Fall 2022

CL04 - Practice with Scopes, for..in - Introducing Sequences

Announcements

- Quiz Grades Post by Saturday
- EX06 - Choose Your Own Adventure
  - Released Tuesday 9/27, Due Monday 10/4
- RD01 - Weapons of Math Destruction
  - Released today, Due 10/28

Challenge Question #1

```python
"""CQ1) Scopes."""


def f(x: float) -> float:
    x += 1.0
    y: float = x + 2.0
    return x + y


def g() -> None:
    global y
    x: float = f(3.0)
    y = f(x + 4.0)


x: float = 0.0
y: float = 0.0
g()
print(f"{x}, {y}")
```

Sequences!

What is a Sequence?

- An **Abstract Data Type** that is an ordered, 0-indexed set of values.

- There are many specific *types* of sequences with their own properties. Common, built-in sequence types in Python include:

1. str - a sequence of character data
2. list - a dynamically-sized sequence of values of a specific type
3. tuple - a fixed-size sequence of values of any types
4. range - a sequence of integers at intervals between a start and end

Tuples !

Tuple Types

1. Tuples types are *made of a specific, fixed-length sequence of any mixed type(s)* by:

```
tuple[type0, type1, ..., typeN]
```

3. Typically you will want to alias your Tuple types to give them a more meaningful name

Examples:

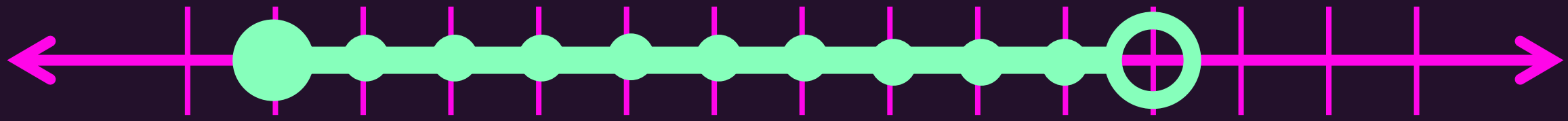
```
Point2D = tuple[float, float]
Color = tuple[int, int, int]
Player = tuple[str, float]
```

4. You **construct** a Tuple with a Tuple literal. Tuple variables of the above types could be initialized as follows:

```
origin: Point2D = (0.0, 0.0)
gray: Color = (128, 128, 128)
bacot: Player = ("Bacot", 5)
```

Ranges !

Ranges of Integers



- What are the ***attributes*** of the ***range*** above?

- A **start** point that is inclusive
- A **stop** point that is exclusive
- A **step** that moves up by one

The `range` type *models* the *idea* of a Range

- `range` is a built-in *sequence type* in Python
  - Just like `str`, `tuple`, and `list`
  - A range value is immutable, like `str` and `tuple`
  - Documentation: https://docs.python.org/3/library/stdtypes.html#ranges

- The `range` constructor returns a range object

```
range(start: int, stop: int[, step: int = 1]) -> range
```

  - **start** is *inclusive.*
  - **stop** is exclusive
  - **step** defaults to **1** and is *optional*, as denoted by the brackets

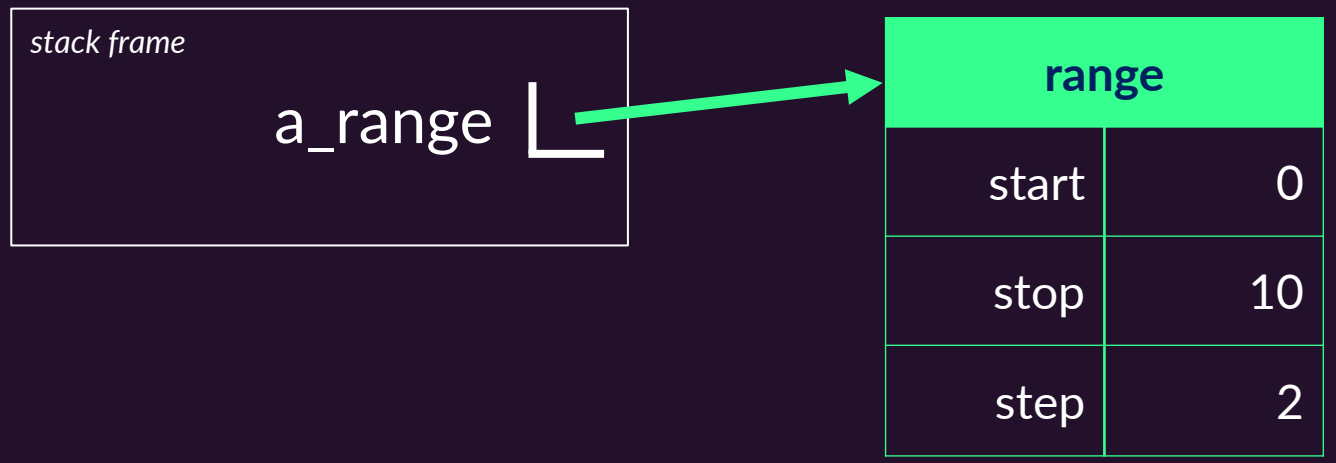
A `range` object has *attributes*

- ***Attributes*** are named values bundled in an object
  - *Attributes* represent the *state* of an object
  - **Named** like variables, unlike indexed items of a tuple or list. Attribute names are *identifiers*.
  - Hold **Values**, also like variables, unlike *methods* which are special functions

- Attributes are accessed using the dot operator following the object:
  `[object].[attribute_name]`

- Example:

```
>>> a_range: range = range(0, 10, 2)
>>> a_range.start
0
>>> a_range.stop
10
>>> a_range.step
2
```



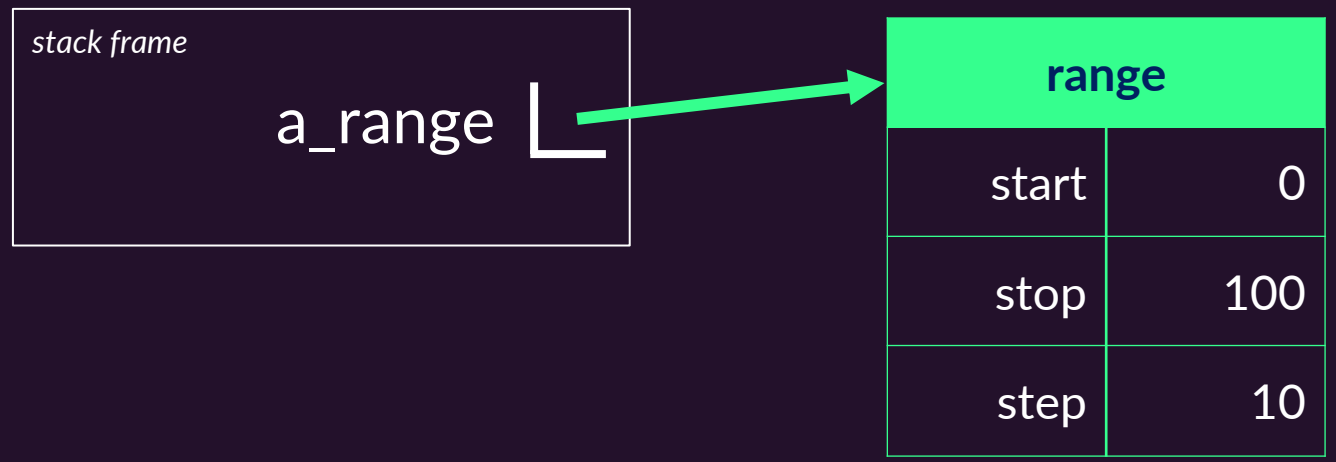
- The range object's attributes are read-only, making a range an *immutable object*

A `range` object is a *sequence* type

- You can access items in a range's sequence *by its index* using subscription:
  - range[0], range[1], ..., range[N]

- Example:

```
>>> a_range: range = range(0, 100, 10)
>>> a_range[0]
0
>>> a_range[1]
10
>>> a_range[9]
90
>>> a_range[10]
IndexError: range object index out of range
```



- Notice the *range* object's state is **only** its three attributes
  - *But* as a *sequence type*, with subscription, it also behaves as if it is made of many more items.
  - How? **Abstraction**! In this case the **abstraction** of a range is fully **represented** by just three attributes.

- This abstraction is possible through arithmetic
  `range[index]` evaluates to `range.start + (range.step * index)`

Using for..in + range