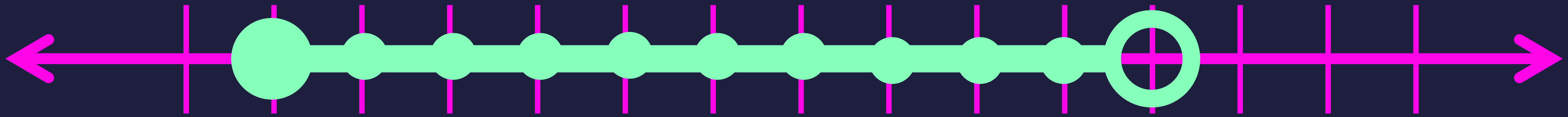
Ranges!

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` *function* constructs a range object

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

The step parameter defaults to **1** and is *optional*, as denoted by the brackets

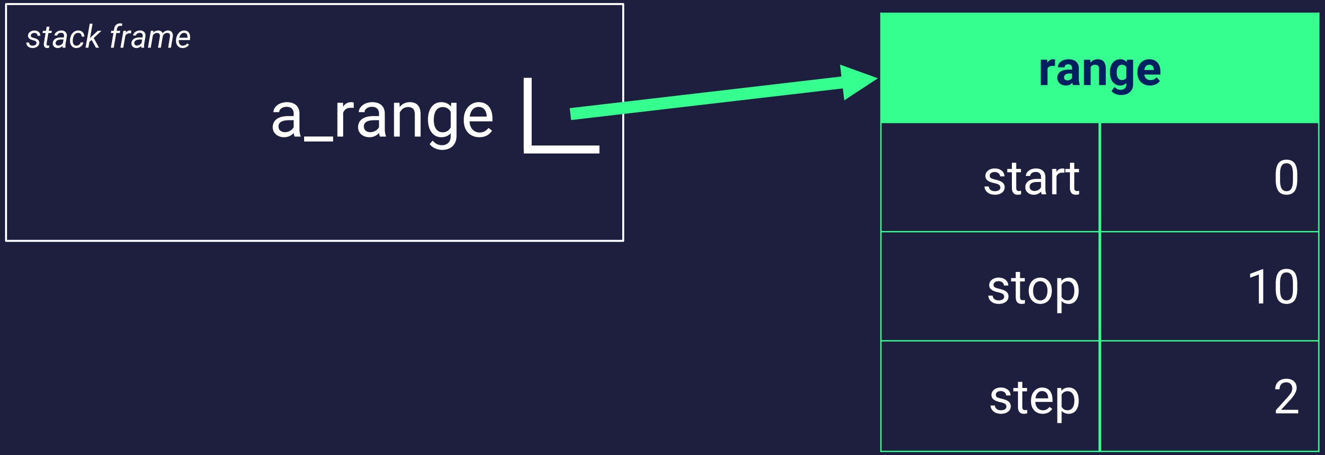
A `range` object has *attributes*

- ***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(0, 10, 2)
>>> a_range.start
0
>>> a_range.stop
10
>>> a_range.step
2
```



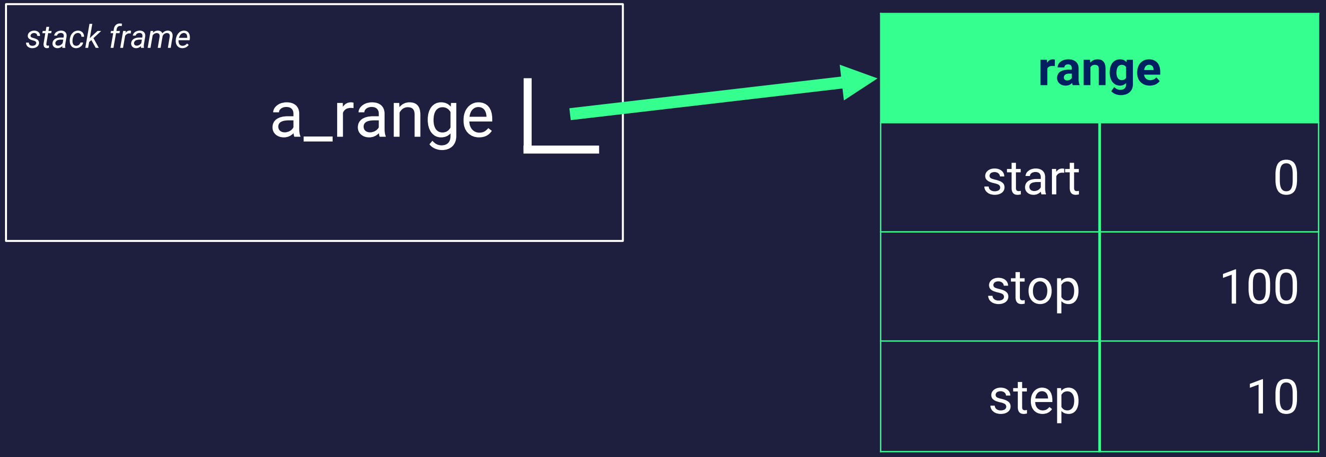
- The range object's attributes are read-only, making a range an *immutable object*

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(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 `ranges` with `for..in` loops (1/2)

- Ranges are commonly used for indexing other sequences:
  - Typically used with other lists and tuples

- Example:

```
>>> a_range = range(0, 6, 2)
>>> for i in a_range:
...     print(i)
...
0
2
4
```

- Be careful: *stop* is *not inclusive!*

Using `range`s for indexing other sequences (2/2)

- Ranges are often used to index other sequences with `for..in` loops

Consider:

```
>>> a_list = ["a", "b", "c", "d"]
```

Example: Index every other item with a step of 2.

```
>>> a_range = range(0, len(a_list), 2)
>>> for i in a_range:
...     print(a_list[i])
a
c
```

Example: Index in reverse.

```
>>> for i in range(len(a_list) - 1, -1, -1):
...     print(a_list[i])
d
c
b
a
```

Notice: This use case is *why* `stop` is *non-inclusive*!

Abstraction Win: Works in *most* indexing scenarios and avoids accidental infinite loops!