Course	
Term	
Week	
Date	
Chapter. Topic	7. Lists and Tuples

Lists

List Unpacking and Slicing

Siva R Jasthi

Computer Science and Cybersecurity

Metropolitan State University

Lists

List is a collection which is **ordered** and **changeable**.
Allows **duplicate** members.

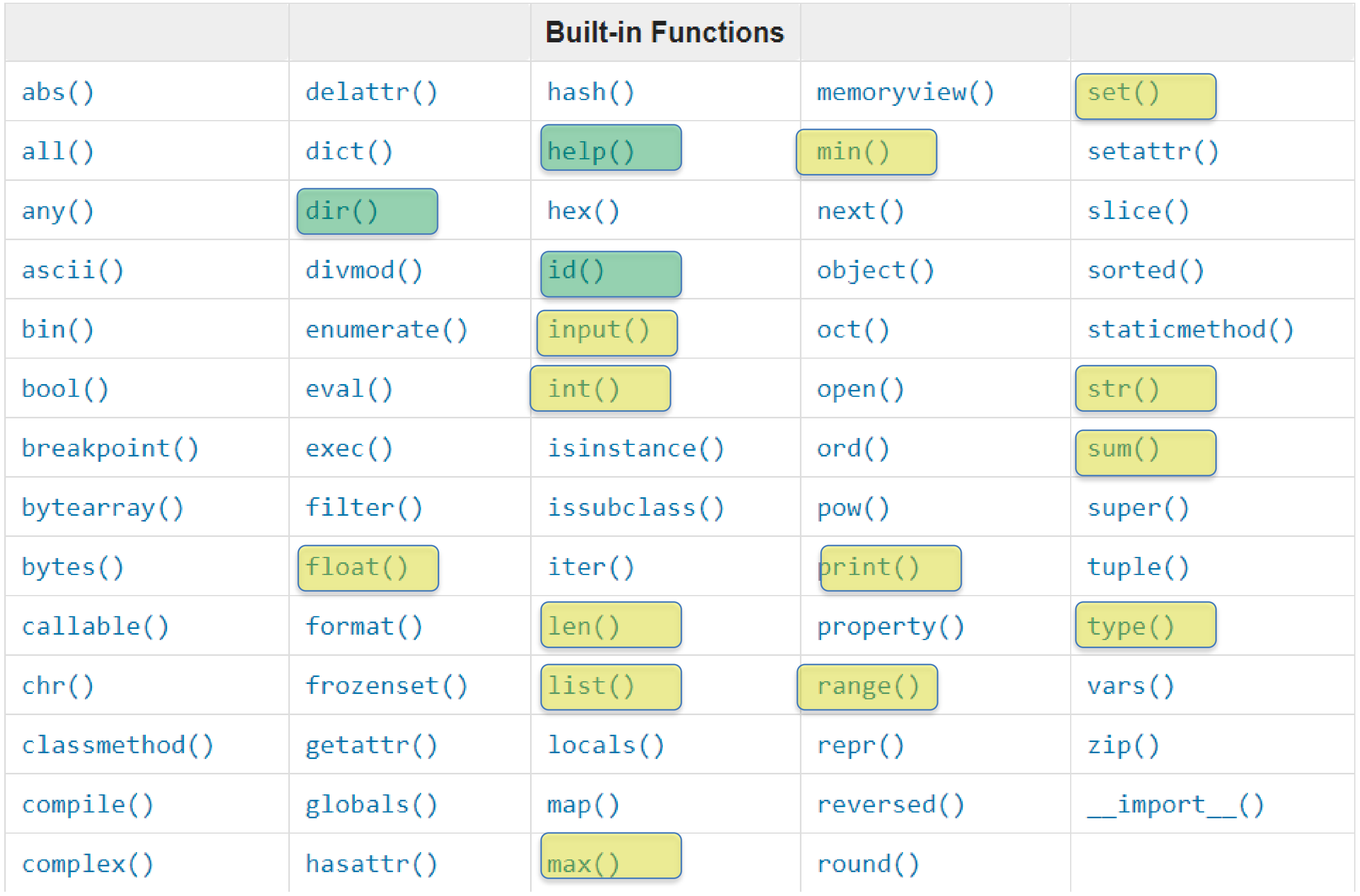
Lists: An introduction
https://www.w3schools.com/python/python_lists.asp

Lists: An introduction
https://openbookproject.net/thinkcs/python/english3e/lists.html

List Methods
http://www.python-ds.com/python-3-list-methods

Built-in Functions
https://docs.python.org/3/library/functions.html

Python's Built-in Functions



		Built-in Functions		
abs()	delattr()	hash()	memoryview()	set()
all()	dict()	help()	min()	setattr()
any()	dir()	hex()	next()	slice()
ascii()	divmod()	id()	object()	sorted()
bin()	enumerate()	input()	oct()	staticmethod()
bool()	eval()	int()	open()	str()
breakpoint()	exec()	isinstance()	ord()	sum()
bytearray()	filter()	issubclass()	pow()	super()
bytes()	float()	iter()	print()	tuple()
callable()	format()	len()	property()	type()
chr()	frozenset()	list()	range()	vars()
classmethod()	getattr()	locals()	repr()	zip()
compile()	globals()	map()	reversed()	__import__()
complex()	hasattr()	max()	round()	

Some functions are valid for lists.

I highlighted some.

Can you find other functions that are valid on lists?

https://docs.python.org/3/library/functions.html

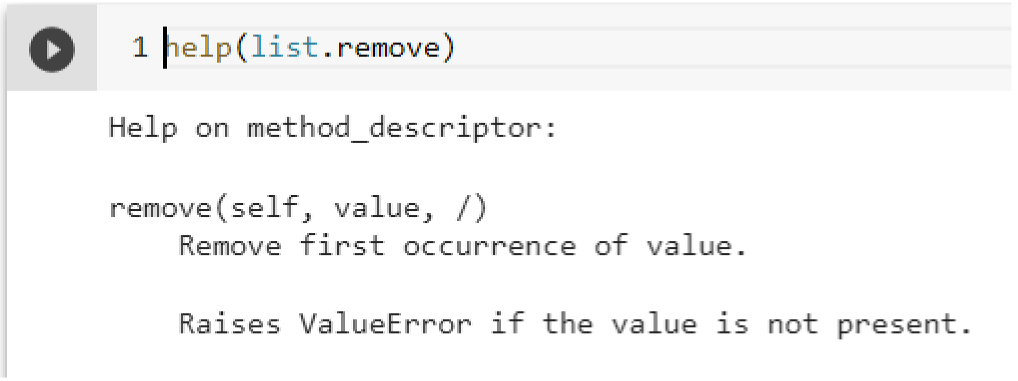
Asking python for help!

- What methods are available?

`dir(list)`
`help(list)`

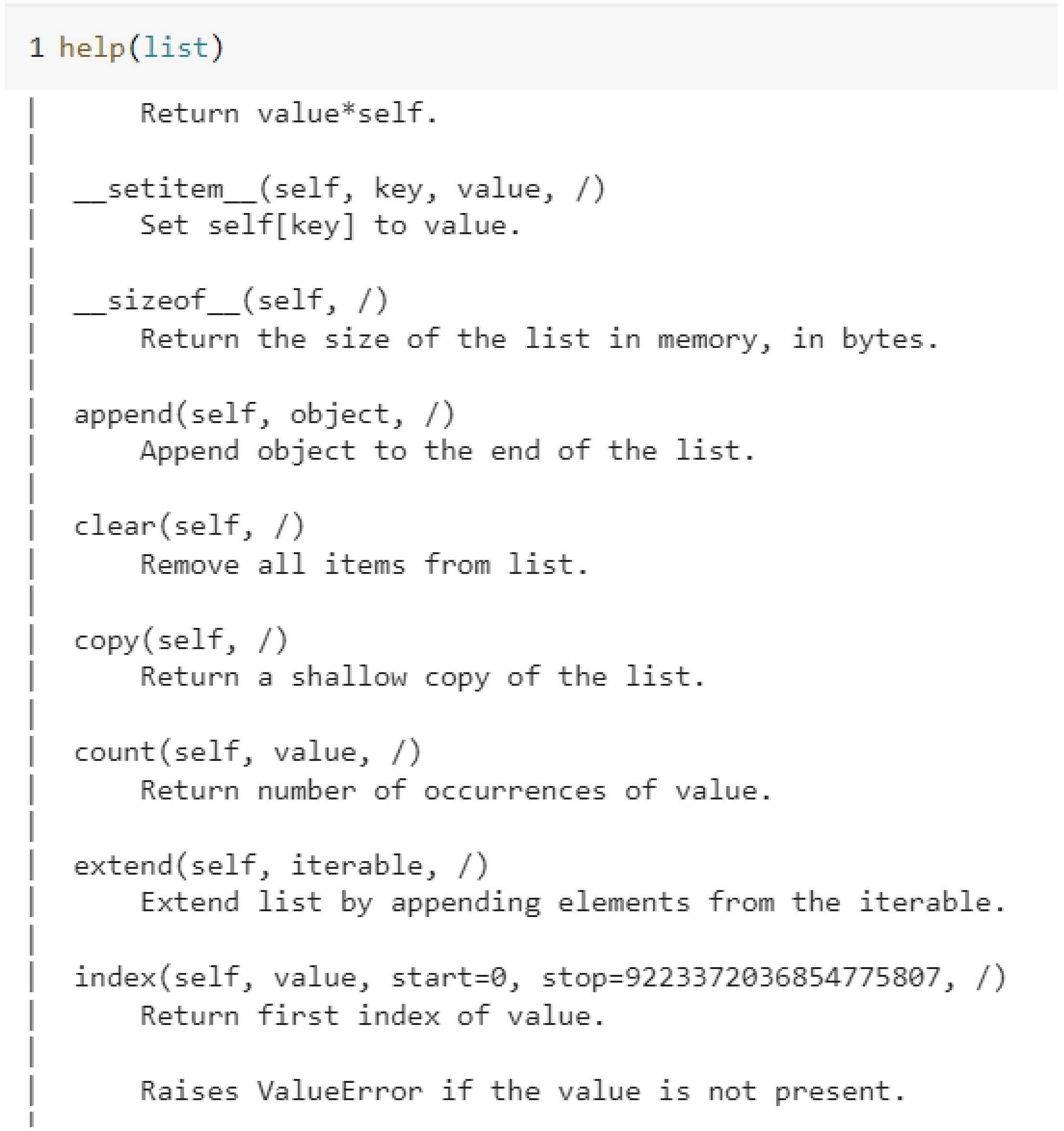
- Can you tell me more about a particular method?

`help(list.remove)`



```
help(list.remove)
```

```
Help on method_descriptor:

remove(self, value, /)
    Remove first occurrence of value.

    Raises ValueError if the value is not present.
```



```
help(list)
```

```
|        Return value*self.
|
|    __setitem__(self, key, value, /)
|        Set self[key] to value.
|
|    __sizeof__(self, /)
|        Return the size of the list in memory, in bytes.
|
|    append(self, object, /)
|        Append object to the end of the list.
|
|    clear(self, /)
|        Remove all items from list.
|
|    copy(self, /)
|        Return a shallow copy of the list.
|
|    count(self, value, /)
|        Return number of occurrences of value.
|
|    extend(self, iterable, /)
|        Extend list by appending elements from the iterable.
|
|    index(self, value, start=0, stop=9223372036854775807, /)
|        Return first index of value.
|
|        Raises ValueError if the value is not present.
|
```

List Methods list.method_name(params)

Method	Purpose
append(x)	Add x to the end of the list
extend(list_x)	Add all items from list_x at the end of the list
insert(i,x)	Inserts an item at a given position. The first argument is the index of the element before which to insert. For example, a.insert(0, x) inserts at the front of the list.
remove(x)	Removes the first item x (note: there can be multiple items x in the list)
pop()	Removes the last item and returns the item
pop([i]	Removes the first item
clear( )	Removed all elements in the list. Empties the list.
index(x)	Returns the index of the first item x.
count(x)	Counts the number of times x is appearing in the list
sort( )	Sorts the elements in ascending order. sort(reverse=True) sorts the elements in descending order
reverse( )	Reverses a list
copy( )	Returns a copy of the list. You can also use "list" built-in function for the same purpose.

http://www.python-ds.com/python-3-list-methods

List Unpacking

List unpacking offers a shorter syntax when you want to assign list values to different variables in a single line.

List Unpacking

```
colors = ['red', 'blue', 'green’]

a = colors[0]

b = colors[1]

c = colors[2]
```

Can be written as

```
colors = ['red', 'blue', 'green’]

a, b, c = colors
```

List Unpacking helps to assign list values to multiple variables at the same time in one single-line statement

https://www.pythontutorial.net/python-basics/python-unpack-list/

List Unpacking: counts must match

```
colors = ['red', 'blue', 'green’, ‘yellow’]

red, blue, green, yellow = colors
```

Right

```
colors = ['red', 'blue', 'green’]

red, blue = colors
```

Wrong

The number of variables on the left side is the same as the number of elements in the list on the right side.

If you use a fewer number of variables on the left side, you’ll get an error.

```
ValueError: too many values to unpack (expected 2)
```

List Unpacking: left-overs

```
colors = ['red', 'blue', 'green’, ‘yellow’]

red, blue, green, yellow = colors
```

Right

```
colors = ['red', 'blue', 'green’]

red, blue, *rest = colors
```

Right

f you want to unpack the first few elements of a list and don’t care about the other elements, you can:

- First, unpack the needed elements to variables.
- Second, pack the leftover elements into a new list and assign it to another variable.
- By putting the asterisk (*) in front of a variable name, you’ll pack the leftover elements into a list and assign it to a variable.

Unpacking the lists in print function (ex.)

We can also unpack the lists during the print.

```
colors = ['red', 'blue', 'green’, ‘yellow’]
```

Analyze the output produced by these two inputs.

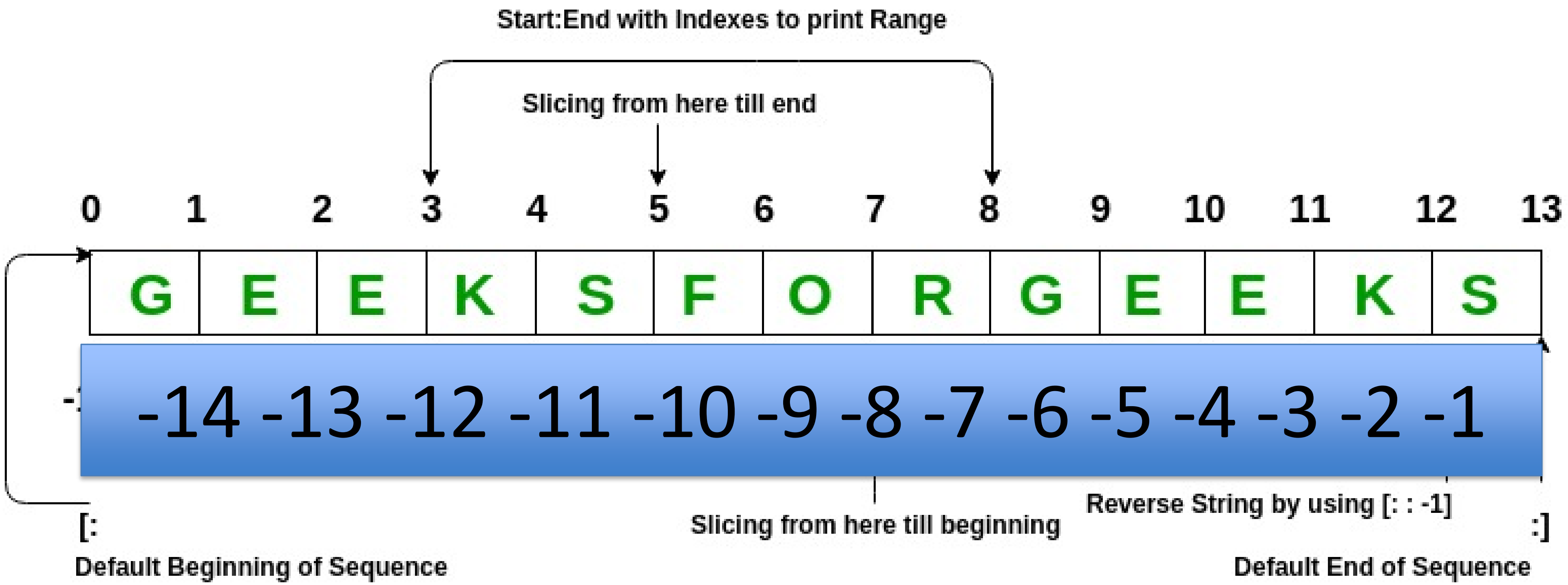
```
# prints   ['red', 'blue', 'green', 'yellow’]
print(colors)
```

```
#prints red blue green yellow
print(*colors)
```

Unpacking Summary

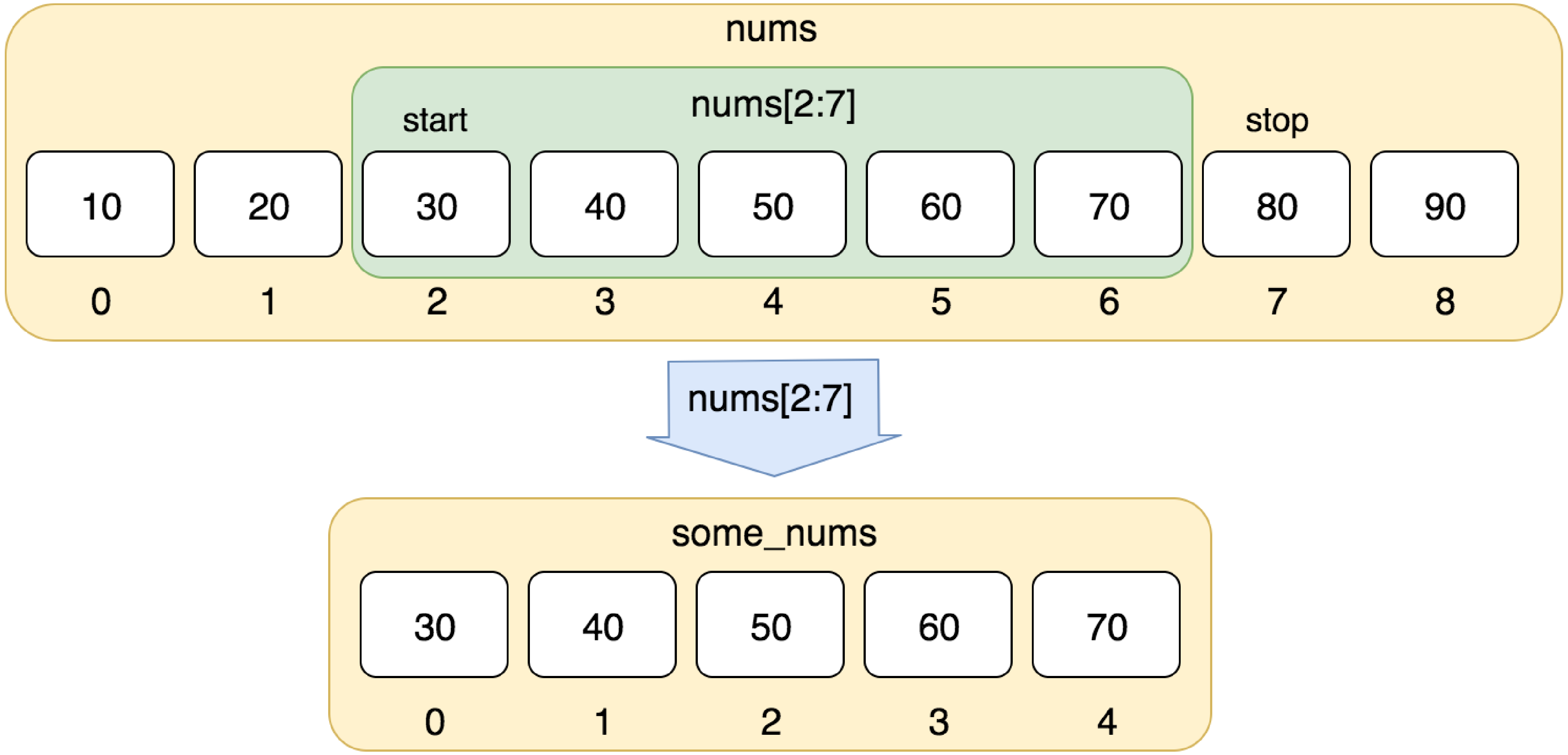
1. Short hand notation to assign list values to variables
2. Counts for unpacking must match
3. Common Error: Too many values to unpack
4. Common Error: Not enough values to unpack
5. You can prefix * to a variable to assign a sub-list

Negative indexes are cool



Start:End with Indexes to print Range
Slicing from here till end
0 1 2 3 4 5 6 7 8 9 10 11 12 13
G E E K S F O R G E E K S
-14 -13 -12 -11 -10 -9 -8 -7 -6 -5 -4 -3 -2 -1
Reverse String by using [: : -1]
[:
Slicing from here till beginning
:]
Default Beginning of Sequence
Default End of Sequence

List Slicing: A typical operation on lists



Slicing syntax (just like RANGE function)

```
L[start:stop:step]
```

Start position   *End position*   *The increment*

Note 1: *Just like in range function*, the second (stop) number is "up to but not including"

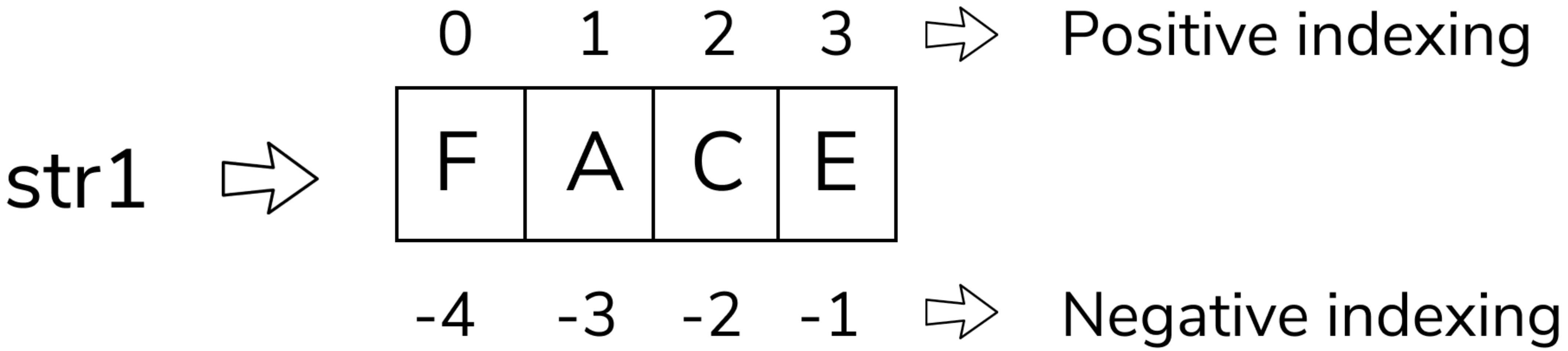
Note 2: *If you omit "start", slicing starts from index 0*
If you omit "stop", slicing ends at the last item.
If you omit "step", the increment will be 1

Lists can be sliced using colon :

```
>>> t = [9, 41, 12, 3, 74, 15]
>>> t[1:3]
[41,12]
>>> t[:4]
[9, 41, 12, 3]
>>> t[3:]
[3, 74, 15]
>>> t[:]
[9, 41, 12, 3, 74, 15]
```

Remember: *Just like in strings*, the second number is "**up to but not including**"

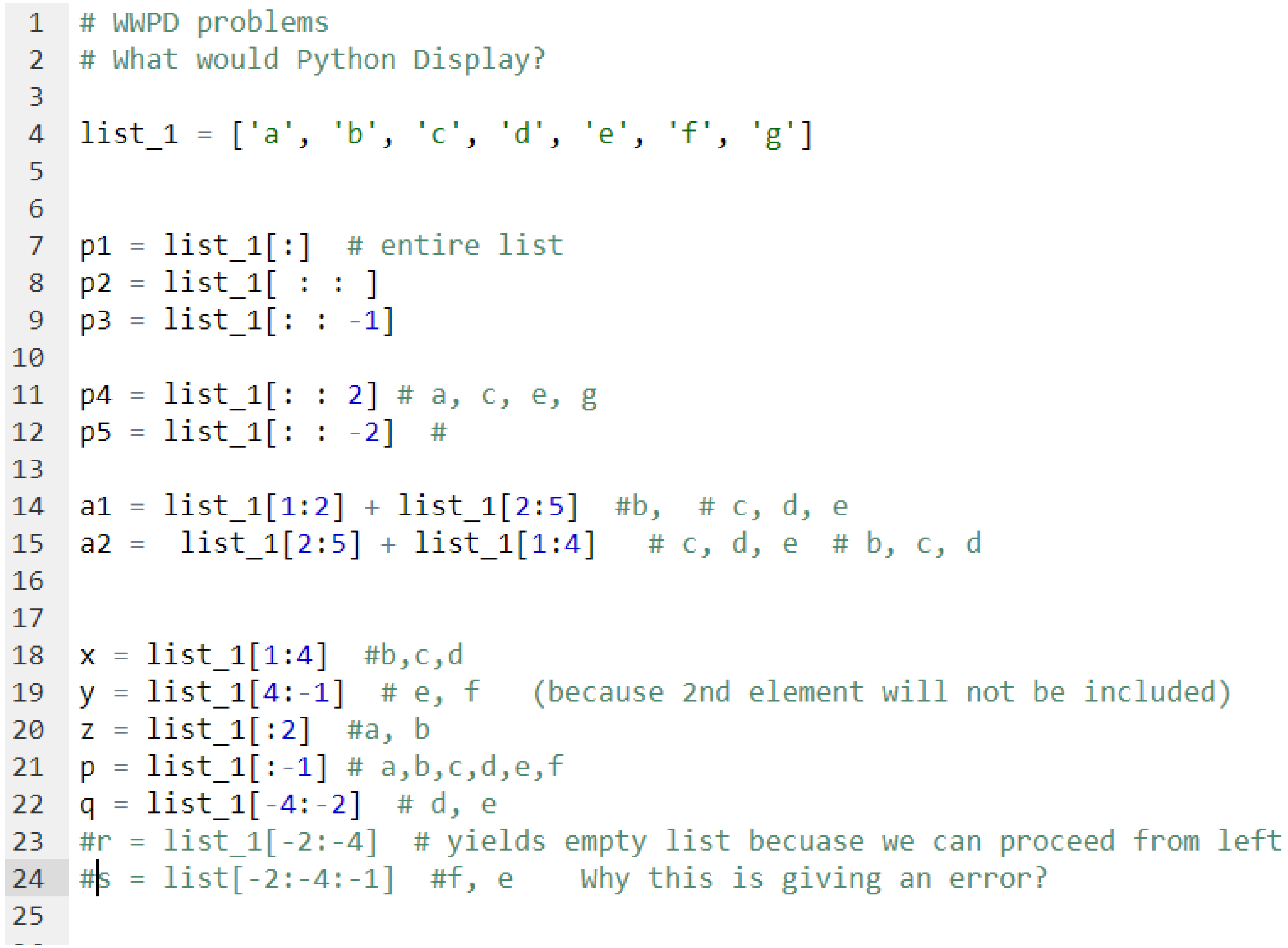
Strings can be sliced too



str1[1:3] = AC

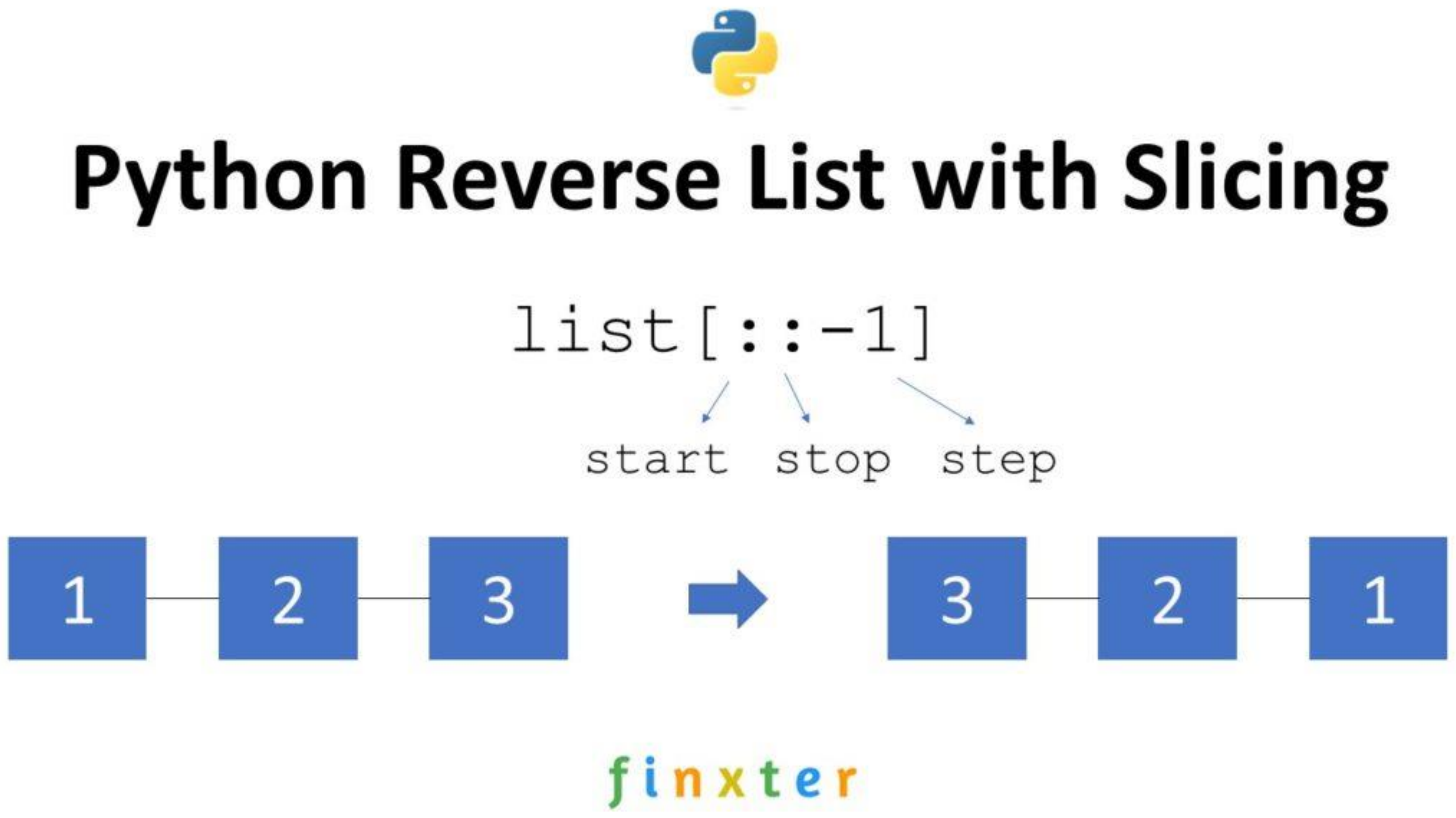
str1[-3:-1] = AC

WWPD problems (link)



```
# WWPD problems
# What would Python Display?

list_1 = ['a', 'b', 'c', 'd', 'e', 'f', 'g']


p1 = list_1[:]  # entire list
p2 = list_1[ : : ]
p3 = list_1[: : -1]

p4 = list_1[: : 2] # a, c, e, g
p5 = list_1[: : -2]  #

a1 = list_1[1:2] + list_1[2:5]  #b,  # c, d, e
a2 =  list_1[2:5] + list_1[1:4]   # c, d, e  # b, c, d


x = list_1[1:4]  #b,c,d
y = list_1[4:-1]  # e, f   (because 2nd element will not be included)
z = list_1[:2]  #a, b
p = list_1[:-1] # a,b,c,d,e,f
q = list_1[-4:-2]  # d, e
#r = list_1[-2:-4]  # yields empty list becuase we can proceed from left
#s = list[-2:-4:-1]  #f, e    Why this is giving an error?

```

Reversing a List with Slicing

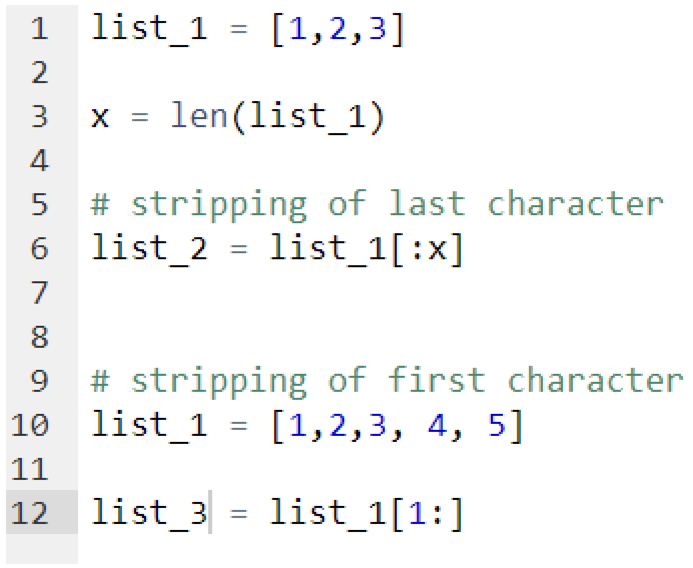


```
list[::-1]
```

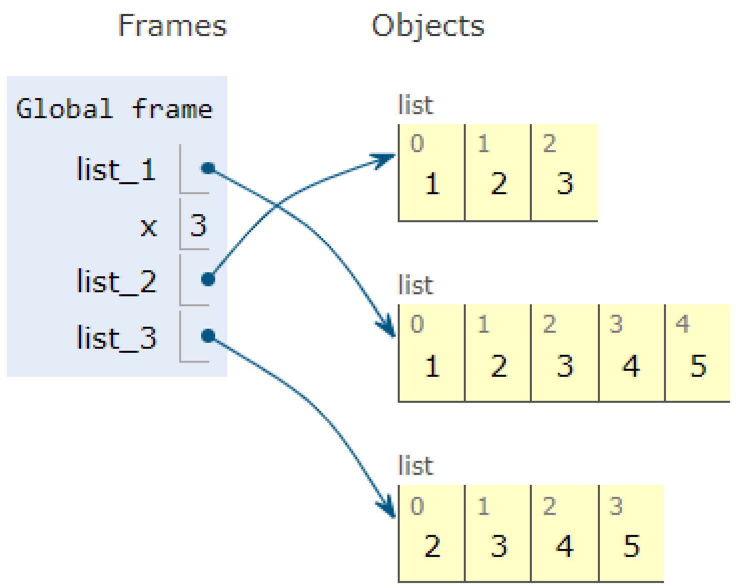
Stripping the last character

list_1 = [1,2,3]

list_1 = list_1[:len(list_1)-1]

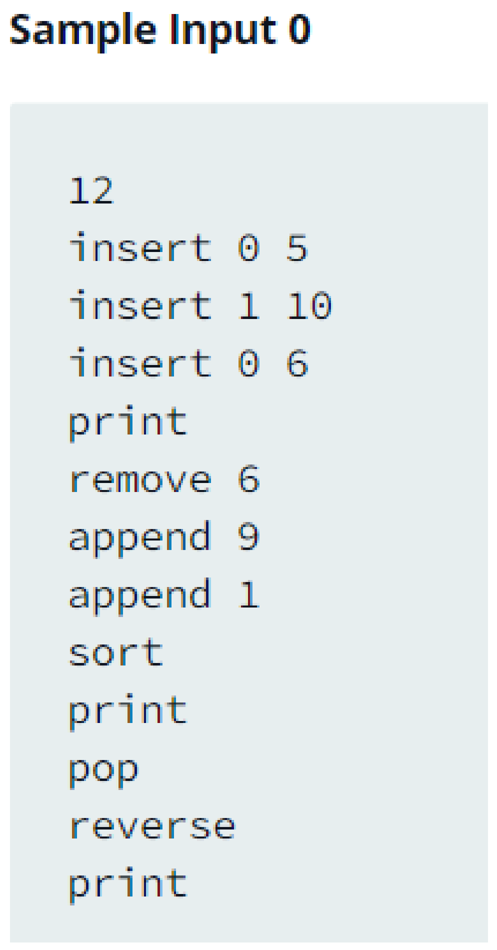


```
list_1 = [1,2,3]

x = len(list_1)

# stripping of last character
list_2 = list_1[:x]


# stripping of first character
list_1 = [1,2,3, 4, 5]

list_3 = list_1[1:]
```



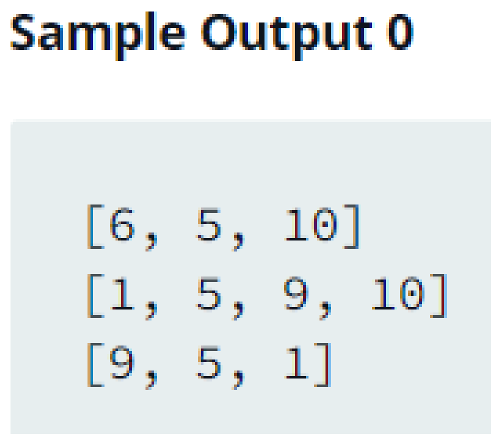
List Operations: Try out this HackerRank Problem.

We will solve this HackerRank problem.
https://www.hackerrank.com/challenges/python-lists/problem?isFullScreen=true



Sample Input 0

```
12
insert 0 5
insert 1 10
insert 0 6
print
remove 6
append 9
append 1
sort
print
pop
reverse
print
```



Sample Output 0

```
[6, 5, 10]
[1, 5, 9, 10]
[9, 5, 1]
```

My_list = []

[5]
[5,10]
[6,5,10]

[5,10]
[5,10,9]
[5,10,9,1]
[1,5,9,10]
[1,5,9]
[9,5,1]

List Slicing: some common algorithms

Strip the last character (new line character)

String the first character

Reverse a String

Reverse a List