Ch 4. Working with lists.

Now that you can define a list, add to it, delete items, rearrange lists, etc ... lets really use LISTS.

LOOPS — In programming a loop is a way to redo a command multiple times, changing just small parts.

For <variable name> in <list or range>: → don't forget the colon.

    do stuff

Indentation matters!!

Example:

```
my_colors = ["red", "blue", "green", "yellow", "black"]

for color in my_colors:
    print(color)
```

first time ↓ color = "red"
second time ↓ color = "blue"
⋮
until it runs out of data in the list.

You can put as many lines of code in a loop as you want!

INDENTATION.

for
    All code in the loop → don't "end" the loop python guesses the end based on indentation.

new code outside of loop

Indentation Errors: "unexpected indent"
"expected an indented block..."

Creating numerical lists range( )

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

new_num_list = list(range(1,5)) [1,2,3,4]

You can use this in loops.

eg. for i in range(len(my_colors)):
    print(my_colors[i])

prints each color just like the other loop!

Write code that squares all the numbers between 1 and 20 AND save them to a list.

Simple stastics: min( ), max( ), sum( ), etc.

List comprehensions. — for loop in a list!

squares = [value**2 for value in range(1,21)]

Slicing a list ... we know how to get just one element: squares[0] but what if we wanted the first three?

squares[0:3] squares[-3:]

squares[:4]

squares[4:]

Copying Lists.

```
my_cat_names = ["Bella", "Bongo", "Tater Tot"]

my_husbands_cat_names = my_cat_names[:]
```

need the slice to make a copy!

Tuples — looks like a list

```
dimensions = (200, 50)
```

but now rounded brackets.!

- we can access the data
- we can over write the tuple
- BUT — can't change just one entry

```
dimensions[0] = 100
```

raises an error!

Use tuples for sets of values that should not be changed during your code