

Lab 4 GRS

Lists and loops

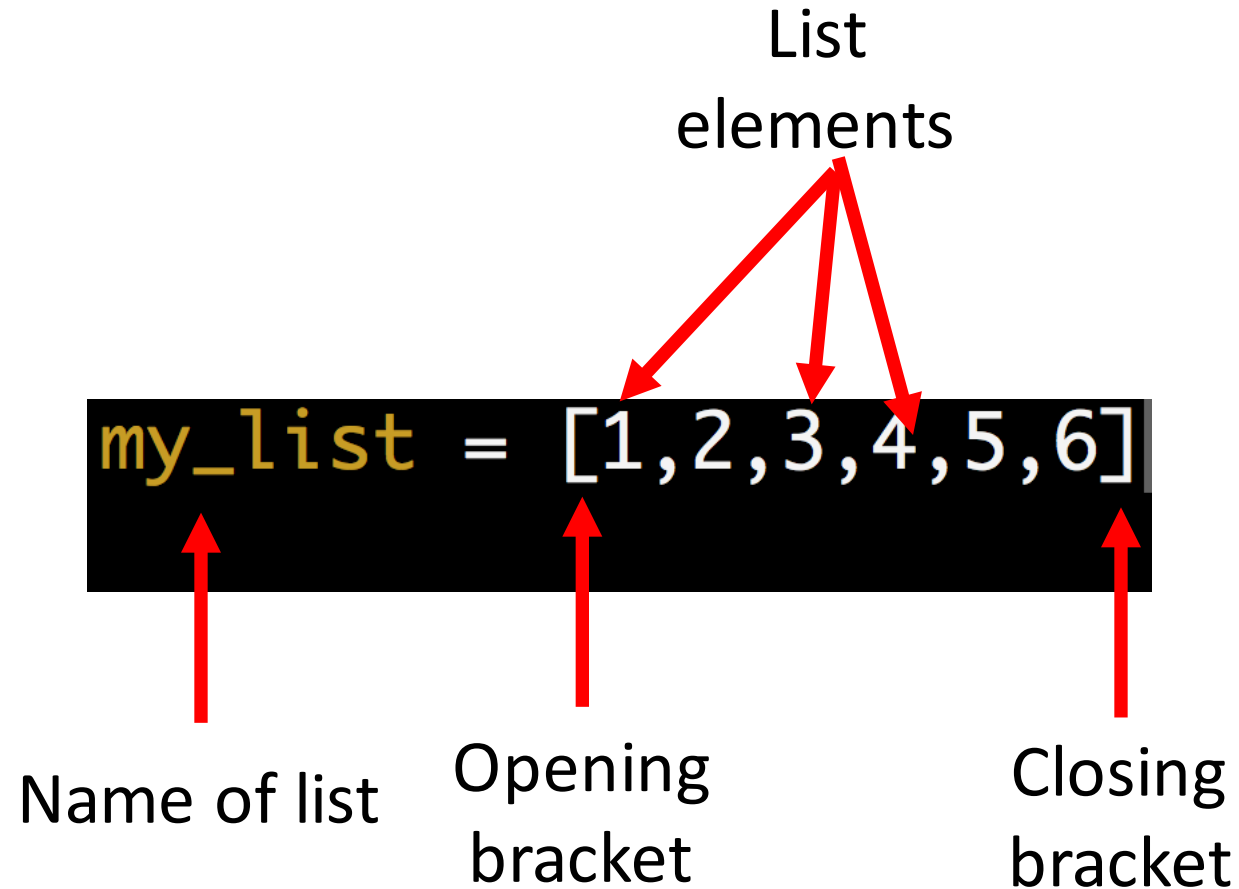
What are lists?

- Lists are a collection of objects
 - These objects can be of different data types in Python
 - **Resizable** and **dynamic**
 - This usually isn't the case for other programming languages
- Lists are a “special” data type called an **object/reference**

How do we create lists?

- Multiple ways to do so

Square bracket notation



Creating an empty list

```
my_empty_list = []
```



Set of empty square
brackets

Creating an empty list

```
another_empty_list = list()
```



Using the **list**
function

Lists can have elements of different types

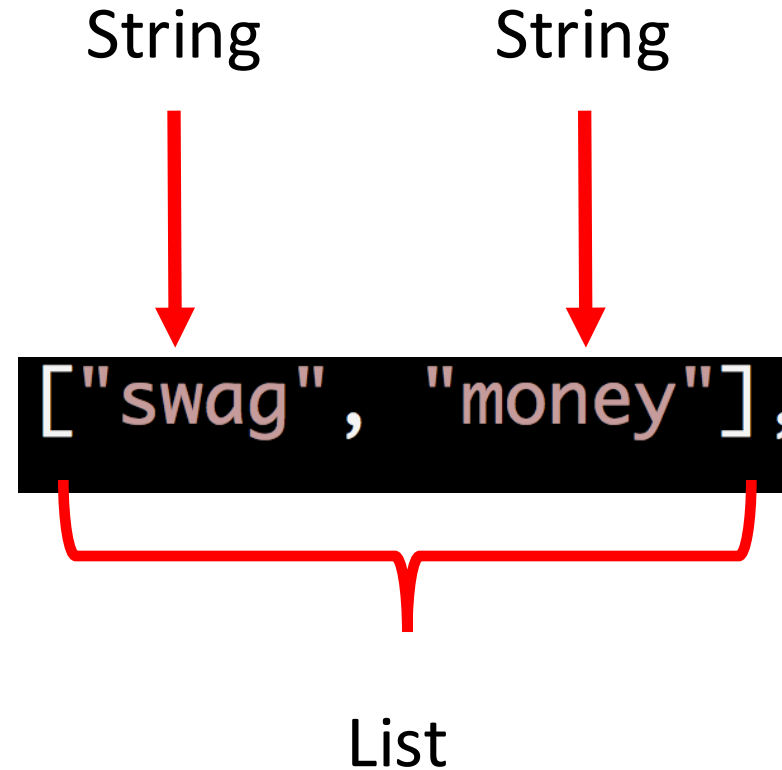
The diagram illustrates a Python list named `my_list` containing elements of different types. The list is defined as `my_list = ["cool", ["swag", "money"], 12345, 33.0]`. Red arrows point from each element to its corresponding data type label:

- The string `"cool"` is labeled `String`.
- The nested list `["swag", "money"]` is labeled `List`.
- The integer `12345` is labeled `int`.
- The float `33.0` is labeled `float`.

Additionally, two red arrows point from the labels `String` to the elements `"swag"` and `"money"` inside the nested list, indicating that these elements are also of type `String`.

Lists can have elements of different types

- Including other lists



Accessing elements

```
my_list = ["cool", "swag", "money", "hello", "eggs"]
```



element 0 element 1 element 2 element 3 element 4

The diagram illustrates the mapping between list elements and their indices. Five red arrows point upwards from the labels 'element 0' through 'element 4' to the corresponding elements in the list: 'cool', 'swag', 'money', 'hello', and 'eggs'.

Accessing elements

```
my_list = ["cool", "swag", "money", "hello", "eggs"]
```

my_list[0] my_list[1] my_list[2] my_list[3] my_list[4]

A diagram illustrating list access. Five red arrows point upwards from the indices my_list[0] through my_list[4] to the corresponding elements in the list: "cool", "swag", "money", "hello", and "eggs".

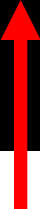
Modifying a list

- Bracket syntax

```
my_list[1] = ["avocado"]
```



index



Object you want
to put at that
index

Before


```
my_list = ["cool", "swag", "money", "hello", "eggs"]  
my_list[1] = ["avocado"]
```

After

```
["cool", "avocado", "money", "hello", "eggs"]
```

Accessing the length of a list

```
len(my_list)
```



Use the **len**
function




A list


Modifying a list

- append

```
my_list.append("lol")
```



Use the
append
function



Object that will be
appended to the
end of the list

Fun facts about lists

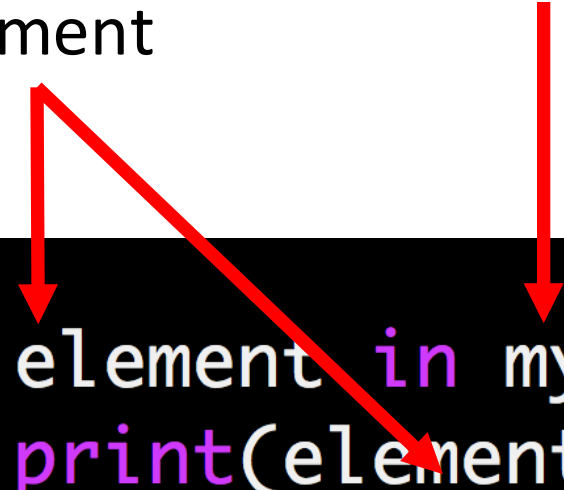
- **These are true for every nonempty list**
- Last element index: length of list $- 1$
- First element index: 0
- Middle element: length of list $/ 2$
- The length of a list is **one past** the last index

Looping through a list

- A **loop** can be used to do a task 0+ times
- **For-each loops** are most often used to **iterate** through a list
- Syntax:

Current
element

A list

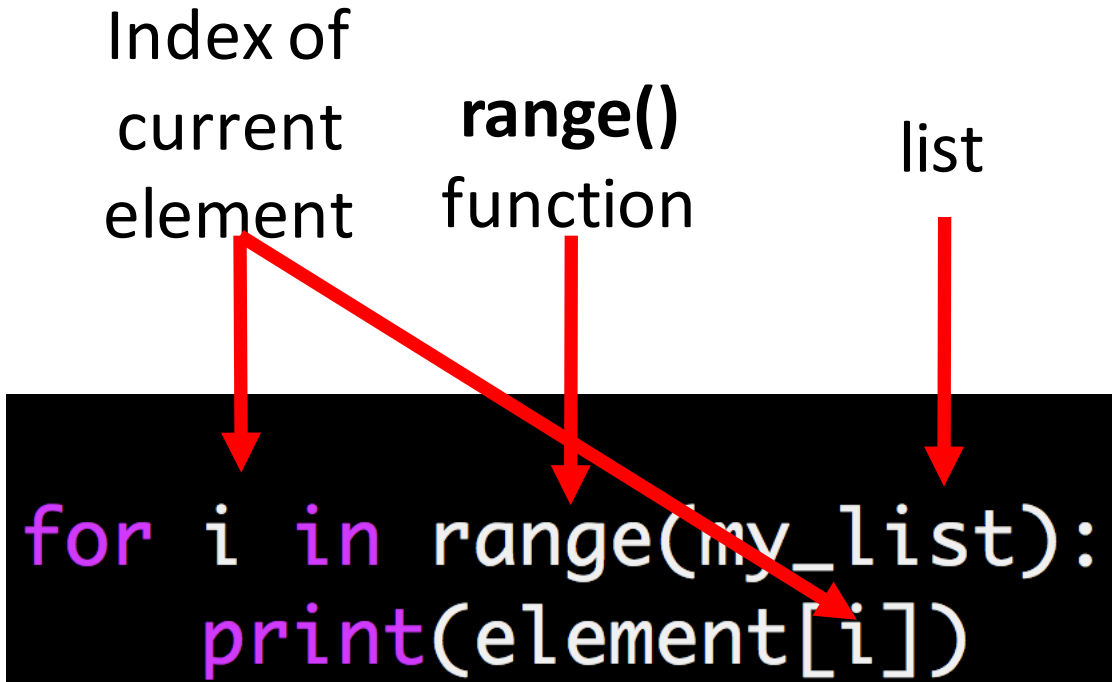


```
for element in my_list:  
    print(element)
```

- For-each loops can't be used to modify lists!!!!!!

For loops

Index of
current
element **range()**
function list



The diagram illustrates the components of a Python for loop. Three labels at the top point to parts of the code below. A red arrow points from 'Index of current element' to the variable 'i'. Another red arrow points from 'range()' function' to the 'range()' function call. A third red arrow points from 'list' to the variable 'my_list'. Additionally, a red arrow points from the 'i' in 'element[i]' to the 'i' in the range function, indicating that the loop variable is used as an index into the list.

```
for i in range(my_list):  
    print(element[i])
```

Activity

- Copy the activity file into your lab4 grs directory

`cp /afs/umbc.edu/users/a/g/agatha3/pub/201_grs/lab4/lab4grs.py .`

- Don't forget the period!!!!