# The Syntax of Classes and Objects in Python

# Defining a Class - *"Inventing a Composite Data Type"*

```
class [ClassName]:
    [attribute0_name]: [attribute0_type]
    [attribute1_name]: [attribute1_type] = [attribute1_default_value]
    ...
    [attributeN_name]: [attributeN_type]
```

- **ClassNames** begin with an uppercase letters, subsequent words capitalized
- **Attributes** are declared in the class body
  - These are *just like* variable declarations
  - Attributes can be assigned default values (as shown in attribute$_1$)
- "A [ClassName] object will have an [name] attribute of type [type]".
  - *"A **TwitterProfile object** will have a **followers attribute** of **type int**"*

## Defining a Class - Example

- Here we are defining a class named **`TwitterProfile`**.
- *Every object* of type `TwitterProfile` will have three attributes:
  - `handle`, `followers`, and `is_private`
- In defining a class, you've invented a new type! You can now use it *as a type*. For example, in a variable declaration:

```
class TwitterProfile:
    handle: str
    followers: number = 0
    is_private: bool = True
```

```
a_profile: TwitterProfile
```

# **Initializing** a composite data type value requires **Constructing** a new object.

```python
a_profile: TwitterProfile = TwitterProfile()

a_profile = TwitterProfile()
```

- Unlike built-in types which have *literal syntax*, to establish an object whose type is custom, you must **"construct"** it
- The **constructor** is a *special function* responsible for **initializing** an object from a class
  - Every Python class has a *default constructor*.
  - Soon you will learn to write your own.

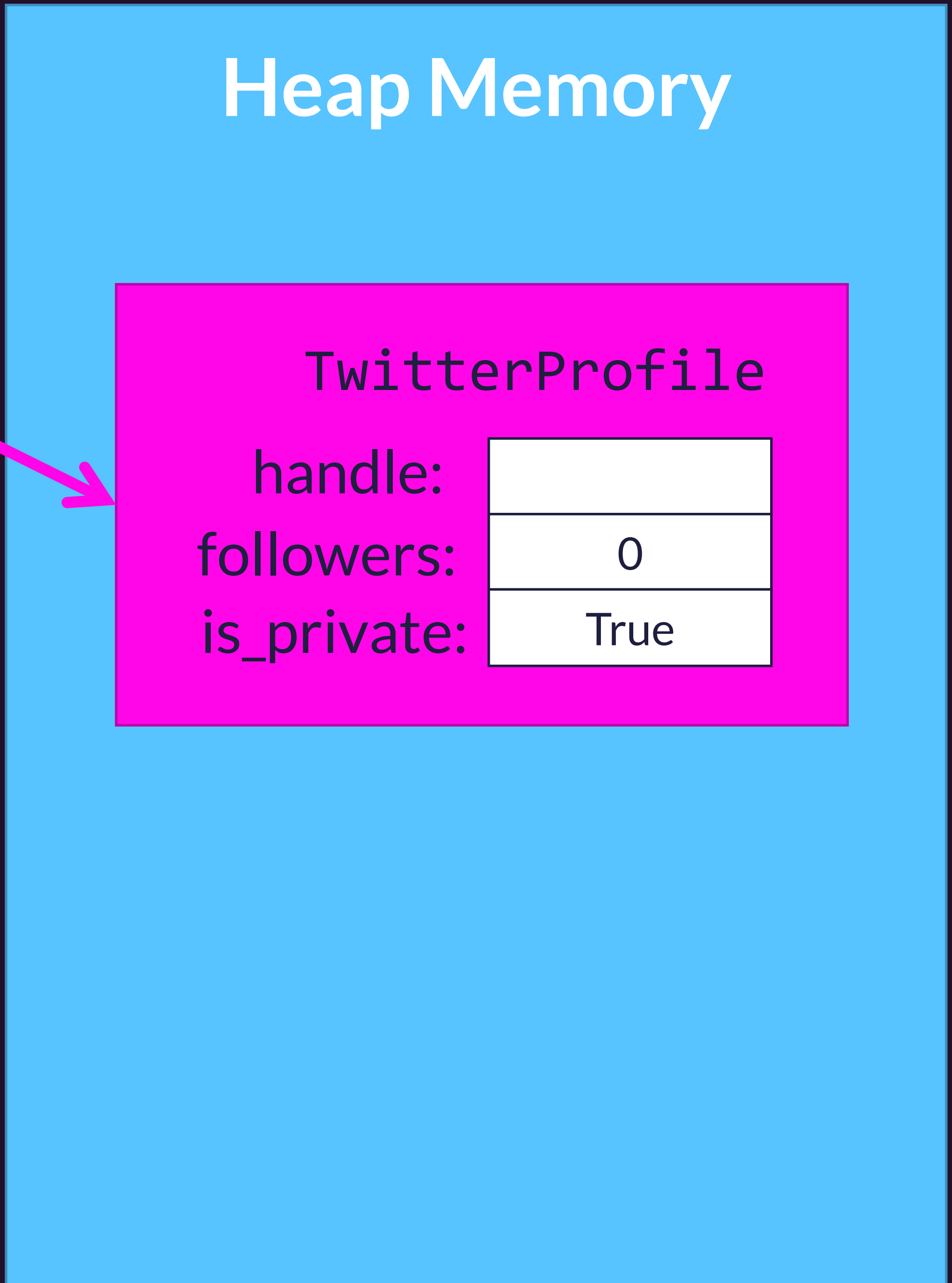
Disclaimer: Constructing objects in Python *does not require* any special keywords. In *many other languages* (Java, C++, TypeScript, PHP, ...) this same task requires using a special keyword often called **new**.

- For example, the second example above would be: **a_profile = new TwitterProfile();** in those languages.

# Constructing an Object

```
a_profile = TwitterProfile()
```

- When the **TwitterProfile()** expression is evaluated...
- ...the processor **constructs** a **new** object in heap memory with space allocated for each attribute.
- Any default values of an attribute are bound to the class' defaults.
- If a *custom constructor* is defined, it is evaluated.
- Finally, **a reference** to this object is returned and assigned to the **a_profile** variable.



# Reading an Attribute

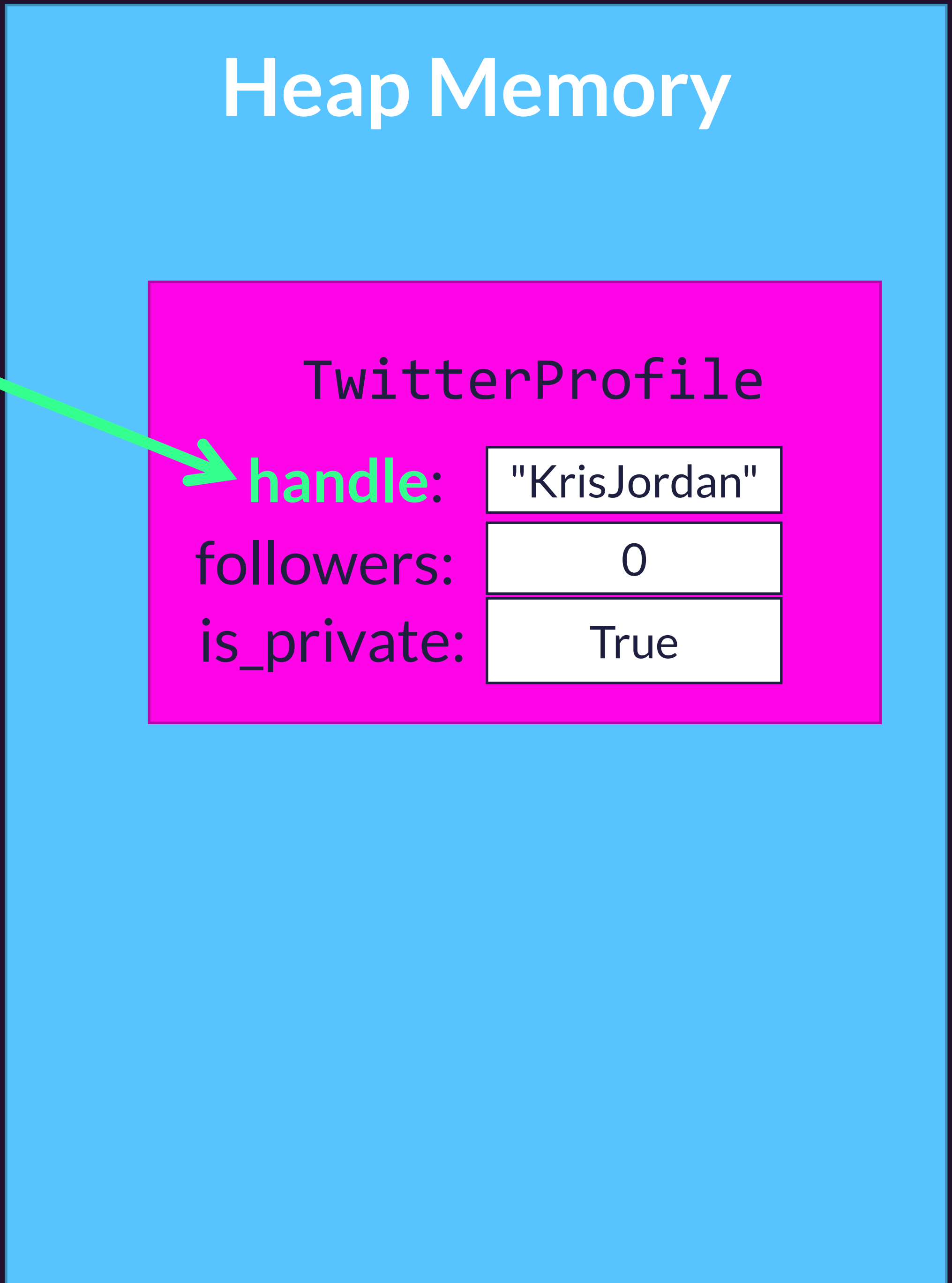
```
print(a_profile.handle)
```

- By referencing the TwitterProfile variable's name, followed by the *dot* operator, followed by an attribute name, we are saying:

*"Hey **a_profile**,*
*what is your **handle** attribute's value?"*

- General form:

```
[object].[attribute]
```



## Assigning to an Attribute

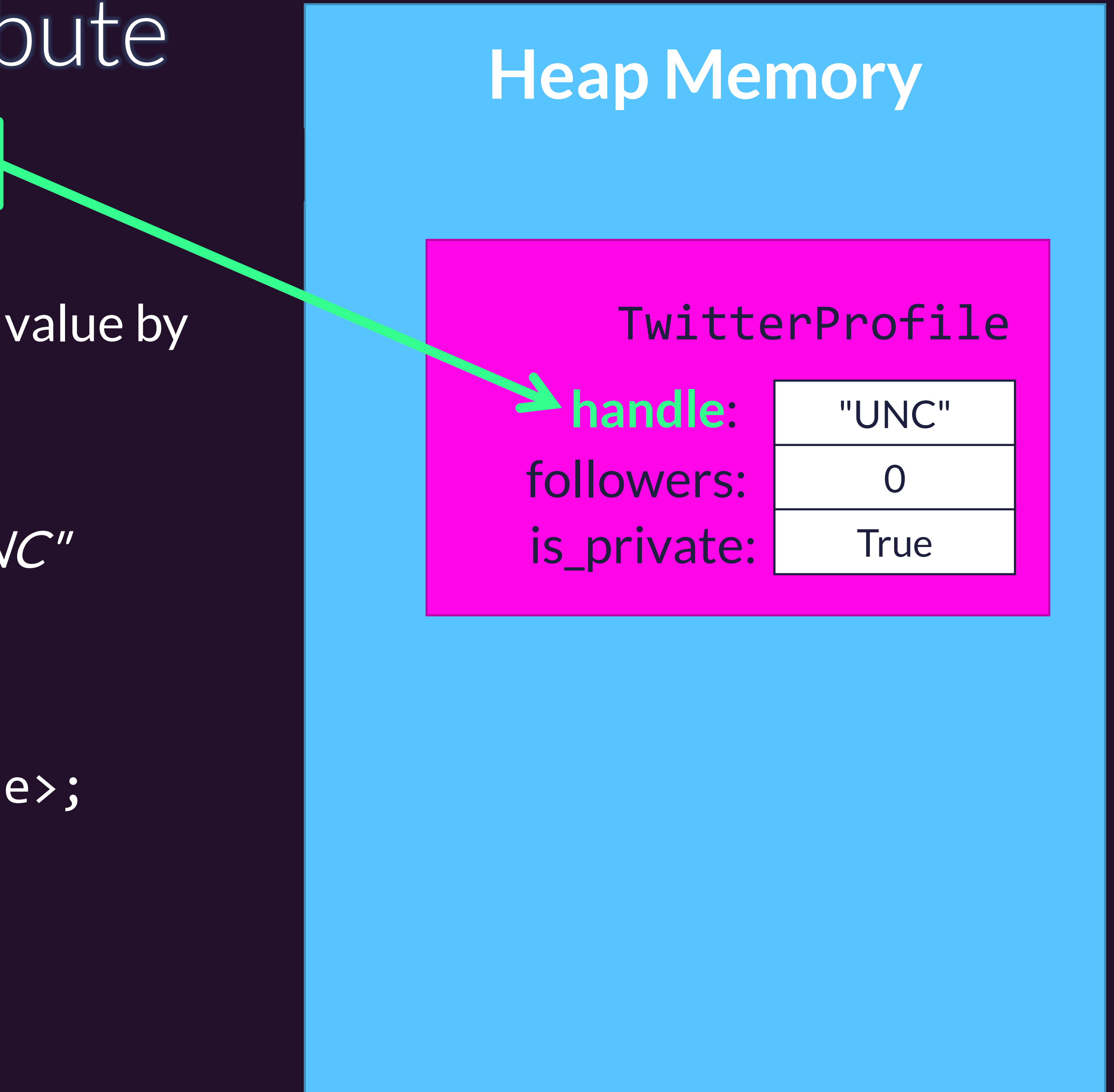
```
a_profile.handle = "UNC";
```

- We can change an object's property value by using the assignment operator.

*Hey **a_profile**, your **handle** is now "UNC"*

- General form:

```
  <object>.<property> = <value>;
```



# A Few Words on Words

- Object-oriented Programming Terminology is language specific
  - The concepts we're focusing on translate directly in other languages, even though other languages will call them by different names.

- Python's ***attributes*** are:
  - Java's **instance variables**
  - C++'s data **members**
  - JavaScript's object **properties**

- **Objects** are often referred to as ***instances*** of a class

- There can be subtle semantic differences between each language's rules around an object's attributes, but these details are far less important than the general concepts.