Week 9 - ~~Final Project~~ Et Cetera

Set - data type, alternative, more succinct, method of appending to a list without appending duplicates.

```
houses = set()
for student in students:
    houses.add(student["house"])
```

global variable - A variable that is declared outside of all functions, classes, or blocks. It is accessible throughout the entire program, including inside functions, unless shadowed by a local variable with the same name

Example / View / Access Global Variable

```
x = 10 # Global Variable - defined outside of any function, at the top

def show_x():
    print(x)

show_x()

Output: 10
```

Example - Modifying a global variable

```
x = 10

def update_x():
    global x
    x = 30
update_x()
print(x)
Output: 30
```

argparse - Library in python used to handle command-line arguments passed in to a script. It allows you to specify what command-line options the program receives either optionally or necessarily, how to parse and interpret those options, and provides useful help and error messages. Useful for making programs configurable and flexible from the command-line.

Example

```
import argparse

parser = argparse.ArgumentParser() or (prog='script_name.py',
                                       description='script description',
                                       epilog='bottom text'
                                       )

parser.add_argument('-n', default=1, help='Number of times to meow',
                    type=int)
args = parser.parse_args()

for i in range(int(args.n)):
    print("meow")
```

Map - The map() function applies a given function to each item in an iterable (such as a list, tuple, etc.) and returns an iterator (a map object) with the results

syntax: map(function, iterable, ...)

example:

```
numbers = ["1", "2", "3", "4", "5"]
int_numbers = map(int, numbers)
print(list(int_numbers))
```

Output: [1, 2, 3, 4, 5]

Easy Unpacking of map

Remember * is used to unpack a list

More of * on next pg →

Unpack List - map() can be used in many ways including controlled unpacking of a list, however, the fastest way to unpack a list in its entirety is to put an asterisk* in front of it ie.

```
coin_list = [100, 50, 25]
print(coin_list)  output: [100, 50, 25]
//but
print(*coin_list) output: 100 50 25
```

You can also slice to unpack a certain amount

ie `coin_list = [100, 50, 25, 12, [4, 24]]`

`print(*coin_list[:3])` output: 100 50 25

--- Unpackin Dictionaries is similiar to the above method, only you instead use 2 asterisk **

↳ it would unpack:

`coins = {"galleons": 100, "sickles": 50, "knuts": 25}`

as

`galleons=100, sickles=50, knuts=25`

useful for passing into a function

↳ slicing does not work inherently for unpacking dictionary

↳ To get relevant information

↳ `print(**{key: coins[key] for key in ["galleons", "sickles", "knuts"]})`

Bonus!

Works directly on map

ie

`print(*map(str.upper, words))`

~~Line Comprehension~~

List Comprehension - Concise and efficient way to create lists in Python. It allows you to generate a new list by applying an expression to each element of an existing iterable (such as a list, range, or other sequence) in a single line of code

Syntax:

[expression for item in iterable (optional: if condition)]

expression - The operation or value to apply to each element
item - Represents each element in the iterable
iterable - The data source you're iterating over (eg. list, range, str, etc)
if condition - (optional) A filter to include only elements that satisfy said condition

Examples:

1. print(*[word.upper() for word in words]) # No Conditional (unpack)
2. squares = [x ** 2 for x in rang(10)] # No Conditional (squared)
3. evens = [x for x in range(10) if x % 2 == 0] # Conditional
4. coordinates = [(x, y) for x in range(3) for y in range(3)] # Nested
5. celsius = [0, 10, 20, 30]
   fahrenheight = [((9/5) * temp + 32) for temp in celsius] # With Function
   Numbers = [1, 2, 3, ...]
6. Categories = ["even" if x % 2 == 0 else "odd" for x in numbers]

Filter() - Similiar to map(), but instead of expecting function like int, str, str.upper etc, it expects a boolean expression resulting in True or False. Returns only the elements that satisfy the condition

syntax:

```
filter(function, iterable)
```

Dictionary Comprehension - Concise & efficient way to create dictionaries by applying an expression to each element in an iterable (such as a list or range) or transforming existing dictionaries

Syntax:

```
{Key_expression : Value_expression for item in iterable if condition}
```

(optional)

Examples:

1. `squares = {x: x**2 for x in range(5)}`

2. `even_squares = {x: x**2 for x in range(10) if x%2 == 0}`

Enumerate() - Adds a counter to an iterable (e.g. list, tuple, string) and returns it as an enumerate object. Useful for looping through an iterable while keeping track of the index

syntax:

```
enumerate(iterable, start=0)
```

→ Default starting position when 'start' not defined

Example:

```
fruits = ['Apple', 'Banana', 'Cherry']
for i, fruit in enumerate(fruits):
    print(i, fruit)
```

Add start enumerate(fruits, start=1)

Output:
0 Apple
1 banana
2 cherry

→ * to start at 1 instead of 0
↳ add start=1 in enumerate()