Beginner's Python Cheat Sheet - Lists

What are lists?

A list stores a series of items in a particular order. Lists allow you to store sets of information in one place, whether you have just a few items or millions of items. Lists are one of Python's most powerful features readily accessible to new programmers, and they tie together many important concepts in programming.

Defining a list

Use square brackets to define a list, and use commas to separate individual items in the list. Use plural names for lists, to make your code easier to read.

Making a list

```python
users = ['val', 'bob', 'mia', 'ron', 'ned']
```

Accessing elements

Individual elements in a list are accessed according to their position, called the index. The index of the first element is 0, the index of the second element is 1, and so forth. Negative indices refer to items at the end of the list. To get a particular element, write the name of the list and then the index of the element in square brackets.

Getting the first element

```python
first_user = users[0]
```

Getting the second element

```python
second_user = users[1]
```

Getting the last element

```python
newest_user = users[-1]
```

Modifying individual items

Once you've defined a list, you can change individual elements in the list. You do this by referring to the index of the item you want to modify.

Changing an element

```python
users[0] = 'valerie'
users[-2] = 'ronald'
```

Adding elements

You can add elements to the end of a list, or you can insert them wherever you like in a list.

Adding an element to the end of the list

```python
users.append('amy')
```

Starting with an empty list

```python
users = []
users.append('val')
users.append('bob')
users.append('mia')
```

Inserting elements at a particular position

```python
users.insert(0, 'joe')
users.insert(3, 'bea')
```

Removing elements

You can remove elements by their position in a list, or by the value of the item. If you remove an item by its value, Python removes only the first item that has that value.

Deleting an element by its position

```python
del users[-1]
```

Removing an item by its value

```python
users.remove('mia')
```

Popping elements

If you want to work with an element that you're removing from the list, you can "pop" the element. If you think of the list as a stack of items, pop() takes an item off the top of the stack. By default pop() returns the last element in the list, but you can also pop elements from any position in the list.

Pop the last item from a list

```python
most_recent_user = users.pop()
print(most_recent_user)
```

Pop the first item in a list

```python
first_user = users.pop(0)
print(first_user)
```

List length

The len() function returns the number of items in a list.

Find the length of a list

```python
num_users = len(users)
print("We have " + str(num_users) + " users.")
```

Sorting a list

The sort() method changes the order of a list permanently. The sorted() function returns a copy of the list, leaving the original list unchanged. You can sort the items in a list in alphabetical order, or reverse alphabetical order. You can also reverse the original order of the list. Keep in mind that lowercase and uppercase letters may affect the sort order.

Sorting a list permanently

```python
users.sort()
```

Sorting a list permanently in reverse alphabetical order

```python
users.sort(reverse=True)
```

Sorting a list temporarily

```python
print(sorted(users))
print(sorted(users, reverse=True))
```

Reversing the order of a list

```python
users.reverse()
```

Looping through a list

Lists can contain millions of items, so Python provides an efficient way to loop through all the items in a list. When you set up a loop, Python pulls each item from the list one at a time and stores it in a temporary variable, which you provide a name for. This name should be the singular version of the list name.

The indented block of code makes up the body of the loop, where you can work with each individual item. Any lines that are not indented run after the loop is completed.

Printing all items in a list

```python
for user in users:
    print(user)
```

Printing a message for each item, and a separate message afterwards

```python
for user in users:
    print("Welcome, " + user + "!")

print("Welcome, we're glad to see you all!")
```

Python Crash Course

Covers Python 3 and Python 2

nostarchpress.com/pythoncrashcourse
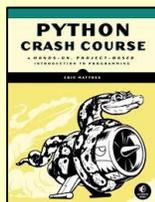
The range() function

You can use the range() function to work with a set of numbers efficiently. The range() function starts at 0 by default, and stops one number below the number passed to it. You can use the list() function to efficiently generate a large list of numbers.

Printing the numbers 0 to 1000

```python
for number in range(1001):
    print(number)
```

Printing the numbers 1 to 1000

```python
for number in range(1, 1001):
    print(number)
```

Making a list of numbers from 1 to a million

```python
numbers = list(range(1, 1000001))
```

Simple statistics

There are a number of simple statistics you can run on a list containing numerical data.

Finding the minimum value in a list

```python
ages = [93, 99, 66, 17, 85, 1, 35, 82, 2, 77]
youngest = min(ages)
```

Finding the maximum value

```python
ages = [93, 99, 66, 17, 85, 1, 35, 82, 2, 77]
oldest = max(ages)
```

Finding the sum of all values

```python
ages = [93, 99, 66, 17, 85, 1, 35, 82, 2, 77]
total_years = sum(ages)
```

Slicing a list

You can work with any set of elements from a list. A portion of a list is called a slice. To slice a list start with the index of the first item you want, then add a colon and the index after the last item you want. Leave off the first index to start at the beginning of the list, and leave off the last index to slice through the end of the list.

Getting the first three items

```python
finishers = ['kai', 'abe', 'ada', 'gus', 'zoe']
first_three = finishers[:3]
```

Getting the middle three items

```python
middle_three = finishers[1:4]
```

Getting the last three items

```python
last_three = finishers[-3:]
```

Copying a list

To copy a list make a slice that starts at the first item and ends at the last item. If you try to copy a list without using this approach, whatever you do to the copied list will affect the original list as well.

Making a copy of a list

```python
finishers = ['kai', 'abe', 'ada', 'gus', 'zoe']
copy_of_finishers = finishers[:]
```

List comprehensions

You can use a loop to generate a list based on a range of numbers or on another list. This is a common operation, so Python offers a more efficient way to do it. List comprehensions may look complicated at first; if so, use the for loop approach until you're ready to start using comprehensions.

To write a comprehension, define an expression for the values you want to store in the list. Then write a for loop to generate input values needed to make the list.

Using a loop to generate a list of square numbers

```python
squares = []
for x in range(1, 11):
    square = x**2
    squares.append(square)
```

Using a comprehension to generate a list of square numbers

```python
squares = [x**2 for x in range(1, 11)]
```

Using a loop to convert a list of names to upper case

```python
names = ['kai', 'abe', 'ada', 'gus', 'zoe']

upper_names = []
for name in names:
    upper_names.append(name.upper())
```

Using a comprehension to convert a list of names to upper case

```python
names = ['kai', 'abe', 'ada', 'gus', 'zoe']

upper_names = [name.upper() for name in names]
```

Styling your code
Readability counts

- Use four spaces per indentation level.
- Keep your lines to 79 characters or fewer.
- Use single blank lines to group parts of your program visually.

Tuples

A tuple is like a list, except you can't change the values in a tuple once it's defined. Tuples are good for storing information that shouldn't be changed throughout the life of a program. Tuples are designated by parentheses instead of square brackets. (You can overwrite an entire tuple, but you can't change the individual elements in a tuple.)

Defining a tuple

```python
dimensions = (800, 600)
```

Looping through a tuple

```python
for dimension in dimensions:
    print(dimension)
```

Overwriting a tuple

```python
dimensions = (800, 600)
print(dimensions)

dimensions = (1200, 900)
```

Visualizing your code

When you're first learning about data structures such as lists, it helps to visualize how Python is working with the information in your program. pythontutor.com is a great tool for seeing how Python keeps track of the information in a list. Try running the following code on pythontutor.com, and then run your own code.

Build a list and print the items in the list

```python
dogs = []
dogs.append('willie')
dogs.append('hootz')
dogs.append('peso')
dogs.append('goblin')

for dog in dogs:
    print("Hello " + dog + "!")
print("I love these dogs!")

print("\nThese were my first two dogs:")
old_dogs = dogs[:2]
for old_dog in old_dogs:
    print(old_dog)

del dogs[0]
dogs.remove('peso')
print(dogs)
```