# LISTS

We have been using variables all year. A variable is a place in memory where we can store a value. It helps to think of a variable as a single box on shelf with a label (name), or a single file in a filing cabinet with a label (name). But how can I model the *entire drawer* of these 'files', or an *entire shelf* of these boxes?
Often we want to create an entire collection of variables. For example, say we wanted to store all the ages of the students in the class (or the whole school)? We would need to create a variable for each student's age. This would be very hard to manage, and you would never know how many variables to make. Instead, we can define a single variable to hold on to a *collection* of student ages. This collection is called a ***list***.

**Creation:**
To create a `list` in Python, we just put the values, separated by commas, in square parentheses. Lists of many values can be assigned to a *single* variable.

*Example:*
```
num_list = [1, 4, 6, 8, 19, 22]
print(type(num_list))  # prints <class 'list'>
```

**List indexing:**
List values are accessed by their index/position, just like strings. In order to access a particular element in a `list`, use square parentheses and the index number.

*Example:*
```
nums = [1, 4, 6, 8, 19, 22]
print(f"First: {nums[0]}, last: {nums[-1]}")
```

*Question*: what happens if you use an invalid index, such as `nums[6]`?

**Slicing:**
You can also take a piece of a `list` (a 'sublist'), just like with strings. This is known in Python as a "slice". To do so, include the start and end indices inside the square braces, separated by a colon. Note that the slice will not include the element at index 'end':

*Example:*
```
num_list = [1, 4, 6, 8, 19, 22]
sub_list = num_list[2:5]
print(sub_list)       # outputs [6, 8, 19]
print(num_list)       # outputs [1, 4, 6, 8, 19, 22]
```

Notice that the slice is actually another list, and the slice does not change the original list. If you omit the start index, it will act like using 0. If you omit the end index, it will take everything from the start index up to the end of the list (this is just like with strings!).

***Caution – Making a copy of a list:***
If you need to copy a list, you should use the list's `copy()` method or take a full slice. Assigning another variable to it (like you would with `int`s, `float`s, `str`s) won't work due the *object* nature of a list, and something called ***aliasing***. You will learn more about this if you take ICS4U.

Proper way to make a copy of a list:
```
list_copy = original_list.copy()
```
or
```
list_copy = original_list[:]   #a slice
```

**Mutability:**
Unlike strings which are immutable, **lists *are* mutable** which means you can modify the elements in a list without reassigning the list to a variable. To modify a specific element, you use its index.

*Example*:
```
num_list = [1, 4, 6, 8, 19]
num_list[2] = 7
print(num_list)
```

**List 'Arithmetic':**
Because lists are mutable, you can actually add on to the end of a list. However, you can only add a ***list*** to a ***list***, so the new item(s) must themselves be in a `list`.
*Example:*

```
num_list = [1, 4, 6, 8, 19]
other_list = [33, 44, 58]
num_list = num_list + other_list   # or num_list += other_list
num_list += [79]     # even adding a single value, it must be a list
num_list.append(85)  # another way to add a single value to the end
print(num_list)      # prints [1, 4, 6, 8, 19, 33, 44, 58, 79, 85]
```

You can also use the multiplication operator on lists, with the same results as with strings:
*Example:*

```
num_list = [1, 4, 6, 8]
num_list = num_list * 2   # or num_list *= 2
print(num_list)           # prints [1, 4, 6, 8, 1, 4, 6, 8]
```

You cannot use subtraction from a list. To remove an item you can use slicing or a method.

**For-Loops & Lists**
Remember the use of `range(0, 10)` for a for loop? Check this out:

```
num_list = list(range(0, 10))
print(num_list)  # prints [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

`range()` is actually a built-in Python function that returns a special '*range*' type of variable, but Python can ***type-cast*** ( like `int()`, `float()`, etc.) it into a **`list`** type. When you use it in a for-loop with something like `for num in range(0, 10):`, Python loops through the range variable just like it would loop through a `list`, ***iterating*** over all elements in the `list`, referring to them with the variable `num` one at a time. This is also just like how you can iterate over every character in a string or every line in a file. All `for`-loops use this *iterator* concept.
*Example: make a nice 'vertical sum string' of the values in a list:*

```
num_list = [8, 24, 5, 108, 19, 15]
sum = 0
for value in num_list:
    print(f"{value:>5}")
    sum += value
print("------")
print(f"={sum:>4}")
```

Output:

```
    8
   24
    5
  108
   19
   15
------
= 179
```

**List Functions & Membership:**
There are useful predefined functions in Python that can be applied to a list for finding the number of elements (length), the sum of the elements, and the min/max of the elements.
*Example:*

```
num_list = [8, 24, 5, 108, 19, 15]
count = len(num_list)
tot = sum(num_list)
large = max(num_list)
small = min(num_list)
print(f"n: {count}, Sum: {tot}, Max: {large}, Min: {small}")
→ prints  n: 6, Sum: 179, Max: 108, Min: 5
```

There is also the **`in`** (and the **`not in`**) membership operator, like with strings. Using it will evaluate to `True` or `False` if the value is (or is not) in the list.

```
num_list = [8, 24, 5, 108, 19, 15]
print(5 in num_list)  # prints True
print(10 in num_list)  # prints False
print(108 not in num_list)  # prints False
```

**List Methods:**
Like strings, lists have a set of built-in methods (things that they can do or that can be done on them).

| Method | Returns | Description |
|---|---|---|
| **`index`**`(value)` | `int` index | Return the index of the first occurrence of `value` in the list (or gives an error if `value` is not in the list) |
| **`index`**`(value, from)` | `int` index | Return the index of the first occurrence of `value` in the list starting at from (or error if `value` is not in list) |
| **`remove`**`(value)` | None | Remove the first occurrence of `value` from the list |
| **`pop`**`(n)` | the value removed | Remove and return the value at index `n` |
| **`copy`**`()` | `list` | Return a full copy of this list. |
| **`count`**`(value)` | `int` count | Return the number of occurrences of `value` in the list |
| **`sort`**`()` | ***None*** | Sort the list in numeric/unicode/other order***, mutating the list itself*** |
| **`reverse`**`()` | ***None*** | Reverse the contents of the list***, mutating the list itself*** |
| **`extend`**`(value_list)` | ***None*** | Like the '+' operator for adding a `list` to a `list`, appends the values in the given `value_list` to the end of the list |
| **`append`**`(value)` | ***None*** | Like the '+' operator, appends the given `value` to the end of the list ( `value` should not be in `[]`, unless you want it to be a *nested* list) |
| **`insert`**`(n, value)` | ***None*** | Inserts a single `value` into the list at index `n`, and shifts everything else down by one. |

***Caution – Lists as arguments to functions:***
You have been cautioned about how to ***copy*** a `list` properly. Due to the same issues behind the scenes, you need to be careful when passing a `list` to a function as an argument. Because of ***aliasing***, If you mutate/modify the list inside the function, you will be mutating the same list that was passed to the function, so changes will persist *outside* the function after it has executed. This was never the case with `int`/`float`/`str` or other values.

Sometimes, we want to take advantage of this, such as writing a function that removes all negatives from a list of numbers. Instead of *returning* the edited version we can ***mutate*** the argument itself. Often, however, this issue causes problems. If you ever need to mess with the contents of a `list` in order to make a function work properly, be sure to work with a *copy* of the list (see above for how to properly copy a list).

**Exercises:**

**1.** Ask the user for a set of marks, one at a time until the user enters a negative number. Output the largest, smallest, and median value of all input numbers. *Do this by putting the numbers in a* **`list`***, and using* ***built-in functions*** *that work with* `lists`.

**2.** Create a list of numbers by reading all values from an input file with *at least* 10 values in it (one per line).

- Using **`list` methods**, sort the list of numbers *de*scending order (highest to lowest).
- Using **slicing** create two new lists: **`positives`** which contains the numbers above 0, and **`not_positives`** which contains the numbers less than or equal to 0.
  Remember the code won't know how many numbers are positive or not based on the file read… you will need to figure out ***where*** (which position) the ordered numbers change from positive to not positive and perform the slice at that position.
- Write a **function** called `squares()` that will take the `list` of numbers as an argument and returns a ***new*** `list` with the square each of the values ($x^2$). The original list of numbers should remain unchanged after calling the function. To make sure this worked, after calling the function print the original list and the returned squares list to verify.

**3.** Write a function called `mirror()` that will take a single list as an argument, and will return a `list` that has the values of the list inside of it added to the end in reverse order. Try to write this in as short a way as possible (as a personal challenge). You must make sure that the original list given to the function does not get mutated/changed at all! Test this:

```
nums = [1, 5, 3, 8]
mirrored = mirror(nums)
print(nums)     → prints [1, 5, 3, 8]
print(mirrored)  → prints [1, 5, 3, 8, 8, 3, 5, 1]
```

**4.** Write a function called `rotated()` that will take a single list of strings and an integer as arguments. The function will return a new list that contains the same elements by 'rotated' forwards by the given integer amount of positions, with values coming off the end and moving to the front. For example, if the list is `['one', 'plus', 'two', 'is', 'three']` and they should be rotated by 3 the returned list would be `['two', 'is', 'three', 'one', 'plus']`

**5.** Write a function called `words()` that takes a string of words separated by commas as an argument (e.g. `"this,is,a,sample"`). The function will return a list of all the words, *in alphabetical order*, in ALL-CAPS (e.g. `["A", "IS", "SAMPLE", "THIS"]`).
As a hint, look at the description of the `split()` method for strings on **sheet 59**!

**6. a)** Write a function called **`intersection()`** that takes two lists as arguments and returns the intersection of the two lists (in order, with no duplicates).
*Intersection* means '*all items that exist in BOTH sets*'.

**b)** Write a function called **`union()`** that takes two lists as arguments, and returns the ***union*** of the two lists (in order, with no duplicates).
*Union* means '*all items that exist in EITHER set*'

**7. a)** Write a function called **`remove_n_largest()`** that takes a `list` of integers and a single `int` value $n$ as an argument. The function will remove the $n$-largest values from the list, mutating the list itself. The list's order should be maintained. *Nothing* is returned.
For example, if the list was `[5, 9, 7, 9, 2, 10, 8]` and 2 were removed ($n$ = 2), then the list would become `[5, 7, 9, 2, 8]` (note the first 9 was removed, not the second).

**7. b)** Write a function called **`n_largest_removed()`** that takes a list of integers and a single int value $n$ as an argument. The function will return a copy of the given list but with the $n$-largest values removed from it. The original list should not be mutated at all.
For example, if the list was `[5, 9, 7, 9, 2, 10, 8]` and 2 were removed, then the function would return the list `[5, 7, 9, 2, 8]`.

**8.** Write a function called **`longest_consecutive()`** that takes a list of integers as an argument and returns the length of the longest set of equal values in consecutive order. For example, if the list was `[5, 6, 6, 2, 9, 9, 9, 9, 4, 1, 1, 1, 0]` then this would return `4`.

*Harder, for those who want the meaningful challenge:*
**9.** Open a file that contains lines of 'student' data in the following sample format:

```
John, 56, 78, 32, 87
Jane, 94, 71
Sam, 89, 100, 78, 92
…etc
```

- Process that input string by storing the name in a list of all names, and store the marks in a separate `list` of all marks but keep that user's marks together as a list itself *within* that list…. For example, names would be `['John', 'Jane', 'Sam', '…']` and marks would be `[[56, 78, 32, 87], [94, 71], [89, 100, 78, 92], …]`, making this marks list a **`list of lists`**.
- Print out a single 'table' summarizing each student's name, their average mark, their highest mark, and their lowest mark.
- Print out the overall highest single mark and lowest mark of all students.
- Sample output with really nice 'table' formatting:

```
+-----------------------------+
|    NAME    | AVG  | LO | HI |
+-----------------------------+
|    John    | 63.2 | 32 | 87 |
|    Jane    | 82.5 | 94 | 71 |
|    Sam     | 90.0 | 78 |100 |
|    ...     |      |    |    |
+-----------------------------+
|           OVERALL | 32 |100 |
+-----------------------------+
```