Efficient Phrases

These are efficient phrases:	These are not efficient phrases:
Cold Windowsill	Chilly Window Ledge
Cool Million	Good Thousand Thousand
Vivid Disillusions	Graphic Disappointments
Suspicious Conclusion	Mistrustful Ending

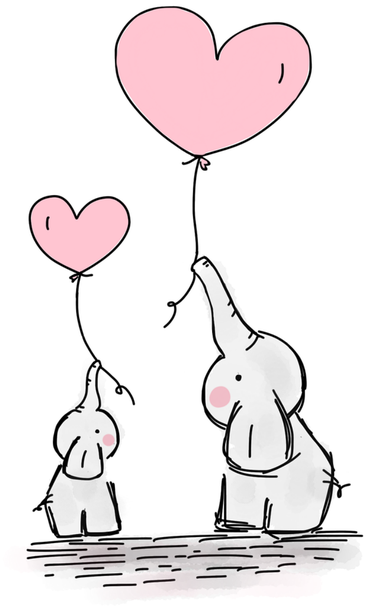
What makes an efficient phrase?

Data Structures

Data Structures

Data Structures

- Sequence Types
  - Tuples
  - Lists
  - `range`
- Mapping Types
  - Dictionaries
- Sets
- Advanced Looping
- Comprehensions



Sequence Types

Sequence type: an object type for storing an <u>ordered</u> collection of objects.

Today, we'll be exploring `tuple` objects, `list` objects, and `range` objects!

Tuples

Cannot be changed after it's been created.

Tuple: immutable sequence type, typically used to store a collection of heterogeneous data.

Can store objects of various type.

```
congrats = ("Happy", 4, "you", "dude!")
```

Parentheses are conventional, but optional!

Why Learn About Tuples?

Tuples are:

- **Hashable** - can be used as keys for dictionaries or as elements of sets. (We'll see more of this soon!)
- **Immutable** - "write-protect" data that doesn't need to be changed.
- **Memory efficient** - immutability means they are stored more compactly than lists. (Matters more when storing many elements).

Unpacking Tuples

- Python supports the ready conversion of tuple elements to variables…

```
tup = (3, 2, 1)
three, two, one = tup
```

- …and function arguments!

```
pow(*tup)
# => 0
```

The `*` indicates tuple unpacking; Python interprets this as `pow(3, 2, 1)`

Note: `pow(base, exp, mod)` returns `base**exp % mod`

Aside: Variable Swapping

Using a Temporary Variable:

```
tmp = a
a = b
b = tmp
```

Using Tuple Packing/Unpacking:

We create a tuple `(a, b)`…

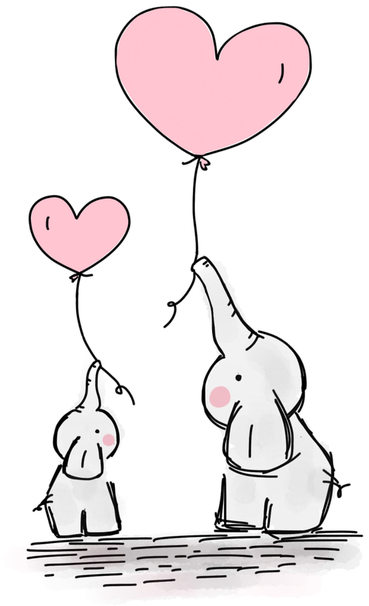
```
tup = (a, b)
b, a = a, b
b, a = tup
```

And unpack its values into `b` and `a`!

Data Structures

Data Structures

- ~~Sequence Types~~
  - ~~Tuples~~
  - Lists
  - `range`
- Mapping Types
  - Dictionaries
- Sets
- Advanced Looping
- Comprehensions



Lists

Can be changed after it's been created.

List: mutable sequence type.

```
jenny = [8, 6, 7, 5, 3, 0, 9]
```

Unlike with tuples, the brackets are mandatory!

Working with Lists

- Standard slicing rules apply to access elements and subsequences.
- Additionally, special list methods include:

`.count(elem)`	Counts the occurrences of `elem` in the list.
`.index(elem)`	Returns the index of the first occurrence of `elem` in the list.
`.append(elem)`	Appends the element `elem` to the end of the list.
`.extend(iterable)`	Extends the list by appending all elements of `iterable` to the end.
`.insert(idx, elem)`	Inserts the element `elem` at the index `idx` of the list.
`.sort(key=None, reverse=False)`	Sorts the list in-place.
`.reverse()`	Reverses the list in-place.
`.pop(i=-1)`	Returns and removes the ith element from the list.
`.remove(elem)`	Removes the first instance of `elem` from the list, or raises `ValueError`.

Mutability and Immutability

```
CS41_staff = (["Elizabeth", "Antonio", "Theo"], ["Pop Tart"])

CS41_staff[1].append("Unicornelius")
```

What's going to happen?

Tuples store references to underlying objects; if the objects are mutable, they can still be changed.

```
CS41_staff = (["Elizabeth", "Antonio", "Theo"],
              ["Pop Tart", "Unicornelius"])
```

(If these objects aren't hashable, though, the tuple won't be either!)

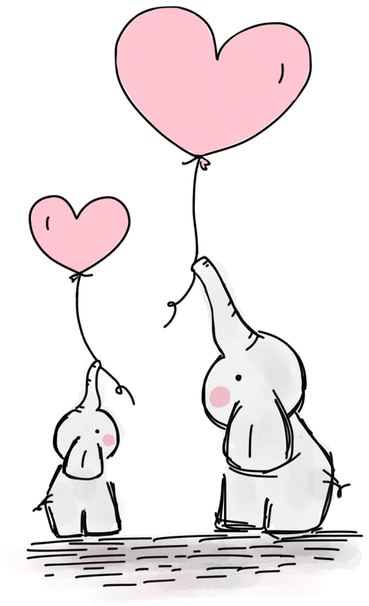
An interview question: write a program to remove duplicate elements of a list.

Fun fact: freshman-year Michael failed to solve this problem
in a real technical interview! 🙃

Data Structures

Data Structures

- ~~Sequence Types~~
  - ~~Tuples~~
  - ~~Lists~~
  - `range`
- Mapping Types
  - Dictionaries
- Sets
- Advanced Looping
- Comprehensions



range

Range: represents an immutable sequence of numbers, commonly used for looping in `for` loops.

Assumed to be 0 if excluded.

```
range(start, stop, step)
```

Assumed to be 1 if excluded.

```
range(10)
```

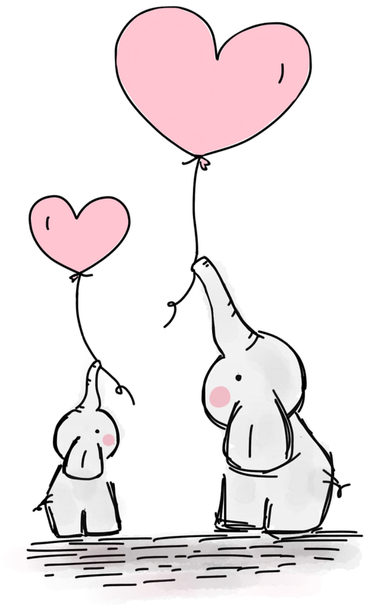
```
range(3, 10)
```

```
range(3, 10, 2)
```

Data Structures

Data Structures

- ~~Sequence Types~~
  - ~~Tuples~~
  - ~~Lists~~
  - ~~`range`~~
- Mapping Types
  - Dictionaries
- Sets
- Advanced Looping
- Comprehensions



Mapping Types

Dictionaries

For today, hashable and immutable mean the same thing - but we'll revisit this definition during the lecture on Object Oriented Python!

Dictionary: a data structure which maps hashable values to arbitrary objects.

Think Java's `HashMap`, or the Stanford C++ Library's `Map`.

Curly braces denote a dictionary.

```
cs41_staff = {
            "Parth": "the wonderful",
            "Antonio": "the bold",
            "Elizabeth": "the intrepid",
            "Theo": "the wizard"
            }
```

Colons separate keys, values; commas separate (key, value) pairs.

Working with Dictionaries

`val = d[key]`	Access the value in `d` corresponding to `key`; place this value into the `val` variable.
`d[key] = val`	Set the value in the dictionary corresponding to `key` equal to the value within `val`.
`d.get(key, default)`	Returns the value associated with `key` in `d`. If `key` does not exist in `d`, return `default`.
`d.keys()`	Returns a collection of the keys in the dictionary.
`d.values()`	Returns a collection of the values in the dictionary.
`d.items()`	Returns a collection of (key, value) tuples in `d`.
`del d[key]`	Removes `key`, and its associated value, from `d`. (If key is not in `d`, raises a `ValueError`).
`d.pop(key, default)`	Removes `key`, and its associated value, from `d`. (Returns the associated value if `key` is in `d`, otherwise returns default).

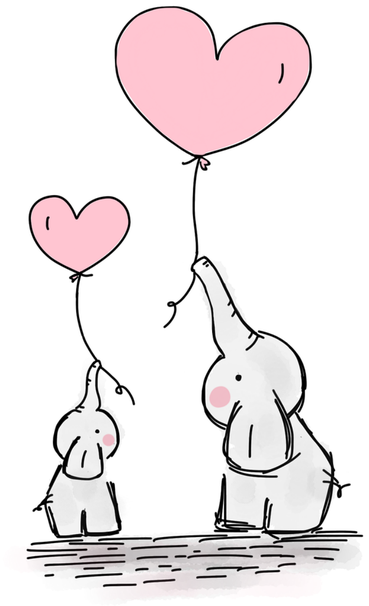
Common Dictionary Operations

`len(d)`	Returns the number of keys in `d`.
`key in d`	Equivalent to `key in d.keys()`
`d.copy()`	Makes a shallow copy of `d`.
`d.clear()`	Removes all (key, value) pairs from `d`.

Data Structures

Data Structures

- ~~Sequence Types~~
  - ~~Tuples~~
  - ~~Lists~~
  - ~~`range`~~
- ~~Mapping Types~~
  - ~~Dictionaries~~
- Sets
- Advanced Looping
- Comprehensions



Sets

Set: an unordered collection with no duplicate elements.

```
nice_animals = {"unicorns", "elephants"}
```

Curly brackets denote a set!*

* As long as it's not the empty set, which is denoted `set()`

Why Learn About Sets?

Sets enable:

- **Fast membership testing** - sets use hashing to enable O(1) membership testing. (List membership testing is O(n)).
- **O(1) Duplicate Elimination** - can eliminate duplicate entries in a collection.
- **Efficient Set Operations** - union, intersection, and more of your favourites from set theory!

Basic Set Operations

`s.add(val)`	Adds the value `val` to set `s`.
`s.remove(val)`	Removes the value `val` from set `s`. (Raises `KeyError` if `val` not in `s`).
`s.discard(val)`	Removes the value `val` from set `s` if it is present.
`s.pop()`	Remove and return an arbitrary element from s. (Raises `KeyError` if `s` is empty)

Mathematical Set Operations

`s & t`	Set intersection.
`s | t`	Set union.
`s < t`	Check whether `s` is a proper subset of `t`.
`s <= t`	Check whether `s` is a subset of `t`.
`s ^ t`	Symmetric difference.
`s - t`	Set difference.

An interview question: write a program to remove duplicate elements of a list that operates in O(n) time.

`set` vs. `frozenset`

- An immutable and hashable set - the elements of a `frozenset` must be hashable for the `frozenset` to be hashable.
  - Can be used - for example - as keys in a dictionary.
- Behaves almost exactly like a regular set, except doesn't support "mutable" operations:
  - `add`
  - `remove`
  - `discard`
  - `pop`

Efficient Phrases

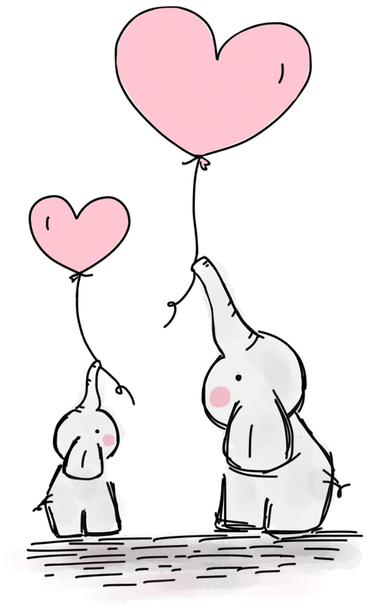
These are efficient phrases:	These are not efficient phrases:
Cold Windowsill	Chilly Window Ledge
Cool Million	Good Thousand Thousand
Vivid Disillusions	Graphic Disappointments
Suspicious Conclusion	Mistrustful Ending

What makes an efficient phrase?

Data Structures

Data Structures

- ~~Sequence Types~~
  - ~~Tuples~~
  - ~~Lists~~
  - ~~`range`~~
- ~~Mapping Types~~
  - ~~Dictionaries~~
- ~~Sets~~
- Advanced Looping
- Comprehensions



Advanced Looping

An *iterator* is an object which iterates through a collection over which it is defined.

`zip`: makes an iterator that aggregates elements from each of the arguments.

An *iterable* is anything that can be looped over using a `for` loops (`list`, `set`, `dict`, etc.)