



Exercise 38: Doing Things to Lists

You have learned about lists. When you learned about `while-loops` you "appended" numbers to the end of a list and printed them out. There were also Study Drills where you were supposed to find all the other things you can do to lists in the Python documentation. That was a while back, so review those topics if you do not know what I'm talking about.

Found it? Remember it? Good. When you did this you had a list, and you "called" the function `append` on it. However, you may not really understand what's going on so let's see what we can do to lists.

When you write `mystuff.append('hello')` you are actually setting off a chain of events inside Python to cause something to happen to the `mystuff` list. Here's how it works:

1. Python sees you mentioned `mystuff` and looks up that variable. It might have to look backward to see if you created with `=`, if it is a function argument, or if it's a global variable. Either way it has to find the `mystuff` first.
2. Once it finds `mystuff` it reads the `.` (period) operator and starts to look at *variables* that are a part of `mystuff`. Since `mystuff` is a list, it knows that `mystuff` has a bunch of functions.
3. It then hits `append` and compares the name to all the names that `mystuff` says it owns. If `append` is in there (it is) then Python grabs *that* to use.
4. Next Python sees the `()` (parenthesis) and realizes, "Oh hey, this should be a function." At this point it *calls* (runs, executes) the function just like normally, but instead it calls the function with an *extra* argument.
5. That *extra* argument is ... `mystuff`! I know, weird, right? But that's how Python works so it's best to just remember it and assume that's the result. What happens, at the end of all this, is a function call that looks like: `append(mystuff, 'hello')` instead of what you read which is `mystuff.append('hello')`.

For the most part you do not have to know that this is going on, but it helps when you get error messages from Python like this:

```
$ python
Python 2.6.5 (r265:79063, Apr 16 2010, 13:57:41)
[GCC 4.4.3] on linux2
Type "help", "copyright", "credits" or "license" for more
information.
>>> class Thing(object):
...     def test(hi):
...         print "hi"
...
>>> a = Thing()
>>> a.test("hello")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
```

1
2
3
4
MAIN



Follow

```
TypeError: test() takes exactly 1 argument (2 given)
>>>
```

What was all that? Well, this is me typing into the Python shell and showing you some magic. You haven't seen `class` yet but we'll get into that later. For now you see how Python said `test() takes exactly 1 argument (2 given)`. If you see this it means that Python changed `a.test("hello")` to `test(a, "hello")` and that somewhere someone messed up and didn't add the argument for `a`.

This might be a lot to take in, but we're going to spend a few exercises getting this concept firm in your brain. To kick things off, here's an exercise that mixes strings and lists for all kinds of fun.

```

1  ten_things = "Apples Oranges Crows Telephone Light Sugar"
2
3  print "Wait there are not 10 things in that list. Let's fix that."
4
5  stuff = ten_things.split(' ')
6  more_stuff = ["Day", "Night", "Song", "Frisbee", "Corn", "Banana", "Girl", "Boy"]
7
8  while len(stuff) != 10:
9      next_one = more_stuff.pop()
10     print "Adding: ", next_one
11     stuff.append(next_one)
12     print "There are %d items now." % len(stuff)
13
14 print "There we go: ", stuff
15
16 print "Let's do some things with stuff."
17
18 print stuff[1]
19 print stuff[-1] # whoa! fancy
20 print stuff.pop()
21 print ' '.join(stuff) # what? cool!
22 print '#'.join(stuff[3:5]) # super stellar!
```

What You Should See

```

$ python ex38.py
Wait there are not 10 things in that list. Let's fix that.
Adding: Boy
There are 7 items now.
Adding: Girl
There are 8 items now.
Adding: Banana
There are 9 items now.
Adding: Corn
There are 10 items now.
There we go: ['Apples', 'Oranges', 'Crows', 'Telephone', 'Light', 'Sugar', 'Boy', 'Girl', 'Banana', 'Corn']
Let's do some things with stuff.
Oranges
Corn
Corn
Apples Oranges Crows Telephone Light Sugar Boy Girl Banana
Telephone#Light
```

What Lists Can Do

Let's say you want to create a computer game based on Go Fish. If you don't know what Go Fish is, take the time now to go read up on it on the internet. To do this you would need to have some way of taking the concept of a "deck of cards" and put it into your Python program. You then have to write Python

code that knows how to work this imaginary version of a deck of cards so that a person playing your game thinks that it's real, even if it isn't. What you need is a "deck of cards" structure, and programmers call this kind of thing a "data structure".

What's a data structure? If you think about it, a "data structure" is just a formal way to *structure* (organize) some *data* (facts). It really is that simple, even though some data structures can get insanely complex, all they are is just a way to store facts inside a program so you can access them in different ways. They structure data.

I'll be getting into this more in the next exercise, but lists are one of the most common data structures programmers use. They are simply ordered lists of facts you want to store and access randomly or linearly by an index. What?! Remember what I said though, just because a programmer said "list is a list" doesn't mean that it's any more complex than what a list already is in the real world. Let's look at the deck of cards as an example of a list:

- 1. You have a bunch of cards with values.
- 2. Those cards are in a stack, list, or list from the top card to the bottom card.
- 3. You can take cards off the top, the bottom, the middle at random.
- 4. If you want to find a specific card, you have to grab the deck and go through it one at a time.

Let's look at what I said:

"An ordered list"

Yes, deck of cards is in order with a first, and a last.

"of things you want to store"

Yes, cards are things I want to store.

"and access randomly"

Yes, I can grab a card from anywhere in the deck.

"or linearly"

Yes, if I want to find a specific card I can start at the beginning and go in order.

"by an index"

Almost, since with a deck of cards if I told you to get the card at index 19 you'd have to count until you found that one. In our Python lists the computer can just jump right to any index you give it.

That is all a list does, and this should give you a way to figure out concepts in programming. Every concept in programming usually has some relationship to the real world. At least the useful ones do. If you can figure out what the analog in the real world is, then you can use that to figure out what the data structure should be able to do.

When to Use Lists

You use a list whenever you have something that matches the list data structure's useful features:

- 1. If you need to maintain order. Remember, this is listed order, not *sorted* order. Lists do not sort for you.
- 2. If you need to access the contents randomly by a number. Remember, this is using *cardinal* numbers starting at 0.
- 3. If you need to go through the contents linearly (first to last).

Remember, that's what `for-loops` are for.

Then that's when you use a list.

Study Drills

1. Take each function that is called, and go through the steps for function calls to translate them to what Python does. For example, `more_stuff.pop()` is `pop(more_stuff)`.
2. Translate these two ways to view the function calls in English. For example, `more_stuff.pop()` reads as, "Call `pop` on `more_stuff`." Meanwhile, `pop(more_stuff)` means, "Call `pop` with argument `more_stuff`." Understand how they are really the same thing.
3. Go read about "object-oriented programming" online. Confused? I was too. Do not worry. You will learn enough to be dangerous, and you can slowly learn more later.
4. Read up on what a "class" is in Python. *Do not read about how other languages use the word "class."* That will only mess you up.
5. Do not worry If you do not have any idea what I'm talking about. Programmers like to feel smart so they invented object-oriented programming, named it OOP, and then used it way too much. If you think that's hard, you should try to use "functional programming."
6. Find 10 examples of things in the real world that would fit in a list. Try writing some scripts to work with them.

Common Student Questions

Q: Didn't you say to not use `while-loops`?

Yes, so just remember sometimes you can break the rules if you have a good reason. Only idiots are slaves to rules all the time.

Q: Why does `join(' ', stuff)` not work?

The way the documentation for `join` is written doesn't make sense. It does not work like that and is instead a method you call on the *inserted* string to put between the list to be joined. Rewrite it like `' '.join(stuff)`.

Q: Why did you use a `while-loop`?

Try rewriting it with a `for-loop` and see if that's easier.

Q: What does `stuff[3:5]` do?

That extracts a "slice" from the `stuff` list that is from element 3 to element 4, meaning it does *not* include element 5. It's similar to how `range(3,5)` would work.

Video

Purchase The Videos For \$29.59

Buying Is Easy

For just \$29.59 you can get access to all the videos for Learn Python The

Buying is easy. Just fill out the form

Hard Way, **plus** a PDF of the book and no more popups all in this one location. For \$29.59 you get:

- All 52 videos, 1 per exercise, almost 2G of video.
- A PDF of the book.
- Email help from the author.
- See a list of everything you get before you buy.




When you buy the videos they will immediately show up **right here** without any hassles.


Already Paid? Reactivate Your Purchase Right Now!

below and we'll get started.

Full Name

Email Address

☒   

☐ 

Pay With Credit Card
(by Stripe™)

Use your PayPal™
account.

Buy Learn Python The Hard
Way, 3rd Edition

| | | | | |
|--|--|--|---|---|
| Zed Shaw PDF + Videos + Updates \$29.95 | Amazon Paper + DVD \$22.74 InformIT eBook + Paper \$43.19 | Amazon Kindle \$17.27 | B&N Paper + DVD \$22.96 | B&N Nook (No Video) \$17.27 |
| | | Interested In Ruby? Ruby is also a great language. Learn Ruby The Hard Way | | |