COMP 110

Spring 2022

Class 16 - Dictionary Practice and Sequences

Today's Goals

1. Practice with dictionaries
2. Sequences
3. Dictionaries vs. Sequences

Announcements

- Monday and Wednesday - Tutoring in Sitterson Lower Lobby
- Tonight: EX06 - Dictionary Utils
- Regrade Requests via Gradescope
- Quiz Thursday 3/3: Lessons through Today's!

Diagram 1)

```
2   square_to_root: dict[int, int] = {}
3
4   i: int = 1
5   while i < 5:
6       square_to_root[i ** 2] = i
7       i += 1
8
9   print(square_to_root)
```

Check for understanding: why couldn't square_to_root be a list[int]?

Diagram #2 - Assume `__name__` is `"__main__"`

```python
"""Helper functions imported elsewhere."""


def main() -> None:
    game0: dict[str, int] = {"KJ": 0, "ML": 1}
    game1: dict[str, int] = {"ML": 2, "EW": 3}
    merged: dict[str, int] = merge(game0, game1)
    print(merged)


def merge(a: dict[str, int], b: dict[str, int]) -> dict[str, int]:
    """Merge two dictionaries."""
    result: dict[str, int] = {}
    for key in a:
        result[key] = a[key]
    for key in b:
        result[key] = b[key]
    return result


if __name__ == "__main__":
    main()
```

Diagram #2

Assume `__name__` is `"__main__"`.

```python
"""Helper functions imported elsewhere."""


def main() -> None:
    game0: dict[str, int] = {"KJ": 0, "ML": 1}
    game1: dict[str, int] = {"ML": 2, "EW": 3}
    merged: dict[str, int] = merge(game0, game1)
    print(merged)


def merge(a: dict[str, int], b: dict[str, int]) -> dict[str, int]:
    """Merge two dictionaries."""
    result: dict[str, int] = {}
    for key in a:
        result[key] = a[key]
    for key in b:
        result[key] = b[key]
    return result


if __name__ == "__main__":
    main()
```

Lists vs. Dictionaries

- Create a grid on your paper:
- Fill in with your neighbors!

	Same	Different
Lists		
Dictionaries		

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:

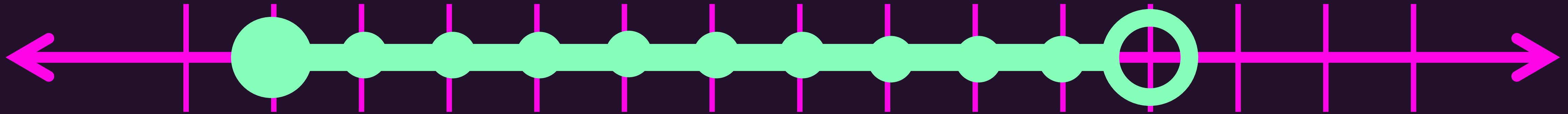
```
Point2D = tuple[float, float]
Color = tuple[int, int, int]
Player = tuple[str, float]
```

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
```

range	
start	0
stop	10
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)`