Tuples are immutable sequences. Lists are mutable sequences. Dictionaries are unordered collections of key-value pairs.

Dictionary keys do have two restrictions:

- A key of a dictionary cannot be an object of a mutable built-in type.
- Two keys cannot be equal. There can be at most one value for a key.

A range is a sequence of consecutive integers:

..., -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5, ...

Generator expressions

\[
\begin{align*}
\text{\{map exp\} for <name> in \{iter exp\} if \{filter exp\}}
\end{align*}
\]

- Evaluates to an iterable object.
- \{iter exp\} is evaluated when the generator expression is evaluated.
- Remaining expressions are evaluated when elements are accessed.

List comprehensions

\[
\begin{align*}
\text{\{map exp\} for <name> in \{iter exp\} if \{filter exp\}}
\end{align*}
\]

- Short version: \{map exp\} for <name> in \{iter exp\}

Unlike generator expressions, the map expression is evaluated when the list comprehension is evaluated.

\[
\begin{align*}
\text{\{lookup('WHITE' + s.upper()) + ' ' + SUIT for s in suits}}
\end{align*}
\]

\[
\begin{align*}
\text{\{lookup('WHITE' + s.upper()) + ' ' + SUIT for s in suits}}
\end{align*}
\]

Mutable values can be changed without a nonlocal statement.

\[
\begin{align*}
\text{\{lookup('WHITE' + s.upper()) + ' ' + SUIT for s in suits}}
\end{align*}
\]

\[
\begin{align*}
\text{\{lookup('WHITE' + s.upper()) + ' ' + SUIT for s in suits}}
\end{align*}
\]

\[
\begin{align*}
\text{\{lookup('WHITE' + s.upper()) + ' ' + SUIT for s in suits}}
\end{align*}
\]

\[
\begin{align*}
\text{\{lookup('WHITE' + s.upper()) + ' ' + SUIT for s in suits}}
\end{align*}
\]

\[
\begin{align*}
\text{\{lookup('WHITE' + s.upper()) + ' ' + SUIT for s in suits}}
\end{align*}
\]

\[
\begin{align*}
\text{\{lookup('WHITE' + s.upper()) + ' ' + SUIT for s in suits}}
\end{align*}
\]

\[
\begin{align*}
\text{\{lookup('WHITE' + s.upper()) + ' ' + SUIT for s in suits}}
\end{align*}
\]

\[
\begin{align*}
\text{\{lookup('WHITE' + s.upper()) + ' ' + SUIT for s in suits}}
\end{align*}
\]
To evaluate a dot expression: `<expression>. <name>`

1. Evaluate the `<expression>` to the left of the dot, which yields the object of that dot expression.
2. `<name>` is matched against the instance attributes of that object; if an attribute with that name exists, its value is returned.
3. If not, `<name>` is looked up in the class, which yields a class attribute value.
4. That value is returned unless it is a function, in which case a bound method is returned instead.

To look up a name in a class:
1. If it names an attribute in the class, return the attribute value.
2. Otherwise, look up the name in the base class, if there is one.

### Assignment statements with a dot expression on their left-hand side

- Assign attributes for the object of that dot expression
- Assignment statements with a dot expression on their left-hand side affect attributes for the object of that dot expression
- If the object is an instance, then assignment sets an instance attribute
- If the object is a class, then assignment sets a class attribute

```python
def __init__(self, account_holder):
    self.balance = 0
    self.balance = account_holder
    self.balance = amount
    return self.balance
```

### Type dispatching: Define a different function for each possible combination of types for which an operation is valid

1. Attempt to coerce arguments into values of the same type
2. Apply type-specific (not cross-type) operations

```python
def coerce_apply(operation_name, x, y):
    if x = y:
        return operation_name(x, y)
```