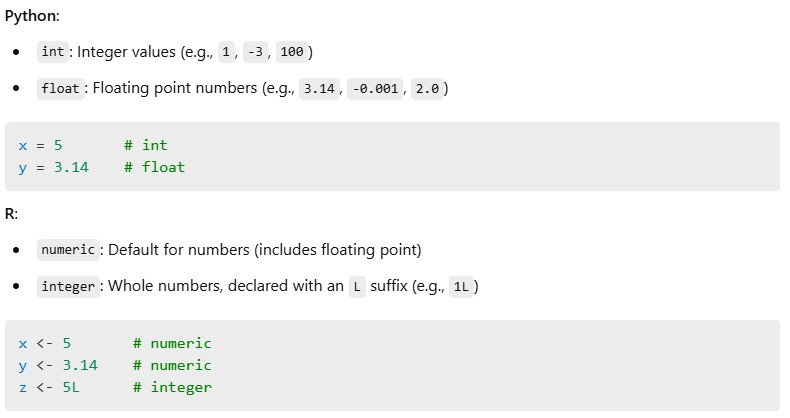
**Pre-reading for sessions 2 and 3**

**Python vs. R data types**

**1. Numeric Types**

R's integers have a unique behavior compared to other languages like Python. In R, when you assign a number (e.g., x <- 5), it defaults to a numeric type (which is actually a double-precision floating point). To explicitly create an integer, you must use the L suffix (e.g., 5L). This matters for type checking and memory usage in larger datasets, and can affect operations like indexing or model specification that expect integer input.

**Similarity**:

* Both languages support integers and floating point values.

**Difference**:

* R treats all numbers as numeric (float) unless explicitly declared as integers.

**2. Character / String Types**

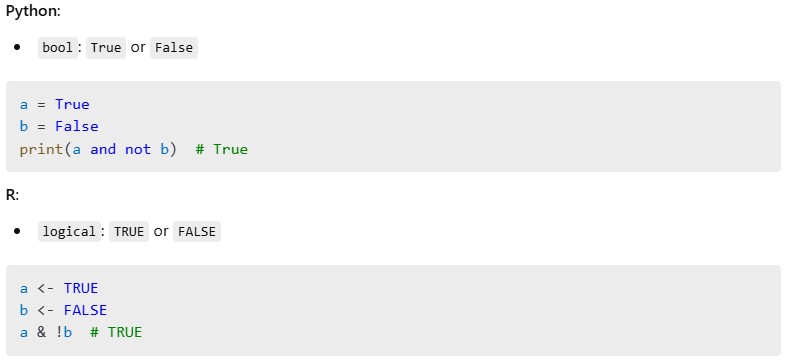
**Similarity**:

* Both use quotes for string literals and support string manipulation functions.

**Difference**:

* Python has native string methods (e.g., .upper(), .split()), while R relies on functions like toupper() and strsplit().

**3. Logical / Boolean Types**



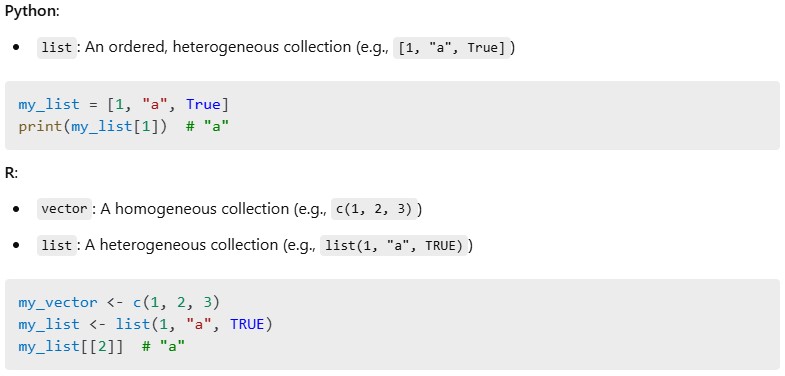
**Similarity**:

* Logical values represent binary truth values.

**Difference**:

* Python uses capitalized True/False; R uses TRUE/FALSE (in all caps).

**4. Lists and Vectors**

**Similarity**:

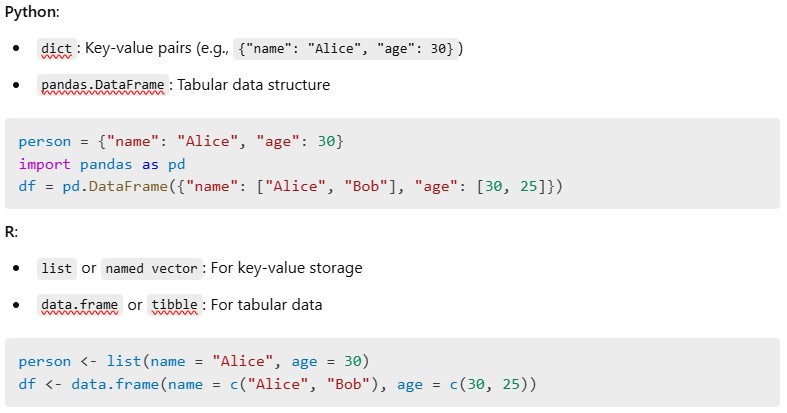
* Both support ordered collections and allow for nested structures.

**Difference**:

* R distinguishes between vector (homogeneous) and list (heterogeneous), whereas Python's list can hold anything.

**5.**

**Data Frames and Dictionaries**

**Similarity**:

* Both support named collections and tabular data formats.

**Difference**:

* Python separates dictionaries and dataframes; R uses lists for both concepts in different forms.

**Summary Table**

|  |  |  |
| --- | --- | --- |
| **Concept** | **Python** | **R** |
| Integer | int | integer (1L) |
| Float | float | numeric |
| String | str | character |
| Boolean | bool | logical |
| Homogeneous Seq. | numpy.array | vector |
| Heterogeneous Seq. | list | list |
| Key-Value Store | dict | named list/vector |
| Table Data | pandas.DataFrame | data.frame, tibble |

**Pandas Nullable vs Non-Nullable Dtypes and Missingness: A Tutorial**

When working with missing data in pandas, it's important to understand the difference between **nullable** and **non-nullable** dtypes. This tutorial explains what they are, how they behave, and how to work with them properly.

**1. What Are Nullable Dtypes?**

**Nullable dtypes** are pandas-native data types that support missing values represented by pd.NA.

Examples:

* Int64 (nullable integer)
* Float64 (nullable float)
* boolean (nullable boolean)
* string (nullable string)

These are distinct from the traditional **non-nullable** NumPy-based dtypes:

* int64, float64, bool, object

**Key Difference:**

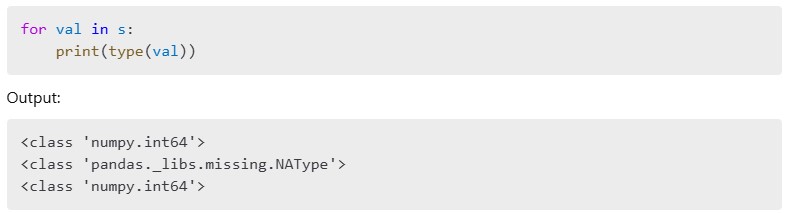
| **Feature** | **Nullable dtype (Int64, Float64, etc.)** | **Non-nullable dtype (int64, float64, etc.)** |
| --- | --- | --- |
| Supports pd.NA | Yes | No |
| Keeps column type with missing values | Yes | No (upcasts to float or object) |
| Scalar types | np.int64 or pd.NA | np.int64, np.nan |

**2. Creating Nullable Columns**



**3. Type Behavior: Scalars vs Series**

When looping over values in a Series:

**Explanation:**

* Even though the Series dtype is Int64, non-missing values are still np.int64 scalars.
* Missing values are pd.NA, which is of type pandas.\_libs.missing.NAType.

**4. Common Pitfall: Using .astype("float")**

**What happens:**

* The column becomes NumPy float64
* pd.NA is silently converted to np.nan
* All values become regular Python floats

**Better approach:**

This keeps pd.NA and allows you to properly track missing values.

**5. Nullable Dtype Conversion Cheatsheet**

| **Desired Type** | **Use This dtype** | **Allows pd.NA?** |
| --- | --- | --- |
| Integer w/ missing | Int64 | Yes |
| Float w/ missing | Float64 | Yes |
| Boolean w/ missing | boolean | Yes |
| String w/ missing | string | Yes |
| Classic integer | int64 | No |
| Classic float | float64 | No |

**6. Quick Recap**

* Use capitalized dtypes (Int64, Float64, boolean, string) to support pd.NA
* Avoid lowercase NumPy dtypes (int64, float64) if your data has missing values
* Looping over values shows actual scalar types: np.int64, float, or pd.NA

**7. Bonus: Auto-convert dtypes**



* Automatically converts columns to the best nullable dtype
* Useful after reading in messy CSV or Excel files

With this understanding, you can handle missing data in pandas more gracefully and avoid silent type conversions that lead to confusing bugs!

**8. Comparison of Common Missing Types in Python/Pandas**

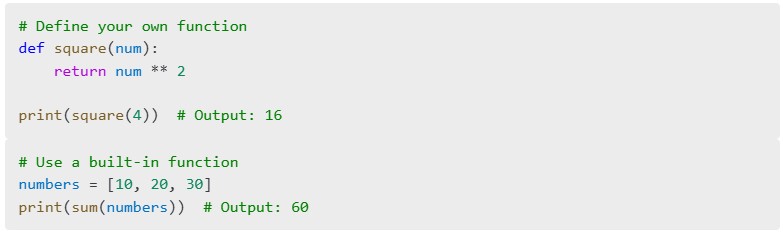
|  |  |  |  |
| --- | --- | --- | --- |
| Feature | None | np.nan | pd.NA |
| **Type** | NoneType | float (np.float64) | pandas.\_libs.missing.NAType |
| **Use Case** | General Python | NumPy/Pandas numeric data | Pandas nullable types |
| **Arithmetic Ops** | Fails (None + 1) | Works but returns nan | Works but returns <NA> |
| **Comparison (==)** | True (None == None) | False (np.nan == np.nan) | <NA> (pd.NA == pd.NA) |
| **Check Method** | x is None | np.isnan(x) | pd.isna(x) |

**Functions vs Methods in Python: Pre-Reading Guide**

* **What’s the difference?**

|  |  |  |
| --- | --- | --- |
| **Concept** | **Function** | **Method** |
| Definition | A block of code that performs an action | A function that is **associated with an object** |
| Called on | Standalone / with parameters | Called **on** an object (e.g., a string or list) |
| Syntax | function(arg) | object.method() |
| Example | max([3, 5, 1]) | [3, 5, 1].sort() |

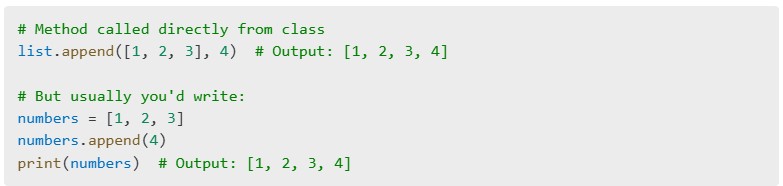
* **Function Example**



* **Method Example**



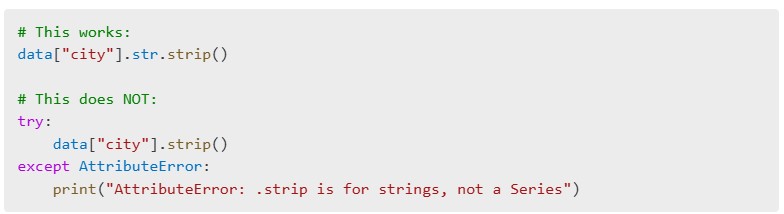
* **Reflection:**
* Why does sentence.replace() not change the original string?
* Why does fruits.extend() modify the original list?
* **Behind the Scenes**



* **Pandas Example: Method Chaining**



* **Why .str.strip() and not just .strip()?**



* **Rule of Thumb:**

|  |  |  |
| --- | --- | --- |
| You have... | Use... | Why? |
| A single string | "hello".strip() | It's a Python string |
| A Series of strings | df["col"].str.strip() | It's pandas, working element-by-element |