Global Variables + Introduction to Lists

Global Variables + Constants

Global Variables

- A global variable can be accessed and modified from any part of a program's code
- Usually declared *outside* of function definitions (but can be used in a function call!)
- Exist in Globals frame of memory diagram

Constants

Constants are variables that don’t change throughout the process of your program.

They are denoted in all capital letters (e.g. **`WHITE_BOX`** in Wordle)

Memory Diagram

```
1   """Practicing global variables and constants."""
2
3   MAX_TACOS: int = 10
4
5   def order(num_tacos: int) -> bool:
6       if num_tacos > MAX_TACOS:
7           return False
8       else:
9           return True
10
11
12  order(num_tacos=11)
```

Lists

Lists

A list is a **data structure**–something that lets you reason about multiple items.

Examples of lists:

- To-do list
- Assignment Due Dates
- Grocery List

***Lists can be an arbitrary length! (Not a fixed number of items.)*

Declaring the type of a list

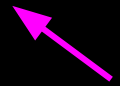
```
<list name>: list[<item type>]

grocery_list: list[str]
```

Declaring the type of a list

```
<list name>: list[<item type>]

grocery_list: list[str]
```



str, int, float, etc.

Initializing an empty list

With a constructor:

- `<list name>: list[<item type>] = list()`
- `grocery_list: list[str] = list()`

With a literal:

- `<list name>: list[<item type>] = []`
- `grocery_list: list[str] = []`

Initializing an empty list

The constructor `list()` is a *function* that returns the literal `[]`

With a constructor:

- `<list name>: list[<item type>] = list()`
- `grocery_list: list[str] = list()`

With a literal:

- `<list name>: list[<item type>] = []`
- `grocery_list: list[str] = []`

Initializing an empty list

With a constructor:

- `<list name>: list[<item type>] = list()`
- `grocery_list: list[str] = list()`

With a literal:

- `<list name>: list[<item type>] = []`
- `grocery_list: list[str] = []`

The constructor `list()` is a *function* that returns the literal `[]`

Bringing it back to something we know, you can create an empty string using the constructor `str()` or the literal `""`

Initializing an empty list

The constructor `list()` is a *function* that returns the literal `[]`

With a constructor:

- `<list name>: list[<item type>] = list()`
- `grocery_list: list[str] = list()`

Bringing it back to something we know, you can create an empty string using the constructor `str()` or the literal `""`

With a literal:

- `<list name>: list[<item type>] = []`
- `grocery_list: list[str] = []`

Let's try it!

Create an empty list of floats with the name my_numbers.

Adding an item to a list

```
<list name>.append(<item>)

grocery_list.append(“bananas”)
```

Adding an item to a list

```
<list name>.append(<item>)

grocery_list.append("bananas")
```

- Method: a function that *belongs* to the **list** class
- Like calling append(grocery_list, “bananas”)

Adding an item to a list

```
<list name>.append(<item>)

grocery_list.append("bananas")
```

- Method: a function that *belongs* to the **list** class
- Like calling append(grocery_list, “bananas”)

Let’s try it!

Add the value 1.5 to my_numbers.

Initializing An Already Populated List

```
<list name>: list[<item type>]  = [<item 0>, <item 1>, …]

grocery_list: list[str] = [“bananas”, “milk”, “bread”]
```

Initializing An Already Populated List

```
<list name>: list[<item type>]  = [<item 0>, <item 1>, …]

grocery_list: list[str] = [“bananas”, “milk”, “bread”]
```

Let’s try it!

Create a list called game_points that stores the following numbers: 102, 86, 94

Indexing

```
grocery_list: list[str] = [“bananas”, “milk”, “bread”]

grocery_list[0]
```

***Starts at 0, like with strings!*

Indexing

```
grocery_list: list[str] = [“bananas”, “milk”, “bread”]

grocery_list[0]
```

***Starts at 0, like with strings!*

Let’s try it!
In game_points, use subscription notation to print out 94.

Modifying by Index

```
grocery_list: list[str] = [“bananas”, “milk”, “bread”]

grocery_list[1] = “eggs”
```

Modifying by Index

```
grocery_list: list[str] = [“bananas”, “milk”, “bread”]

grocery_list[1] = “eggs”
```

Let’s try it!
In game_points, use subscription notation to change 86 to 72.

Modifying by Index

```
grocery_list: list[str] = [“bananas”, “milk”, “bread”]

grocery_list[1] = “eggs”
```

Let’s try it!
In game_points, use subscription notation to change 86 to 72.

Question: Could you do this type of modification with a string? Try it out!

Length of a List

```
grocery_list: list[str] = [“eggs”, “milk”, “bread”]

len(grocery_list)
```

Length of a List

```
grocery_list: list[str] = [“eggs”, “milk”, “bread”]

len(grocery_list)
```

Let’s try it!
Print the length of game_points.

Remove an Item From a List

```
grocery_list: list[str] = [“eggs”, “milk”, “bread”]

grocery_list.pop(2)
```

Index of item you want to remove

Remove an Item From a List

```
grocery_list: list[str] = [“eggs”, “milk”, “bread”]

grocery_list.pop(2)
```

Index of item you want to remove

Let’s try it!
Remove 72 from game_points.

Lists in Memory: Comparing Lists and Strings

```
1   a: str = "24"
2   b: str = a
3   a += "6"
4   print(b)
```

```
1   a: list[int] = [2,4]
2   b: list[int] = a
3   a.append(6)
4   print(b)
```

Lists + Functions

Functions can:

- Take lists as arguments
- Return or create lists
- *Modify* lists!

Taking a List as an Argument

```
1  def display(vals: list[int]) -> None:
2      idx: int = 0
3      while idx < len(vals):
4          print(vals[idx])
5          idx += 1
6
7  display([1,2,3])
```

Creating + Returning a List

```
1  def odds_list(min: int, max: int) -> list[int]:
2      """returns list of odds between min and max"""
3      odds: list[int] = list()
4      x: int = min
5      while x <= max:
6          if x % 2 == 1:
7              odds.append(x)
8          x += 1
9      return odds
10
11 global_odds: list[int] = odds_list(2,10)
12 print(global_odds)
```

Modifying a List

```
1  def remove_first(xs: list[str]):
2      xs.pop(0)
3
4  course: list[str] = ["Comp", "110"]
5  remove_first(course)
```

Coding Example (if we have time)

- Let's implement a function named contains where we can call with 2 arguments:
  - A needle: int value we are searching for
  - A haystack: list[int] of values we are searching in
- The return value of the function should be True if needle appears in haystack at least once and False otherwise

Recap…

- **Global variables**
  - Same syntax as local variables, but defined outside function body
  - Can be accessed anywhere in program (e.g. inside function calls)
- **Constant Variables**
  - Variables that don't change throughout the process of your program
  - Common syntax convention: All-caps
- **Lists**
  - A list is a **data structure**–something that lets you reason about multiple items.
  - Syntax: `grocery_list: list[str] = ["eggs", "milk", "bread"]`
  - Can be an arbitrary length (you can add or remove items)
  - Empty List: list() or []
  - Indexing like strings, but can *modify* by index
  - Methods: append and pop
  - Functions can: take lists as arguments, return (or create + return) lists, and *modify* lists!