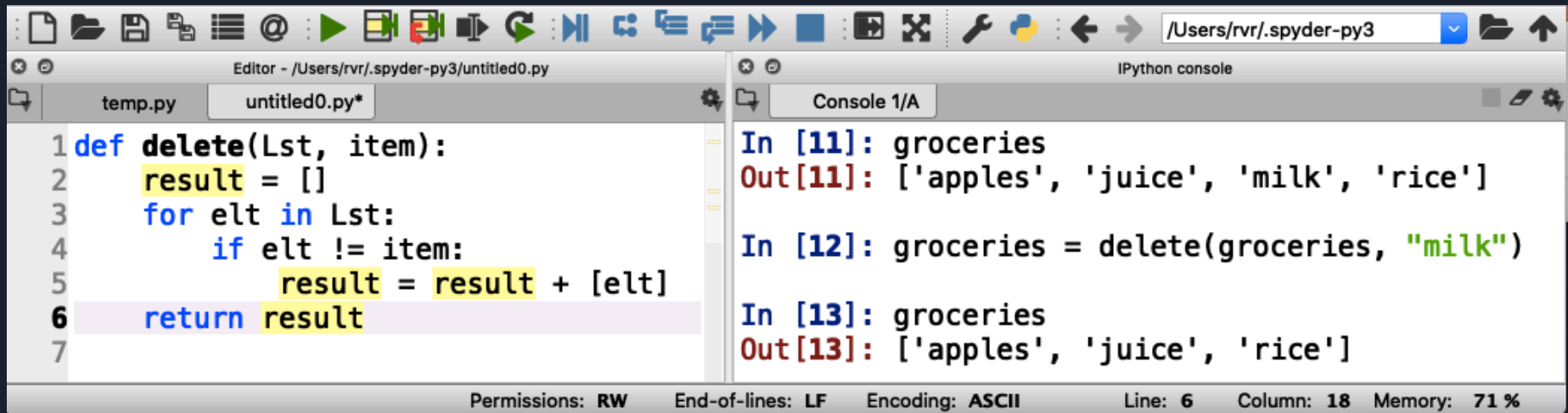
In [45]: feeling(30)  
Out [45]: 'hot'

In [46]:

missions: RW    id-of-lines: LF    coding: ASCII    Line 5    Col 21    Memory 55 %

# Using if statements in for loops

Write a function `delete(Lst, item)` that returns a list consisting of the elements of `Lst` except for `item`



The screenshot shows a Python IDE with two panes. The left pane is the code editor, and the right pane is the IPython console.

```
1 def delete(Lst, item):
2     result = []
3     for elt in Lst:
4         if elt != item:
5             result = result + [elt]
6     return result
7
```

The console shows the following output:

```
In [11]: groceries
Out[11]: ['apples', 'juice', 'milk', 'rice']

In [12]: groceries = delete(groceries, "milk")

In [13]: groceries
Out[13]: ['apples', 'juice', 'rice']
```

The status bar at the bottom indicates: Permissions: RW End-of-lines: LF Encoding: ASCII Line: 6 Column: 18 Memory: 71 %

`!=` means “is not the same as”



# Testing for equality

`==` is used to check if two values are the same

(do not confuse with `=`, which is used to assign a value to a variable!)

For example: `if elt == x:`

`. . .`

## Correct indentation

```
1 def delete(Lst, item):
2     total = []
3     for x in Lst:
4         if x != item:
5             total = total + [x]
6     return total
```

# Incorrect indentations: what do these do?

```
def delete(Lst, item):  
    total = []  
    for x in Lst:  
        if x != item:  
            total = total + [x]  
    return total
```

```
def delete(Lst, item):  
    total = []  
    for x in Lst:  
        if x != item:  
            total = total + [x]  
    return total
```

## Incorrect indentations: what do these do?

```
def delete(Lst, item):  
    total = []  
    for x in Lst:  
        if x != item:  
            total = total + [x]  
    return total
```

This returns after  
the first element  
in `Lst`

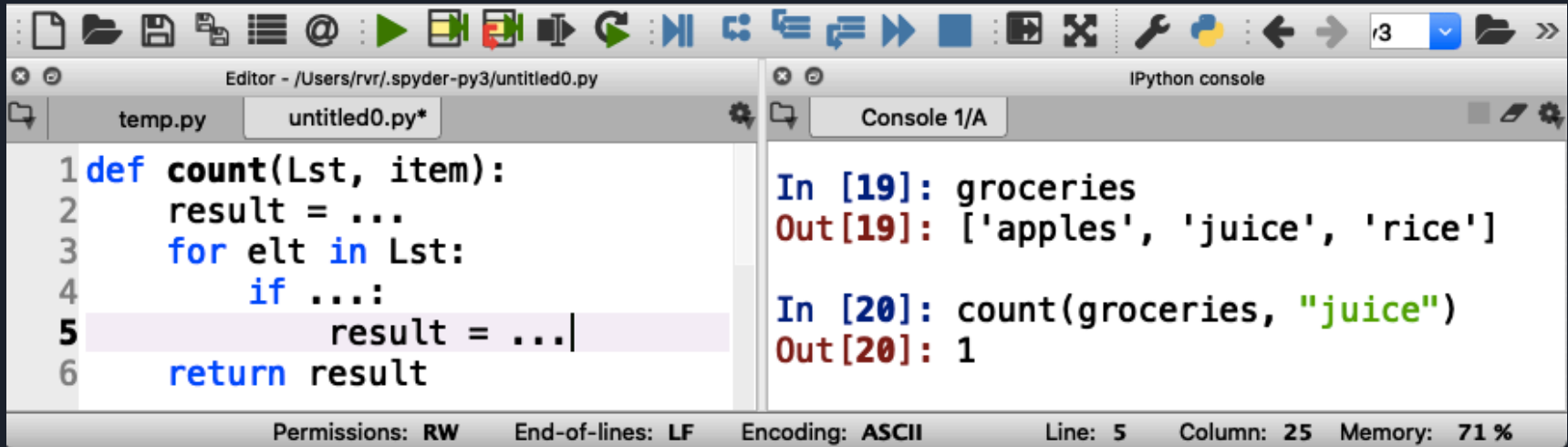
```
def delete(Lst, item):  
    total = []  
    for x in Lst:  
        if x != item:  
            total = total + [x]  
    return total
```

This returns after  
the first element  
that is not `item`

# Your turn to practice

Write a function `count(Lst, item)` that counts how many times `item` occurs in `Lst`

- For example, `count(["apples", "juice", "apples"], "apples")` should return 2



The screenshot shows a Python IDE with two windows. The left window is an editor showing a function definition in `untitled0.py*`:

```
1 def count(Lst, item):
2     result = ...
3     for elt in Lst:
4         if ...:
5             result = ...
6     return result
```

The right window is the IPython console, showing the execution of the function:

```
In [19]: groceries
Out[19]: ['apples', 'juice', 'rice']

In [20]: count(groceries, "juice")
Out[20]: 1
```

The status bar at the bottom indicates: Permissions: RW, End-of-lines: LF, Encoding: ASCII, Line: 5, Column: 25, Memory: 71 %.

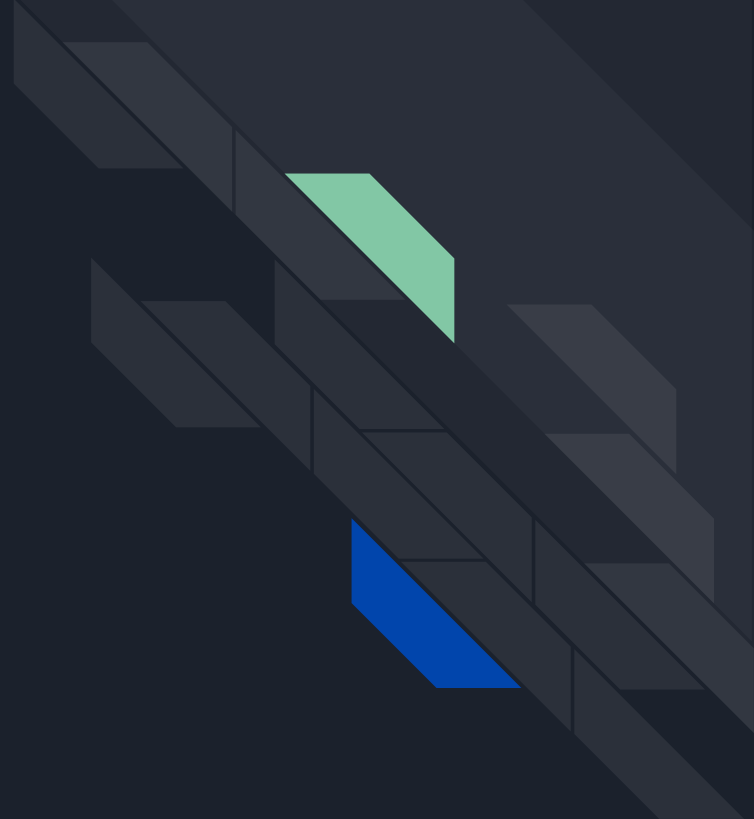


## What have we learned about *if* statements?

- You can use an *if statement* to execute code based on a condition
- An *if* statement may have an “*else*” part (but not required)
- You have to get indentation just so
- You can use `!=` to test if two values are different or not
- By the way, `!=` works on numbers, strings, and even lists!



Programs with multiple  
functions





# How to write difficult programs

## Split program into multiple functions!

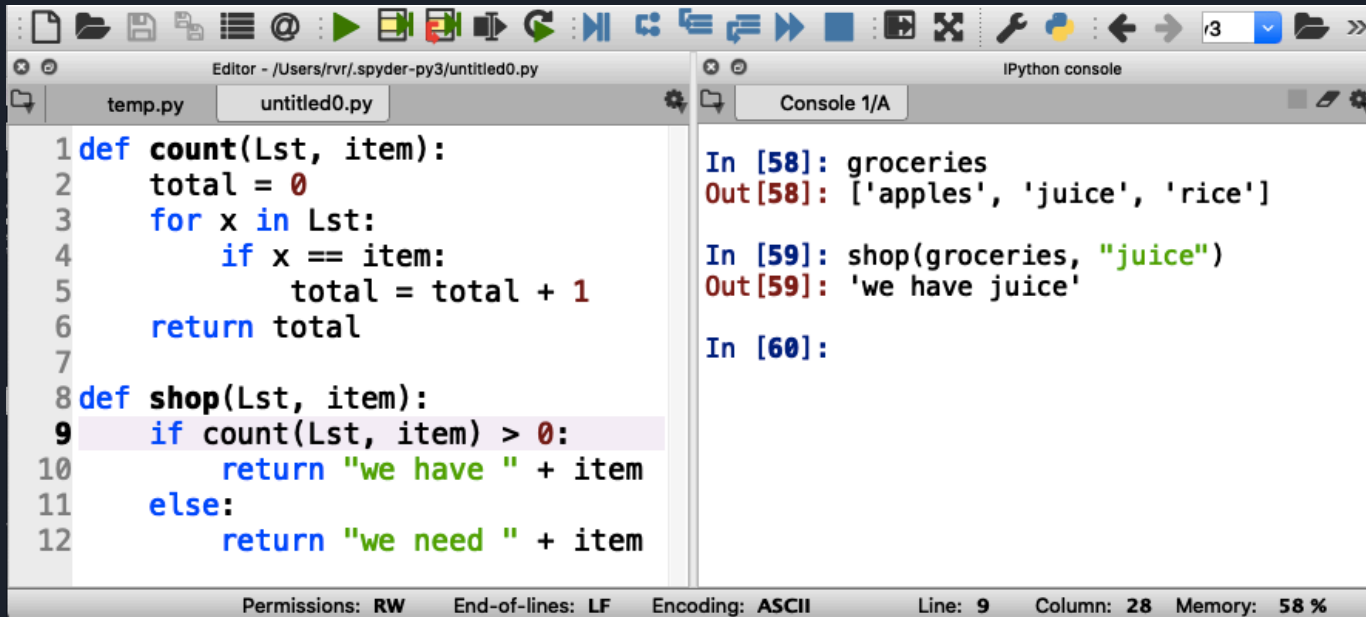
- Write a function `shop(Lst, item)` that returns the string  
    `"we have item"` if `item` is in `Lst`, or  
    `"we need item"` if not

For example: `shop(groceries, "rice")` returns  
`"we have rice"` if `"rice"` is in `groceries`

# How to write difficult programs

## Split program into multiple functions!

- Write a function `shop(Lst, item)` that returns the string "we have item" if `item` is in `Lst`, or "we need item" if not



The screenshot shows a Python IDE with two panes. The left pane is a code editor showing the following code:

```
1 def count(Lst, item):
2     total = 0
3     for x in Lst:
4         if x == item:
5             total = total + 1
6     return total
7
8 def shop(Lst, item):
9     if count(Lst, item) > 0:
10        return "we have " + item
11    else:
12        return "we need " + item
```

The right pane is an IPython console showing the following output:

```
In [58]: groceries
Out[58]: ['apples', 'juice', 'rice']

In [59]: shop(groceries, "juice")
Out[59]: 'we have juice'

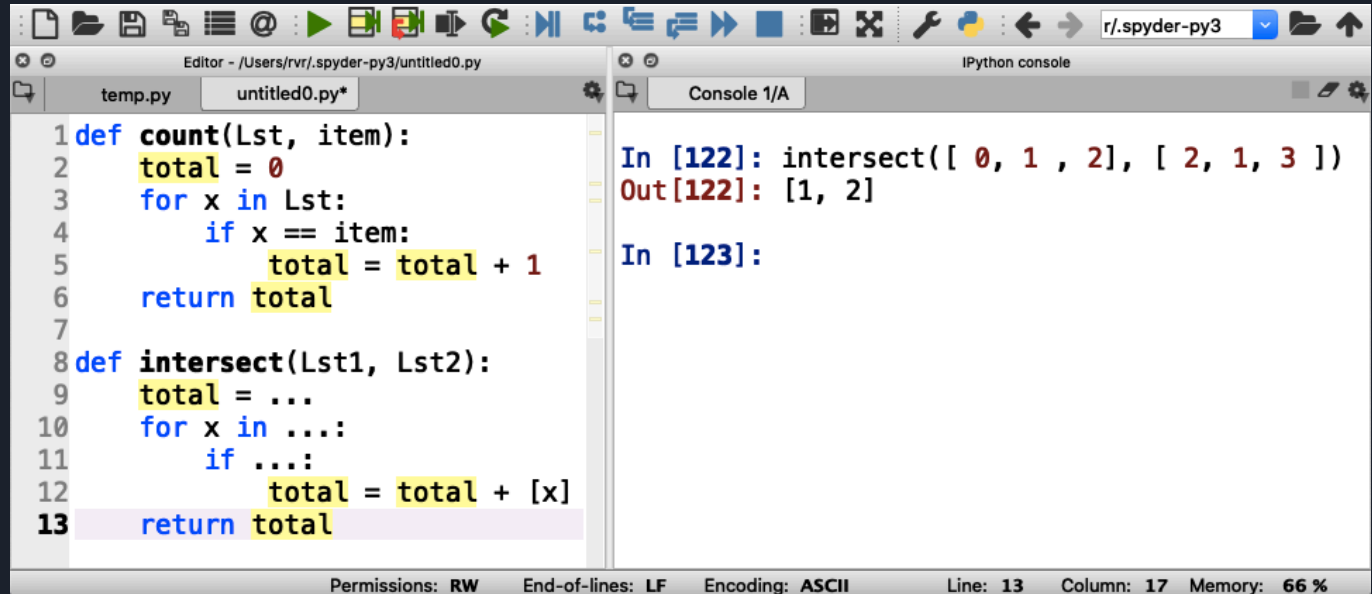
In [60]:
```

The status bar at the bottom of the IDE shows: Permissions: RW End-of-lines: LF Encoding: ASCII Line: 9 Column: 28 Memory: 58 %

# Practice writing a difficult program

Write a function `intersect(Lst1, Lst2)` that returns the intersection of lists `Lst1` and `Lst2` (a list of the elements that are in both lists).

- You can again use the `count(list, item)` function to simplify the task



The screenshot shows the Spyder Python IDE interface. The left pane is the code editor, and the right pane is the IPython console. The code editor contains the following Python code:


```
1 def count(Lst, item):
2     total = 0
3     for x in Lst:
4         if x == item:
5             total = total + 1
6     return total
7
8 def intersect(Lst1, Lst2):
9     total = ...
10    for x in ...:
11        if ...:
12            total = total + [x]
13    return total
```

The IPython console shows the following interaction:

```
In [122]: intersect([ 0, 1 , 2], [ 2, 1, 3 ])
Out[122]: [1, 2]

In [123]:
```

The status bar at the bottom of the IDE displays: Permissions: RW End-of-lines: LF Encoding: ASCII Line: 13 Column: 17 Memory: 66 %



Just a little math  
before we go on



## Two types of numbers

- “integers”
  - Examples: 0, 1, 2, 3, -3, 93723881
- “floating point numbers”
  - Examples: 3.14159, -0.05, 123.45



# Division with integers and floating point numbers

- What is 9 divided by 4?
  - Using integers, it is 2 with a remainder of 1
  - Using floating point numbers, it is 2.25

# Same thing in Python

```
In [6]: 9 / 4
Out[6]: 2.25

In [7]: 9 // 4
Out[7]: 2

In [8]: 9 % 4
Out[8]: 1
```

Permissions: RW    End-of-lines: LF    Encoding: ASCII    Line: 1    Colum 2    Memory: 64 %

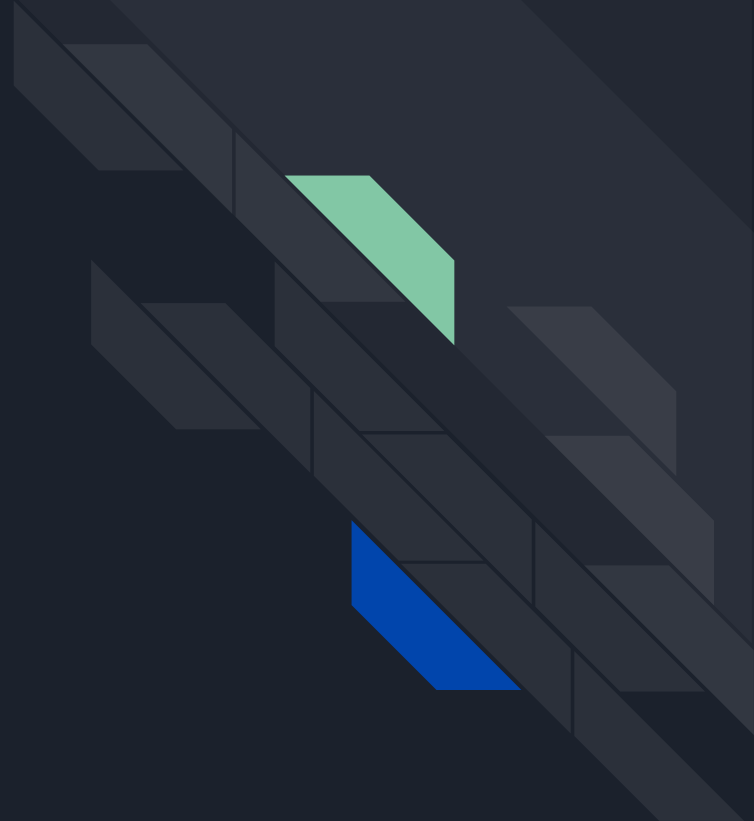
Floating point division

Integer division

Integer remainder (modulo)  
You say "9 modulo 4"



# Approaching Coding Questions





# A 3-Step Approach To Coding Problems

1. Read the question carefully. Understand what information you have available to you and what the question wants you to do.
2. Try to write out an algorithm or solution to the problem in “English” or “pseudocode”.
3. Translate your English solution to code



## Example

Q: Write a function `oddOrEven(x)` that returns "even" if `x` is even and "odd" if not



Step 1: Read the question *carefully*

Q: Write a function `oddOrEven(x)` that returns "even" if `x` is even and "odd" if not

After reading the question, we see that we have one integer number as our only input.

Since our function only has one input, our function header will look something like:

```
def oddOrEven(x) :
```



## Step 2: Design a solution in “English”

**Q: Write a function `oddOrEven(x)` that returns “even” if `x` is even and “odd” if not**

- If the input parameter is divisible by 2, then we know it is even
- Otherwise, it must be odd

We also know that we can check if something is divisible with the modulo operator, %



## Step 3: translate English into code

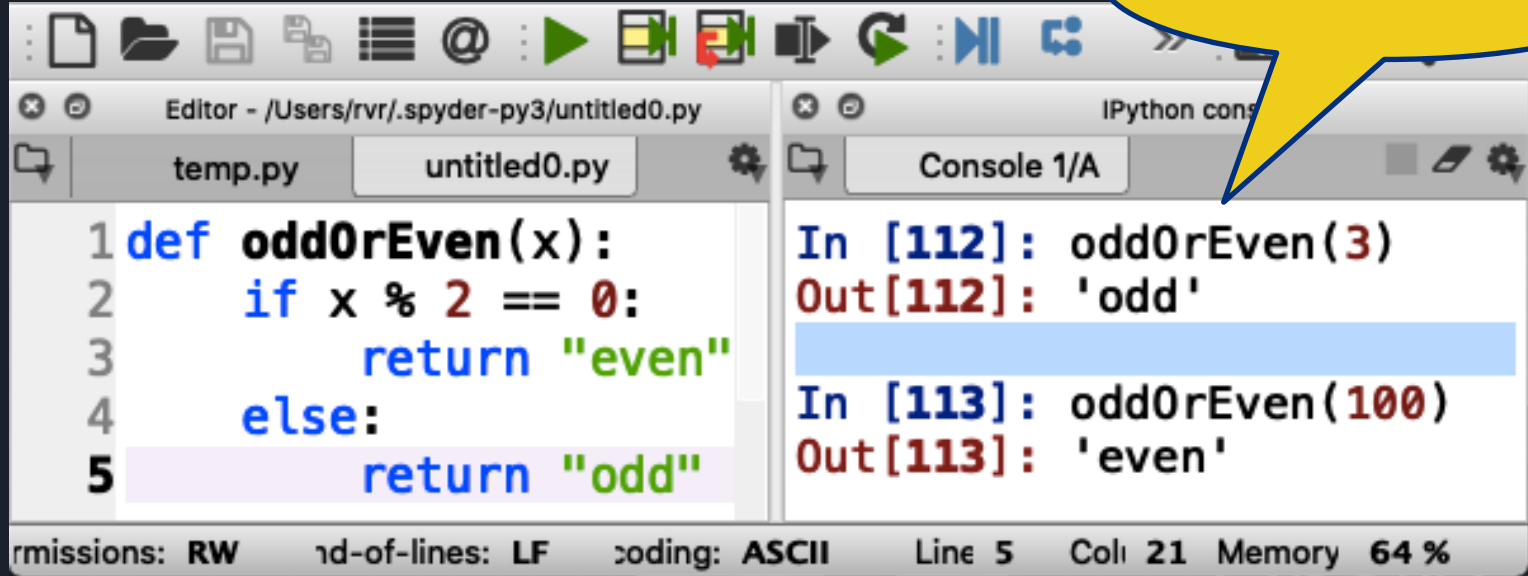
If the input parameter is divisible by 2, then it is even

```
if num % 2 == 0:  
    return "even"
```

Otherwise, it must be odd

```
else:  
    return "odd"
```

Finally, test it out



The screenshot shows a Python IDE with two main panes. The left pane is a code editor showing a function definition for `oddOrEven(x)`. The function checks if `x` is even (`x % 2 == 0`) and returns "even", otherwise it returns "odd". The right pane is an IPython console showing the function being called with `3` and `100`, returning `'odd'` and `'even'` respectively. A yellow speech bubble points to the console with the text "Test carefully!".

```
1 def oddOrEven(x):  
2     if x % 2 == 0:  
3         return "even"  
4     else:  
5         return "odd"
```

In [112]: oddOrEven(3)  
Out[112]: 'odd'

In [113]: oddOrEven(100)  
Out[113]: 'even'

Permissions: RW | End-of-lines: LF | Encoding: ASCII | Line 5 | Col 21 | Memory 64 %

Test carefully!



Time to practice

Write a function `isPrime(n)` that returns "yes" if  $n$  is prime and "no" otherwise

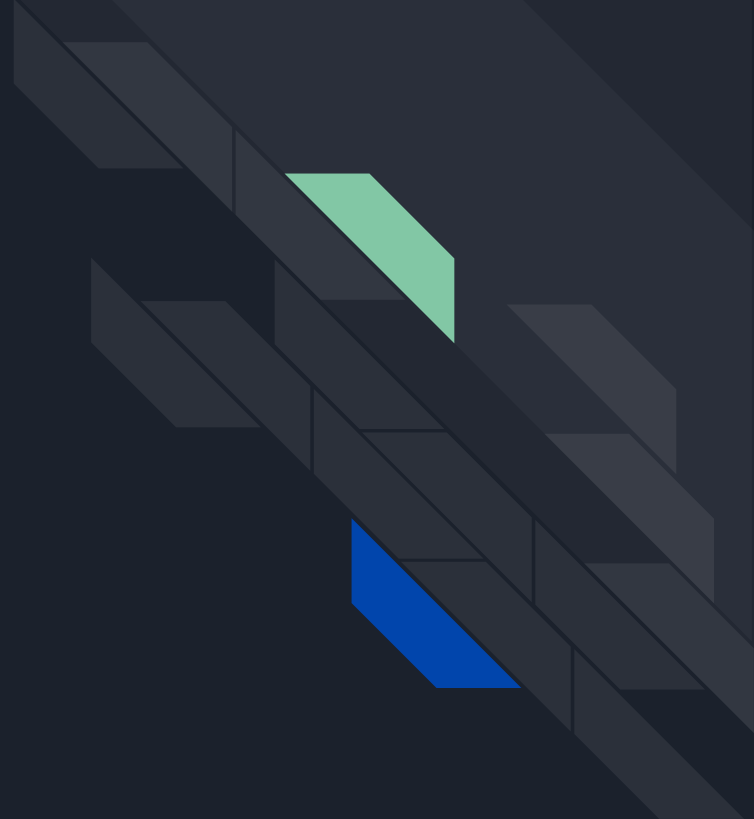
In English: a number  $n$  is prime if it can only be divided by 1 and itself

(1 is an exception: it is not considered prime)

So, try to divide  $n$  by all numbers between 2 and  $n-1$  and make sure there is always a non-zero remainder

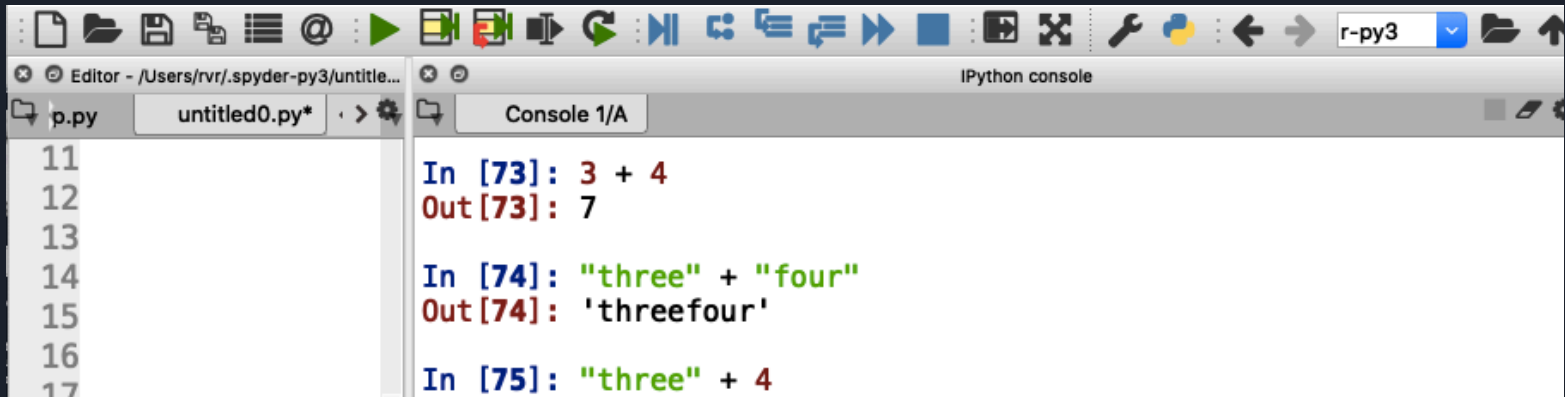


# A primer on conversion



# Strings and Numbers

- $3 + 4 == 7$
- `"three" + "four" == "threefour"`
- What is `"three" + 4`???



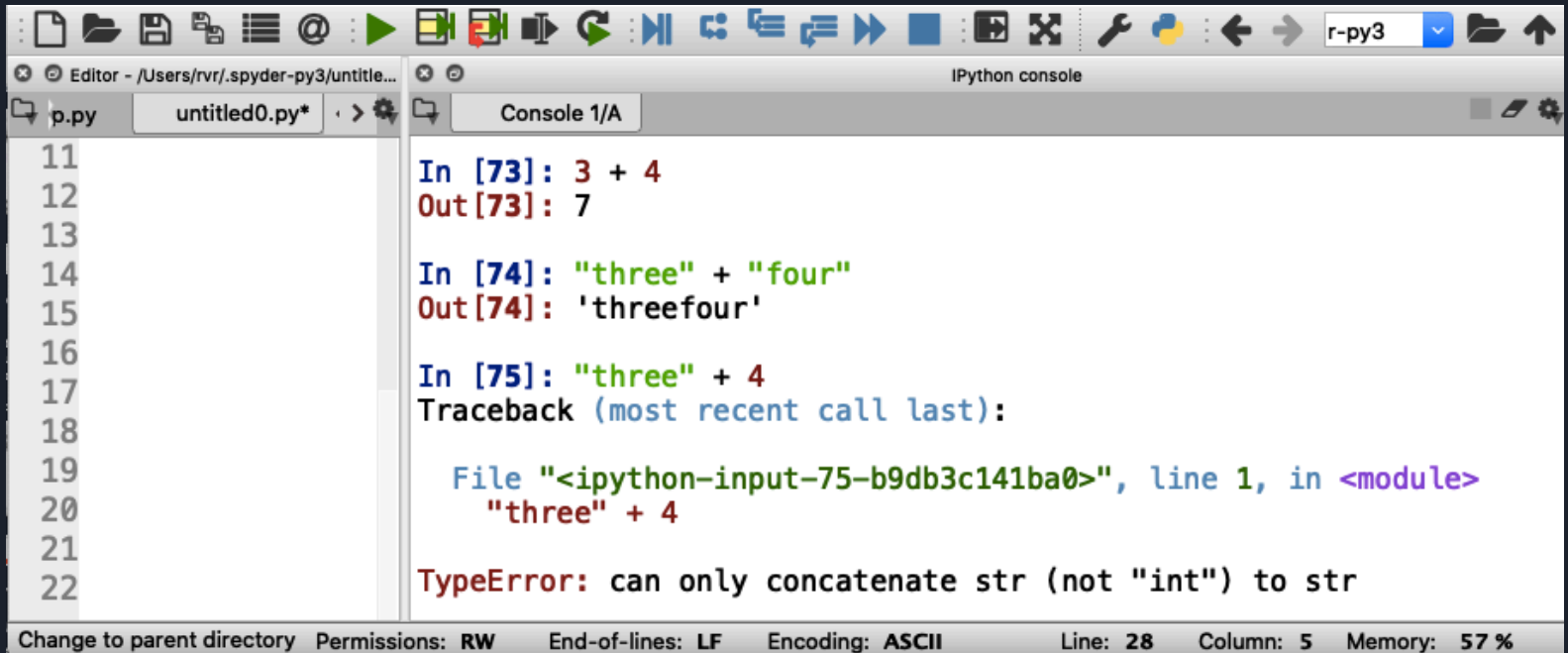
The screenshot shows a Python IDE window with a toolbar at the top. The main area is split into two panes. The left pane shows a code editor with line numbers 11 through 17. The right pane is the IPython console, titled 'Console 1/A', showing the results of three code executions:

```
In [73]: 3 + 4
Out[73]: 7

In [74]: "three" + "four"
Out[74]: 'threefour'

In [75]: "three" + 4
```

# Strings and Numbers



```
11
12
13
14
15
16
17
18
19
20
21
22

In [73]: 3 + 4
Out[73]: 7

In [74]: "three" + "four"
Out[74]: 'threefour'

In [75]: "three" + 4
Traceback (most recent call last):

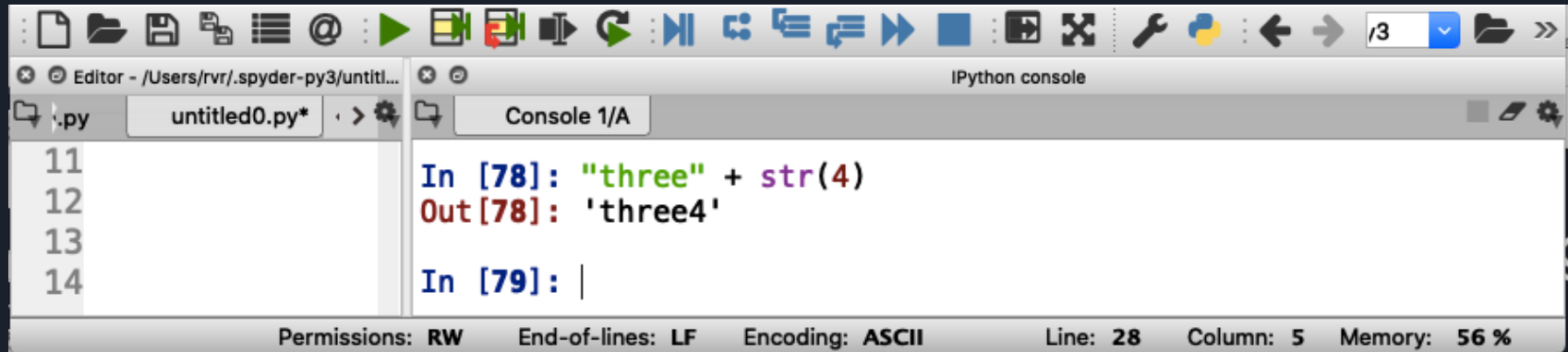
  File "<ipython-input-75-b9db3c141ba0>", line 1, in <module>
    "three" + 4

TypeError: can only concatenate str (not "int") to str

Change to parent directory  Permissions: RW  End-of-lines: LF  Encoding: ASCII  Line: 28  Column: 5  Memory: 57 %
```

# Converting integers to strings

- `str(4)` is the string `"4"`

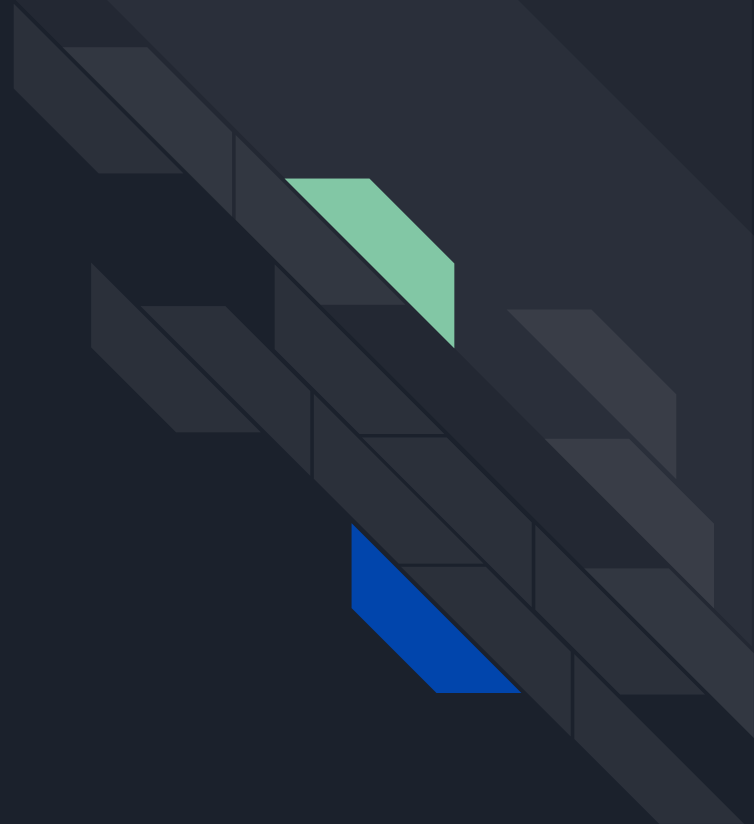


```
11  
12  
13  
14
```

```
In [78]: "three" + str(4)  
Out[78]: 'three4'  
  
In [79]: |
```

Permissions: RW End-of-lines: LF Encoding: ASCII Line: 28 Column: 5 Memory: 56 %

Input/Output



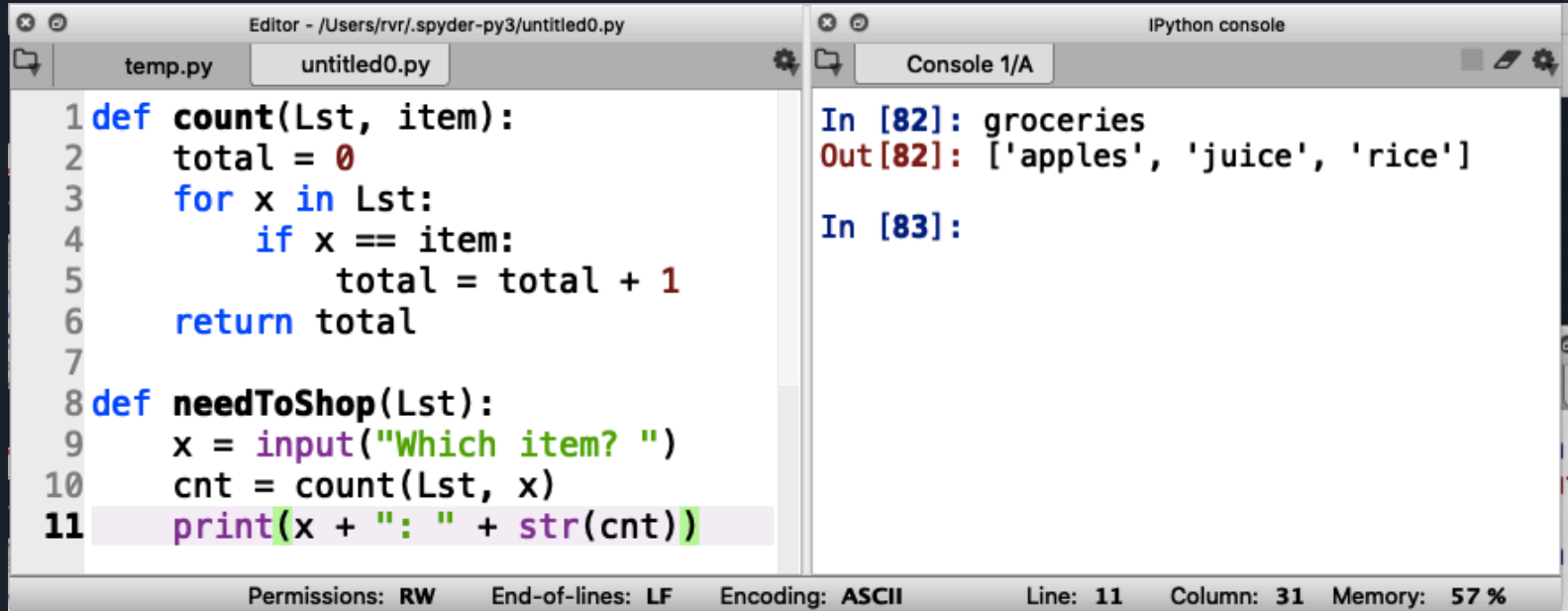


Python functions can “print” (output) and read from the keyboard or even files (input)

Write a function `needToShop (Lst)` that asks for the name of an item and prints the item and how many instances of the item are in `Lst`

Note: the item is *not* an input parameter to `needToShop (Lst)`

# Function `needToShop(Lst)`



The image shows a screenshot of a Python IDE with two windows. The left window is an editor titled 'Editor - /Users/rvr/.spyder-py3/untitled0.py' containing a Python script. The right window is an IPython console titled 'IPython console' showing the execution of the script.

```
1 def count(Lst, item):
2     total = 0
3     for x in Lst:
4         if x == item:
5             total = total + 1
6     return total
7
8 def needToShop(Lst):
9     x = input("Which item? ")
10    cnt = count(Lst, x)
11    print(x + ": " + str(cnt))
```

The console shows the following output:

```
In [82]: groceries
Out[82]: ['apples', 'juice', 'rice']

In [83]:
```

At the bottom of the IDE, the status bar displays: Permissions: RW End-of-lines: LF Encoding: ASCII Line: 11 Column: 31 Memory: 57 %

# Function needToShop(Lst)

Read  
input

```
Editor - /Users/rvr/.spyder-py3/untitled0.py
temp.py  untitled0.py

1 def count(Lst, item):
2     total = 0
3     for x in Lst:
4         if x == item:
5             total = total + 1
6     return total
7
8 def needToShop(Lst):
9     x = input("Which item? ")
10    cnt = count(Lst, x)
11    print(x + ": " + str(cnt))

IPython console
Console 1/A

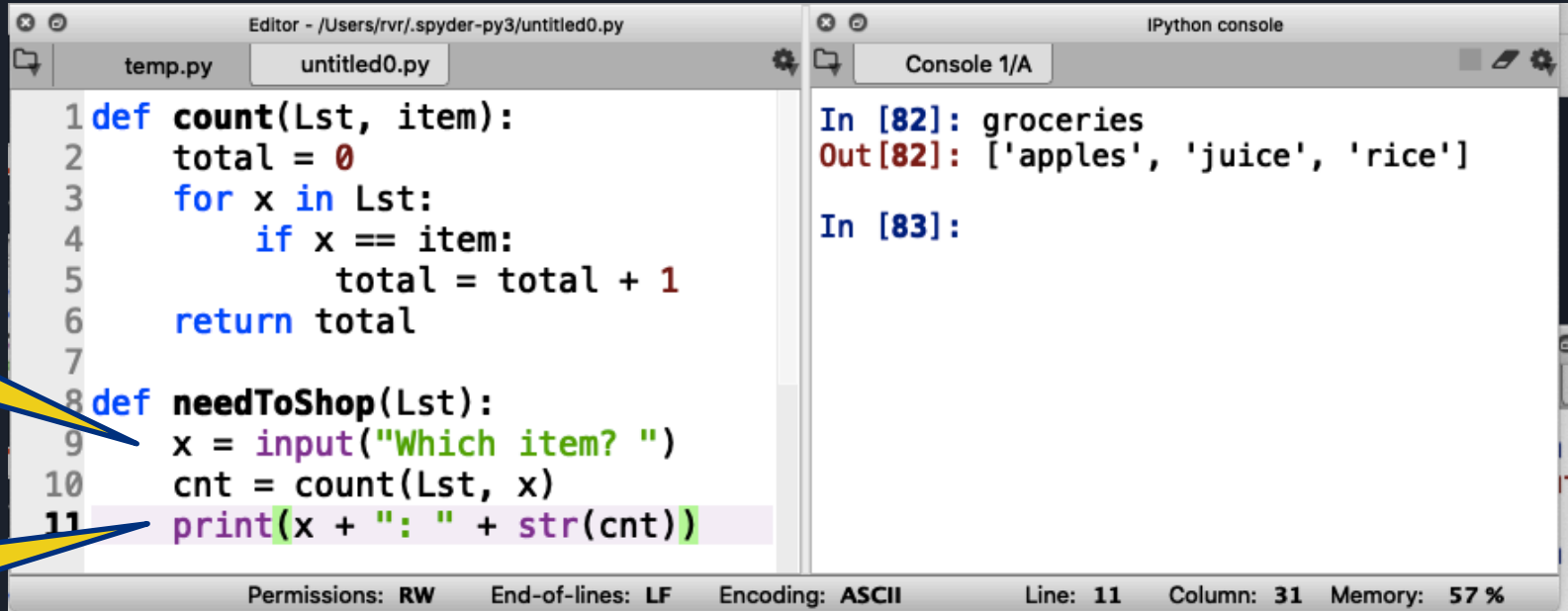
In [82]: groceries
Out[82]: ['apples', 'juice', 'rice']

In [83]:
```

Permissions: RW End-of-lines: LF Encoding: ASCII Line: 11 Column: 31 Memory: 57 %



# Function needToShop(Lst)



The image shows a Python IDE window with two panes. The left pane is a code editor showing a Python script named 'untitled0.py'. The right pane is an IPython console showing the execution of the script. The code in the editor defines a function 'count' and a function 'needToShop'. The 'needToShop' function prompts the user for an item and prints the count of that item in a list. The console shows the output of the 'needToShop' function, which is a list of items: ['apples', 'juice', 'rice'].

```
1 def count(Lst, item):
2     total = 0
3     for x in Lst:
4         if x == item:
5             total = total + 1
6     return total
7
8 def needToShop(Lst):
9     x = input("Which item? ")
10    cnt = count(Lst, x)
11    print(x + ": " + str(cnt))
```

IPython console output:

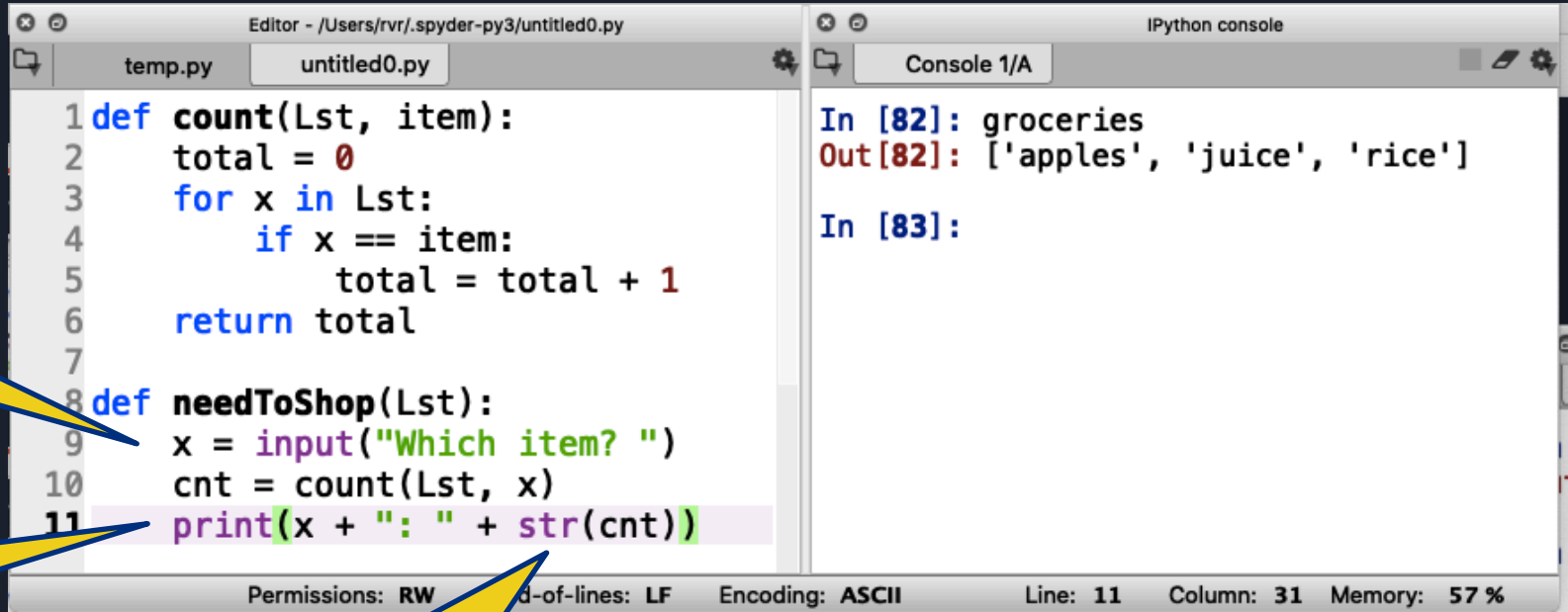
```
In [82]: groceries
Out[82]: ['apples', 'juice', 'rice']
In [83]:
```

Permissions: RW End-of-lines: LF Encoding: ASCII Line: 11 Column: 31 Memory: 57 %

Read  
input

Write  
output

# Function needToShop(Lst)



The screenshot shows a Python IDE with two windows. The left window, titled 'Editor - /Users/rvr/.spyder-py3/untitled0.py', contains the following code:

```
1 def count(Lst, item):
2     total = 0
3     for x in Lst:
4         if x == item:
5             total = total + 1
6     return total
7
8 def needToShop(Lst):
9     x = input("Which item? ")
10    cnt = count(Lst, x)
11    print(x + ": " + str(cnt))
```

The right window, titled 'IPython console', shows the execution of the code:

```
In [82]: groceries
Out[82]: ['apples', 'juice', 'rice']

In [83]:
```

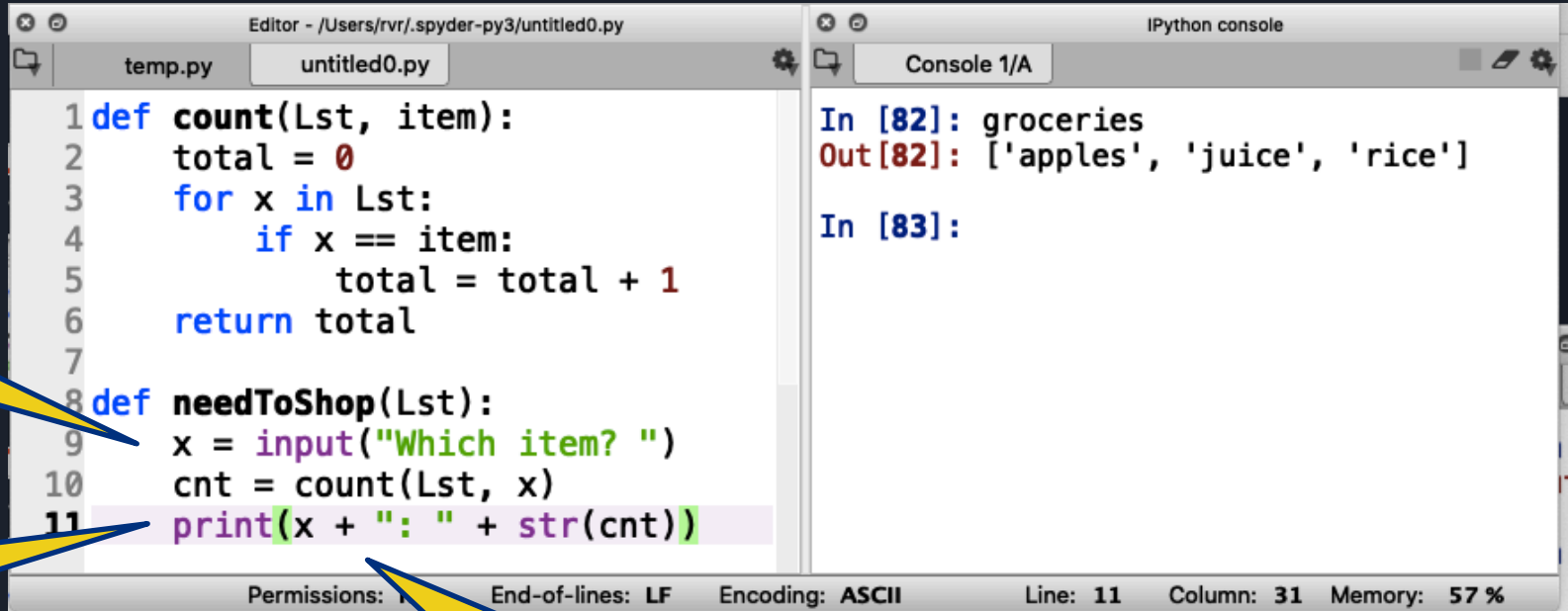
At the bottom of the IDE, the status bar displays: Permissions: RW, End-of-lines: LF, Encoding: ASCII, Line: 11, Column: 31, Memory: 57 %.

Read input

Write output

Note integer to string conversion

# Function needToShop(Lst)



The image shows a screenshot of a Python IDE with two windows. The left window, titled 'Editor - /Users/rvr/.spyder-py3/untitled0.py', contains the following Python code:

```
1 def count(Lst, item):
2     total = 0
3     for x in Lst:
4         if x == item:
5             total = total + 1
6     return total
7
8 def needToShop(Lst):
9     x = input("Which item? ")
10    cnt = count(Lst, x)
11    print(x + ": " + str(cnt))
```

The right window, titled 'IPython console', shows the execution of the code:

```
In [82]: groceries
Out[82]: ['apples', 'juice', 'rice']

In [83]:
```

At the bottom of the IDE, the status bar displays: 'Permissions: ... End-of-lines: LF Encoding: ASCII Line: 11 Column: 31 Memory: 57 %'.

Read input

Write output

no return statement

# Function needToShop(Lst)

Read  
input

Write  
output

```
Editor - /Users/rvr/.spyder-py3/untitled0.py
temp.py  untitled0.py

1 def count(Lst, item):
2     total = 0
3     for x in Lst:
4         if x == item:
5             total = total + 1
6     return total
7
8 def needToShop(Lst):
9     x = input("Which item? ")
10    cnt = count(Lst, x)
11    print(x + ": " + str(cnt))

In [82]:
Out[82]: ['a', 'juice', 'rice']

In [83]: needToShop(groceries)

Which item?
```

Permissions: RW End-of-lines: LF Encoding: ASCII Line: 11 Column: 31 Memory: 57 %

Call function here

# Function needToShop(Lst)

Read input

Write output

```
Editor - /Users/rvr/.spyder-py3/untitled0.py
temp.py  untitled0.py

1 def count(Lst, item):
2     total = 0
3     for x in Lst:
4         if x == item:
5             total = total + 1
6     return total
7
8 def needToShop(Lst):
9     x = input("Which item? ")
10    cnt = count(Lst, x)
11    print(x + ": " + str(cnt))

In [82]:
Out[82]: ['apple', 'juice', 'rice']

In [83]: needToShop(groceries)

Which item? juice
juice: 1

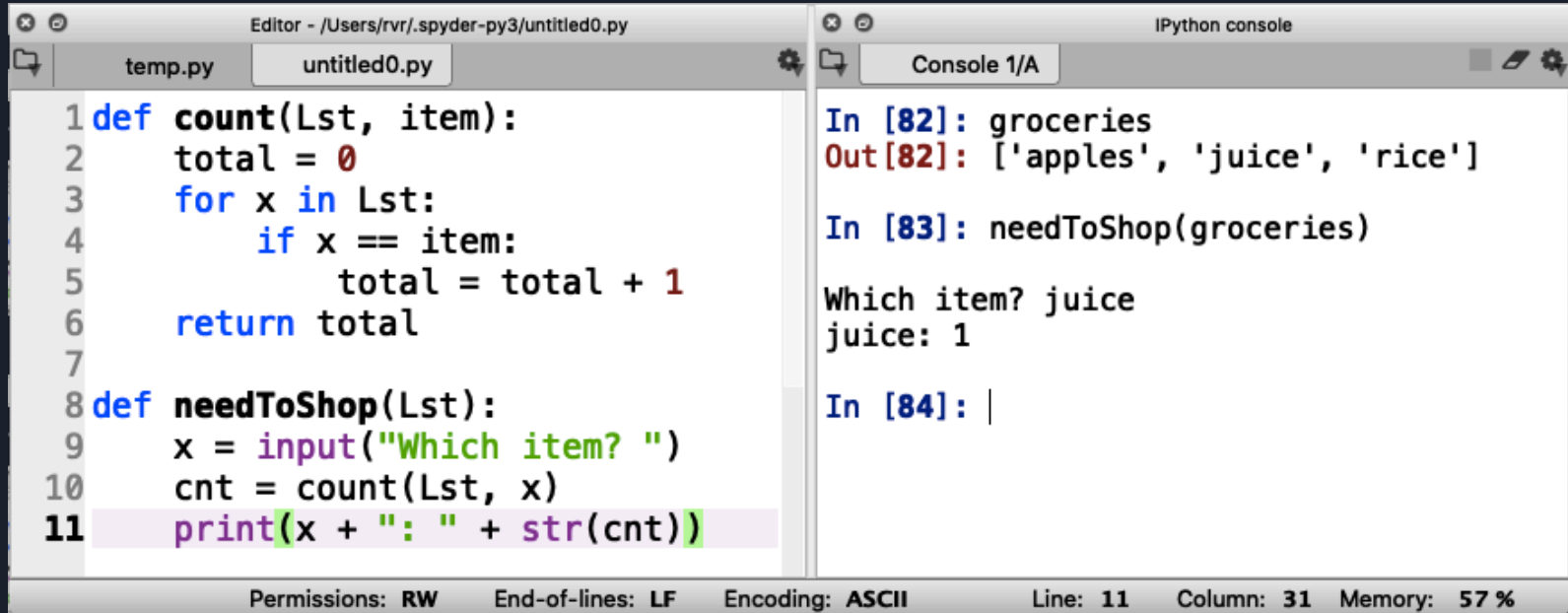
In [84]: |

Permissions: RW  End-of-lines: LF  Encoding: ASCII  Line: 11  Column: 31  Memory: 57 %
```

Call function here

Enter input here

Give it a shot yourself (and take turns!)



The screenshot shows a Python IDE with two windows: an editor and an IPython console. The editor window, titled 'Editor - /Users/rvr/.spyder-py3/untitled0.py', contains the following Python code:

```
1 def count(Lst, item):
2     total = 0
3     for x in Lst:
4         if x == item:
5             total = total + 1
6     return total
7
8 def needToShop(Lst):
9     x = input("Which item? ")
10    cnt = count(Lst, x)
11    print(x + ": " + str(cnt))
```

The IPython console window, titled 'IPython console', shows the following execution steps:

```
In [82]: groceries
Out[82]: ['apples', 'juice', 'rice']

In [83]: needToShop(groceries)

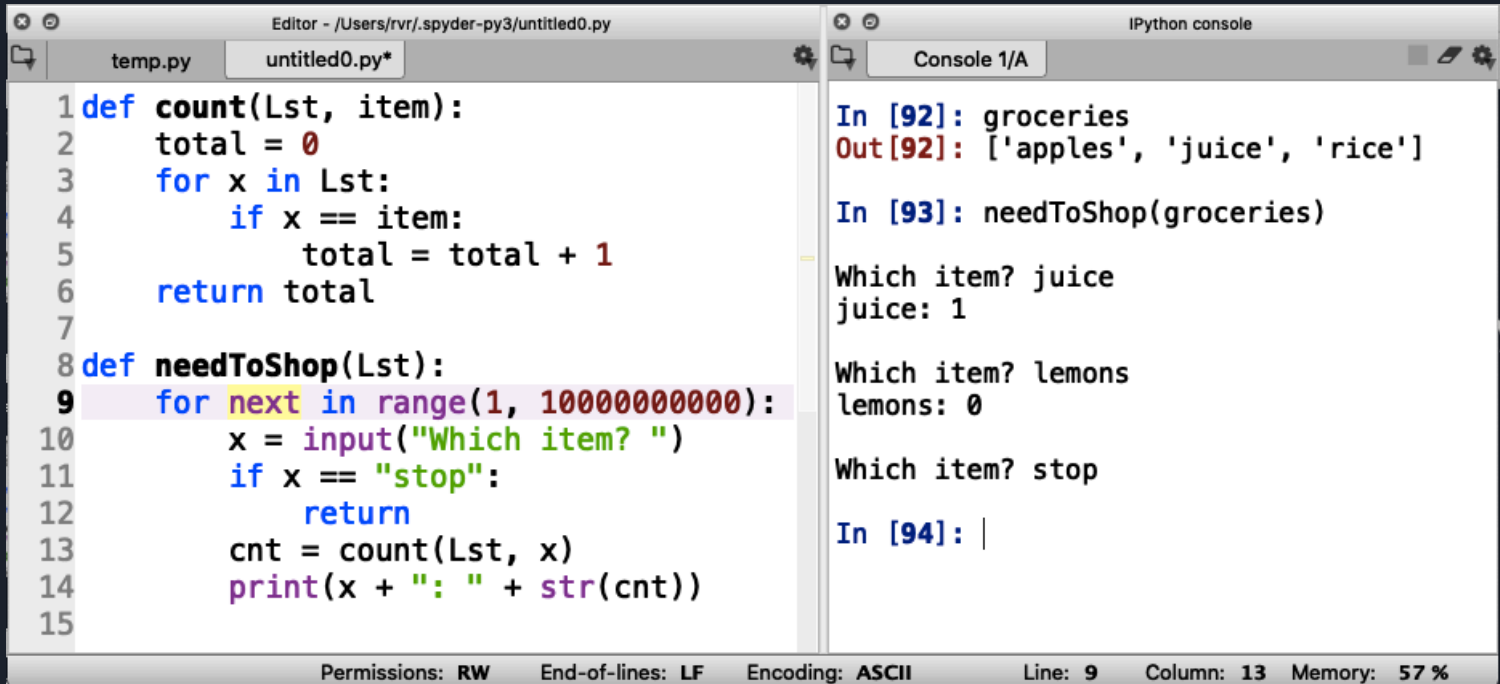
Which item? juice
juice: 1

In [84]: |
```

The status bar at the bottom of the IDE displays the following information: Permissions: RW, End-of-lines: LF, Encoding: ASCII, Line: 11, Column: 31, Memory: 57 %.

# Input/Output in a loop

How to ask for input over and over again?



The screenshot shows a Python IDE with two windows: 'temp.py' and 'IPython console'. The code in 'temp.py' defines two functions: 'count' and 'needToShop'. The 'needToShop' function uses a loop to repeatedly ask for input and calculate the count of items. The IPython console shows the execution of the code, including the input 'groceries' and the output ['apples', 'juice', 'rice'], followed by the input 'juice' and the output 'juice: 1', and then the input 'lemons' and the output 'lemons: 0', and finally the input 'stop' and the output 'stop: 0'.

```
1 def count(Lst, item):
2     total = 0
3     for x in Lst:
4         if x == item:
5             total = total + 1
6     return total
7
8 def needToShop(Lst):
9     for next in range(1, 10000000000):
10        x = input("Which item? ")
11        if x == "stop":
12            return
13        cnt = count(Lst, x)
14        print(x + ": " + str(cnt))
15
```

IPython console

```
In [92]: groceries
Out[92]: ['apples', 'juice', 'rice']

In [93]: needToShop(groceries)

Which item? juice
juice: 1

Which item? lemons
lemons: 0

Which item? stop

In [94]: |
```

Permissions: RW End-of-lines: LF Encoding: ASCII Line: 9 Column: 13 Memory: 57 %

# Input/Output in a loop

How to ask for input over and over again?

Long loop

```
Editor - /Users/rvr/.spyder-py3/untitled0.py
temp.py  untitled0.py*

1 def count(Lst, item):
2     total = 0
3     for x in Lst:
4         if x == item:
5             total = total + 1
6     return total
7
8 def needToShop(Lst):
9     for next in range(1, 10000000000):
10        x = input("Which item? ")
11        if x == "stop":
12            return
13        cnt = count(Lst, x)
14        print(x + ": " + str(cnt))
15

IPython console
Console 1/A

In [92]: groceries
Out[92]: ['apples', 'juice', 'rice']

In [93]: needToShop(groceries)

Which item? juice
juice: 1

Which item? lemons
lemons: 0

Which item? stop

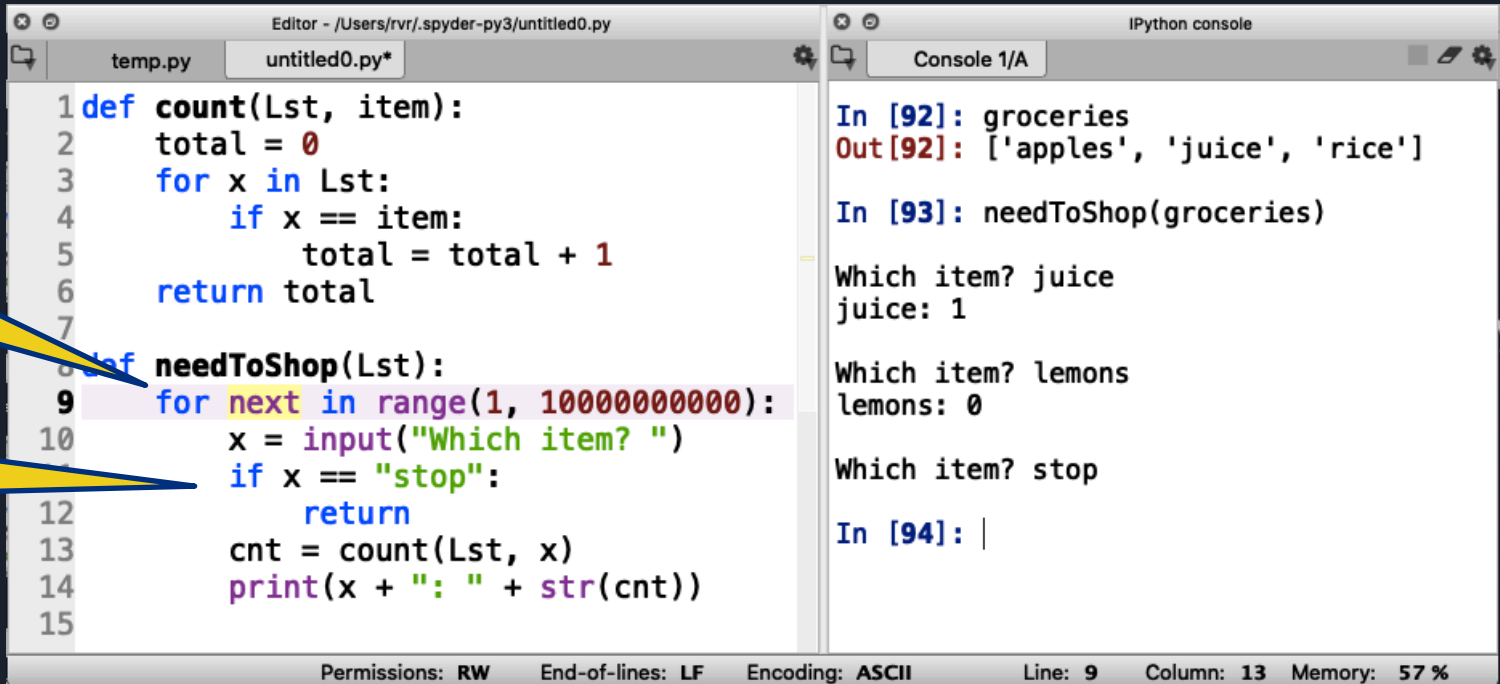
In [94]: |

Permissions: RW  End-of-lines: LF  Encoding: ASCII  Line: 9  Column: 13  Memory: 57 %
```



# Input/Output in a loop

How to ask for input over and over again?



```
1 def count(Lst, item):
2     total = 0
3     for x in Lst:
4         if x == item:
5             total = total + 1
6     return total
7
8 def needToShop(Lst):
9     for next in range(1, 10000000000):
10        x = input("Which item? ")
11        if x == "stop":
12            return
13        cnt = count(Lst, x)
14        print(x + ": " + str(cnt))
15
```

Permissions: RW End-of-lines: LF Encoding: ASCII Line: 9 Column: 13 Memory: 57 %

IPython console

```
In [92]: groceries
Out[92]: ['apples', 'juice', 'rice']

In [93]: needToShop(groceries)

Which item? juice
juice: 1

Which item? lemons
lemons: 0

Which item? stop

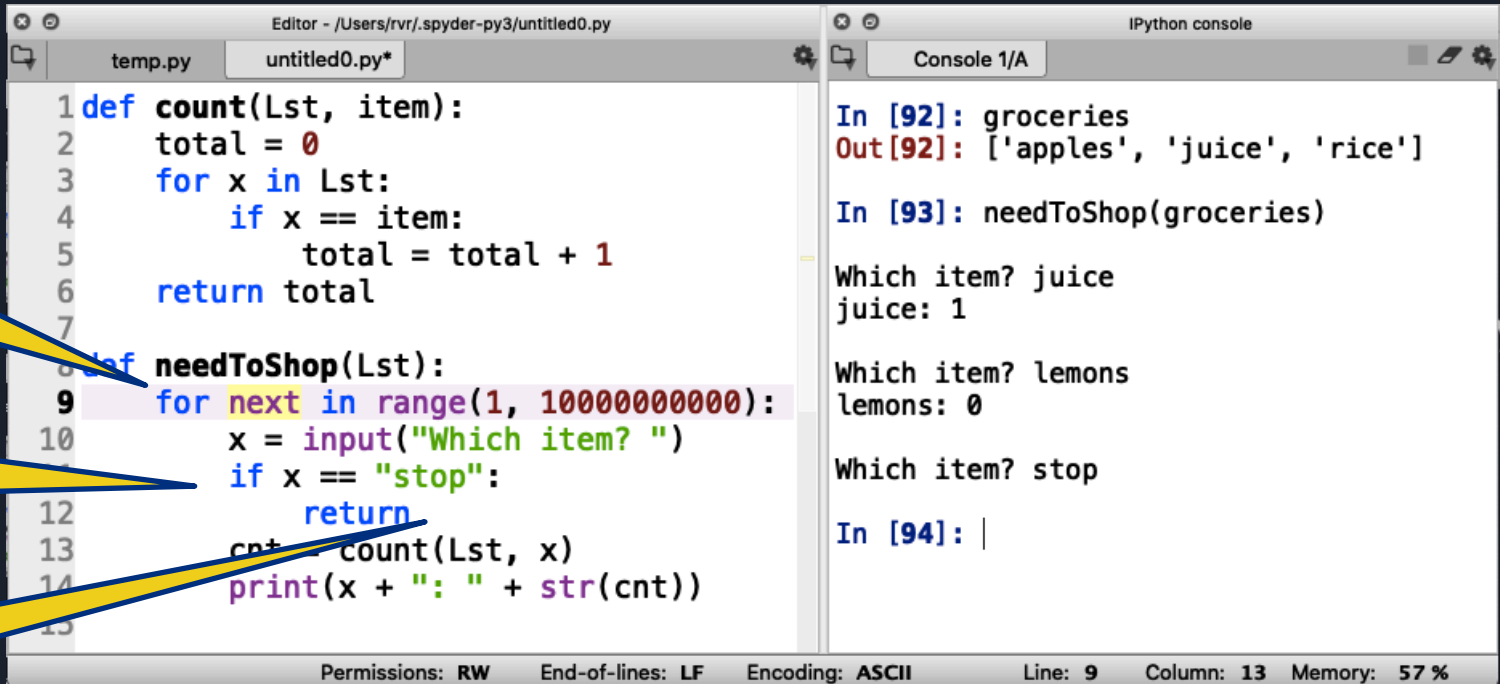
In [94]: |
```

Long loop

Add a way to stop

# Input/Output in a loop

How to ask for input over and over again?



```
1 def count(Lst, item):
2     total = 0
3     for x in Lst:
4         if x == item:
5             total = total + 1
6     return total
7
8 def needToShop(Lst):
9     for next in range(1, 10000000000):
10        x = input("Which item? ")
11        if x == "stop":
12            return
13        cnt = count(Lst, x)
14        print(x + ": " + str(cnt))
15
```

IPython console output:

```
In [92]: groceries
Out[92]: ['apples', 'juice', 'rice']

In [93]: needToShop(groceries)

Which item? juice
juice: 1

Which item? lemons
lemons: 0

Which item? stop

In [94]: |
```

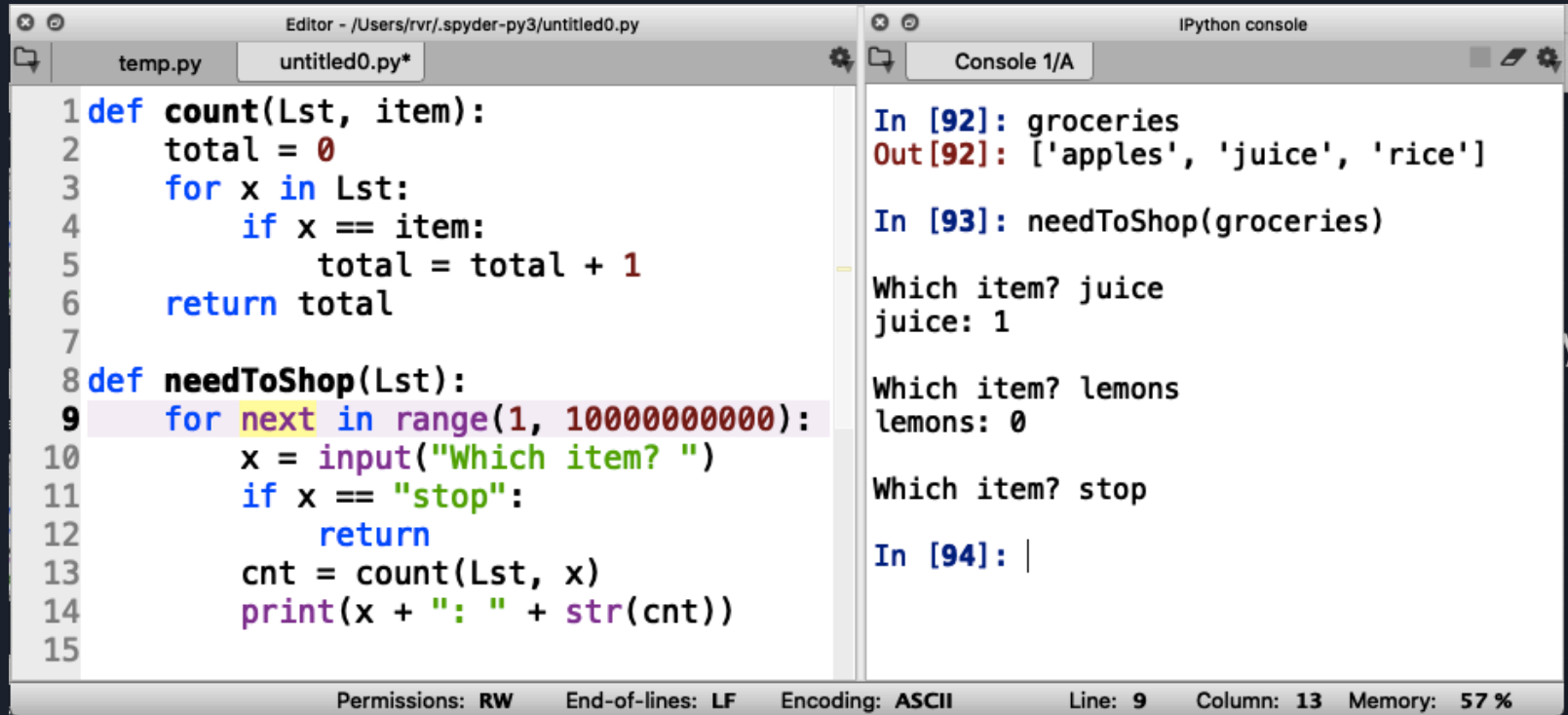
Permissions: RW End-of-lines: LF Encoding: ASCII Line: 9 Column: 13 Memory: 57 %

Long loop

Add a way to stop

Return with no value

# Try it out



The image shows a screenshot of a Python IDE with two windows: an editor and an IPython console. The editor window shows a Python script with two functions: `count` and `needToShop`. The `needToShop` function has a loop that runs 10 billion times, which is highlighted in yellow. The console window shows the execution of the code, including the input of 'groceries' and 'juice', and the output of the `count` function.

```
Editor - /Users/rvrr/.spyder-py3/untitled0.py
temp.py  untitled0.py*

1 def count(Lst, item):
2     total = 0
3     for x in Lst:
4         if x == item:
5             total = total + 1
6     return total
7
8 def needToShop(Lst):
9     for next in range(1, 10000000000):
10        x = input("Which item? ")
11        if x == "stop":
12            return
13        cnt = count(Lst, x)
14        print(x + ": " + str(cnt))
15

IPython console
Console 1/A

In [92]: groceries
Out[92]: ['apples', 'juice', 'rice']

In [93]: needToShop(groceries)

Which item? juice
juice: 1

Which item? lemons
lemons: 0

Which item? stop

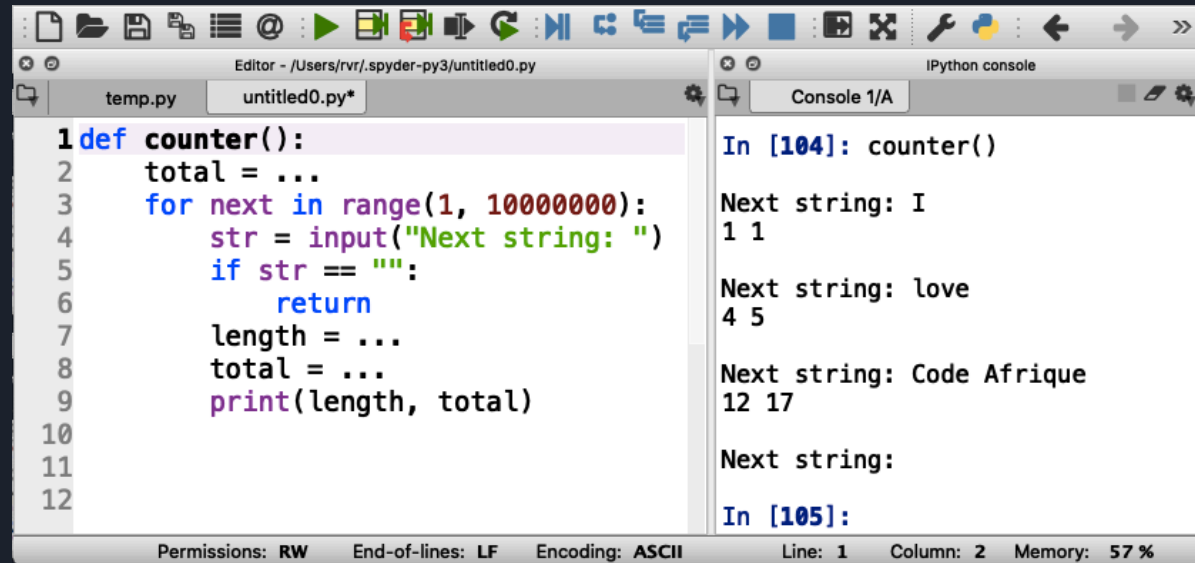
In [94]: |

Permissions: RW  End-of-lines: LF  Encoding: ASCII  Line: 9  Column: 13  Memory: 57 %
```

# Write your own function

Write a function that, in a loop, reads a string and prints the length of the string and the total length of all the strings that were input. Stop if the input is the empty string ("").

Recall that `len(str)` returns the length of string `str`



```
def counter():
    total = ...
    for next in range(1, 10000000):
        str = input("Next string: ")
        if str == "":
            return
        length = ...
        total = ...
    print(length, total)
```

```
In [104]: counter()
Next string: I
1 1

Next string: love
4 5

Next string: Code Afrique
12 17

Next string:

In [105]:
```

Permissions: RW End-of-lines: LF Encoding: ASCII Line: 1 Column: 2 Memory: 57 %



# What have we learned about input/output

- You can print values using the `print(value)` statement
- You can input values using the `input(prompt)` statement (where `prompt` is a string that is printed to ask the user for input)