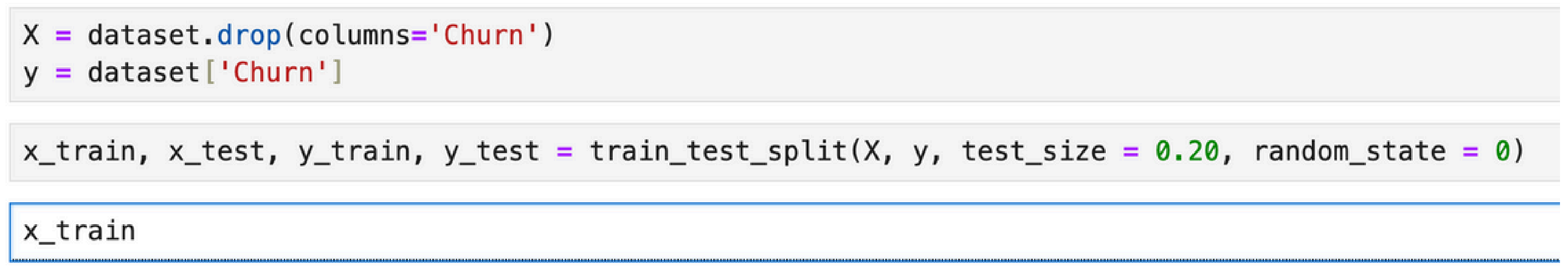
WELLCOME

INTRO. TO DATA SCIENCE

EMBARKING ON A JOURNEY INTO DATA SCIENCE

YA MANON



```
X = dataset.drop(columns='Churn')
y = dataset['Churn']
```

```
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size = 0.20, random_state = 0)
```

```
x_train
```

VARIABLES

VARIABLES

In this section we'll introduce the concept of **variables**, including how to properly name, overwrite, delete, and keep track of them

TOPICS WE'LL COVER:

- Variable Assignment
- Overwriting & Deleting
- Naming Conventions
- Tracking Variables

GOALS FOR THIS SECTION:

- Learn to assign variables in Python
- Understand the behavior for overwriting variables
- Learn the rules & best practices for naming variables

VARIABLE ASSIGNMENT

Variables are containers used to label and store values for future reference

- They can store any Python data type (and even calls to functions!)

```
variable_name = value
```

Intuitive label assigned to the variable

Examples:

- *price*
- *city*
- *heights*

Initial value assigned to the variable

Examples:

- *10.99*
- *'Los Angeles'*
- *[180, 173, 191]*

Make sure you add the **space** on both sides of the equal sign!

VARIABLE ASSIGNMENT

Variables are containers used to label and store values.

- They can store any Python data type (and even calls to functions!)

EXAMPLE *Creating a price variable*

```
price = 5

print(price)
```

```
5
```

*We're creating a variable named **price**and assigning it a value of 5, then we're printing the variable, returning its value*

TIP: Only assign variablesfor values that can change or will be used repeatedly; if you're not sure, it's likely a good idea to assign a variable

```
price = 5

print(price + 1)
```

```
6
```

Any operation that can be performed on the value of a variable can be performed using the variable name

*Here we're printing the result of adding 1, a hard coded value, to the **price***

VARIABLE ASSIGNMENT

Variables are containers used to label and store values for future reference

- They can store any Python data type (and even calls to functions!)

EXAMPLE *Creating a price list variable*

```
price_list = [2.50, 4.99, 10, None, 'PROMO']

print_price_list = print(price_list)

print_price_list
```

```
[2.5, 4.99, 10, None, 'PROMO']
```

*First, we're creating a variable named **price_list** and assigning it to list of values*

*Then we're assigning a call to the print function with our **price_list**variable as input*

Note that assigning the print() function to a variable here doesn't necessarily improve our program over simply calling print() outside of it, but it's worth knowing **even a function call can be assigned to a variable**

OVERWRITING VARIABLES

You can **overwrite a variable**by assigninga new value to it

- They can be overwritten any number of times

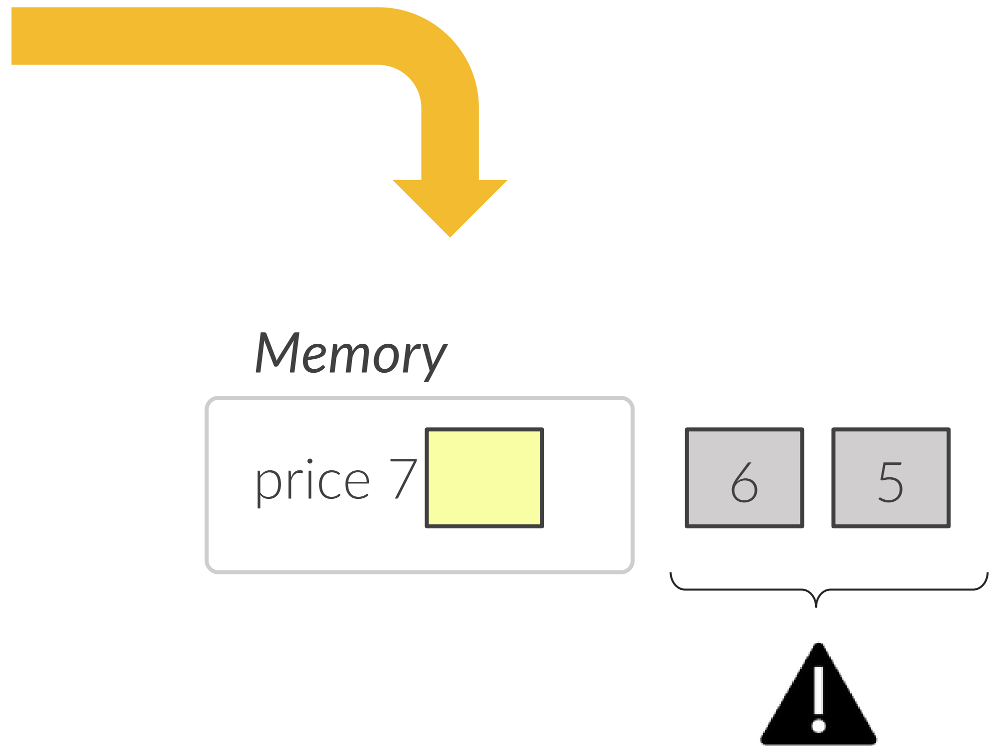
```
price = 5
price = 6
price = 7

print(price)
```

```
7
```

*Python stores the value of 5 in memory when it is assigned to **price***

Then price stores the value 6, and 5 gets removed from memory

Then price stores the value 7, and 6 gets removed from memory



When you overwrite a variable, **its previous value will be lost** and cannot be retrieved

```
price = 5
new_price = 6

print(price, new_price)
```

```
5 6
```

Consider creating a new variable for a new value rather than overwriting

OVERWRITING VARIABLES

You can **overwrite a variable**by assigning a new value to it

- They can be overwritten any number of times

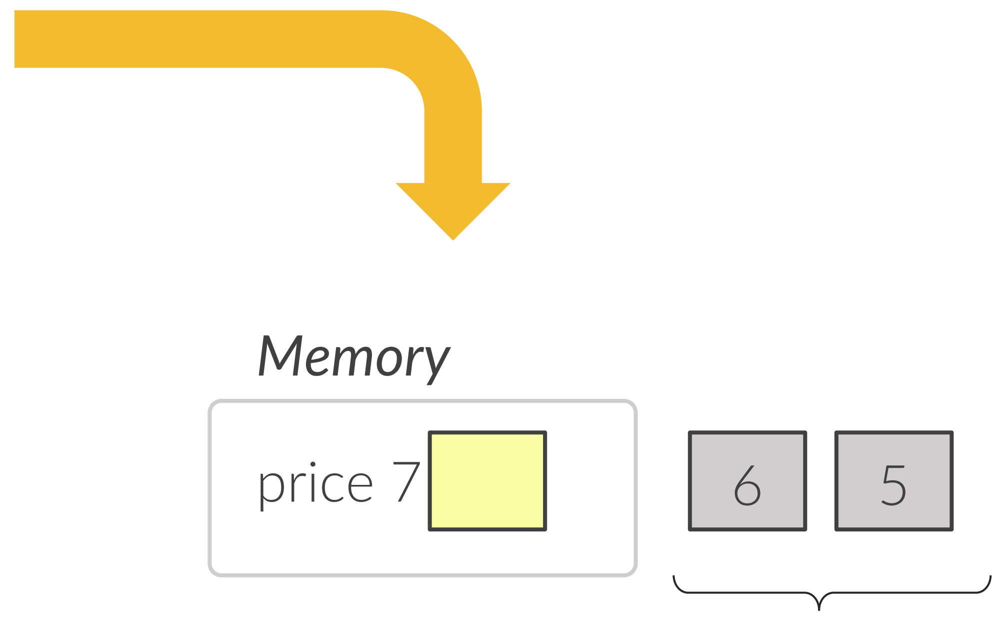
```
price = 5
price = 6
price = 7

print(price)
```

```
7
```

Python stores the value of 5 in memory when it is assigned to ***price***

Then price stores the value 6, and 5 gets removed from memory

Then price stores the value 7, and 6 gets removed from memory



```
old_price = 5
price = 6

print(old_price, price)
```

```
5 6
```

Or create a new variable for the old value

When you overwrite a variable, **its previous value will be lost** and cannot be retrieved

OVERWRITING VARIABLES

You can **overwrite a variable** by assigning a new value to it

- They can be overwritten any number of times

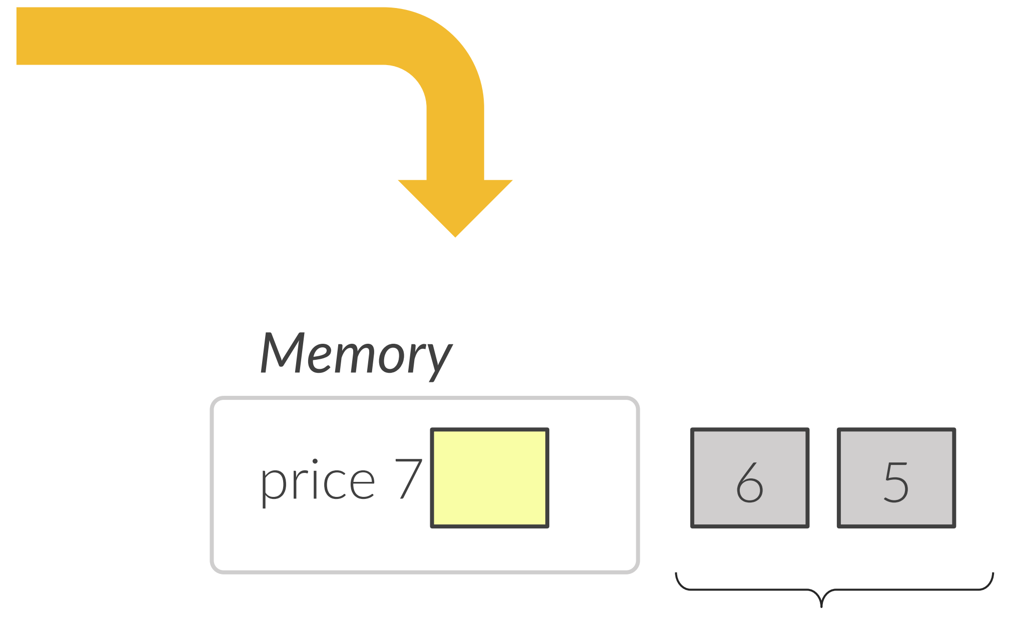
```
price = 5
price = 6
price = 7

print(price)
```

```
7
```

*Python stores the value of 5 in memory when it is assigned to **price***

Then price stores the value 6, and 5 gets removed from memory

Then price stores the value 7, and 6 gets removed from memory



When you overwrite a variable, **its previous value will be lost** and cannot be retrieved

TIP: Use variables like 'new_price' and 'old_price' when testing programs with different values to make sure you don't lose important data –once the code works as needed, remove any extra variables!

OVERWRITING VARIABLES

Variables can also be **assigned as values to other variables**

- The underlying value is assigned, but there is no remaining association between the variables

```python
price = 5
new_price = 6
price = new_price
new_price = 7

print(price)
```

```
6
```

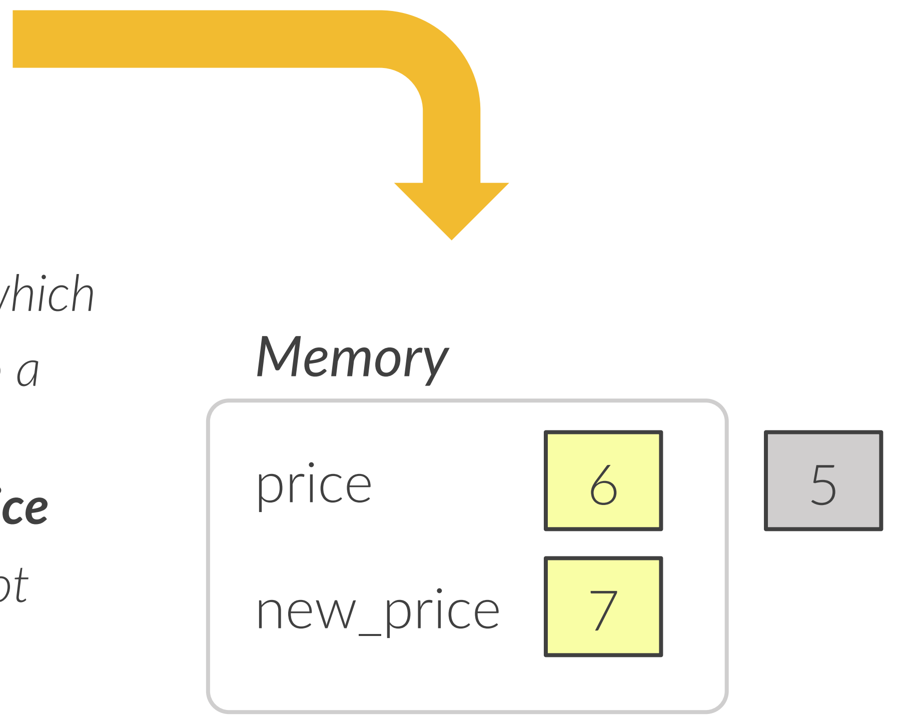
*Python stores the value of 5 in memory when it is assigned to **price***

*Then stores the value of 6 when it is assigned to **new_price***

*Then assigns the value of **new_price**, which is 6, to **price,** 5 is no longer assigned to a variable, so gets removed*

*Then assigns the value of 7 to **new_price***

*Note that **price** is still equal to 6, it's not tied to the value of**new_price**!*



DELETING VARIABLES

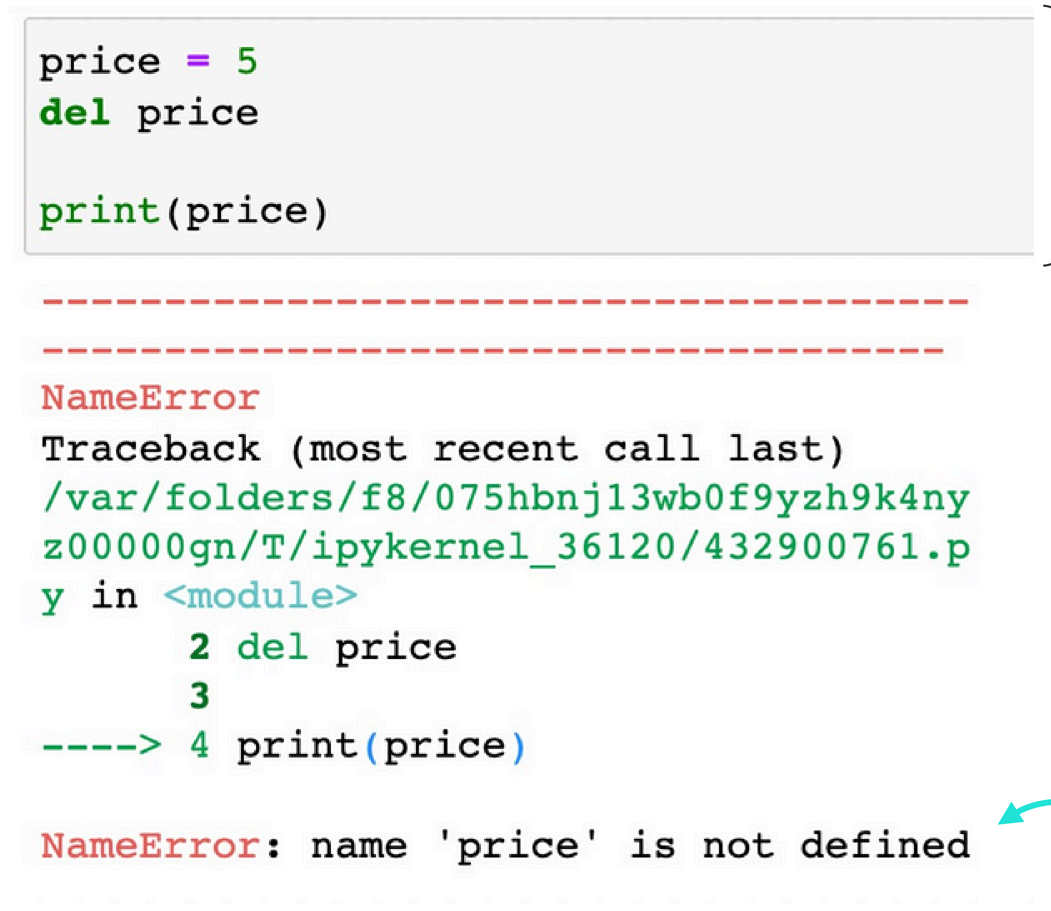
The **del** keyword will permanently remove variables and other objects

Variable Assignment

Overwriting & Deleting

Naming Conventions

Tracking Variables



```
price = 5
del price

print(price)
```

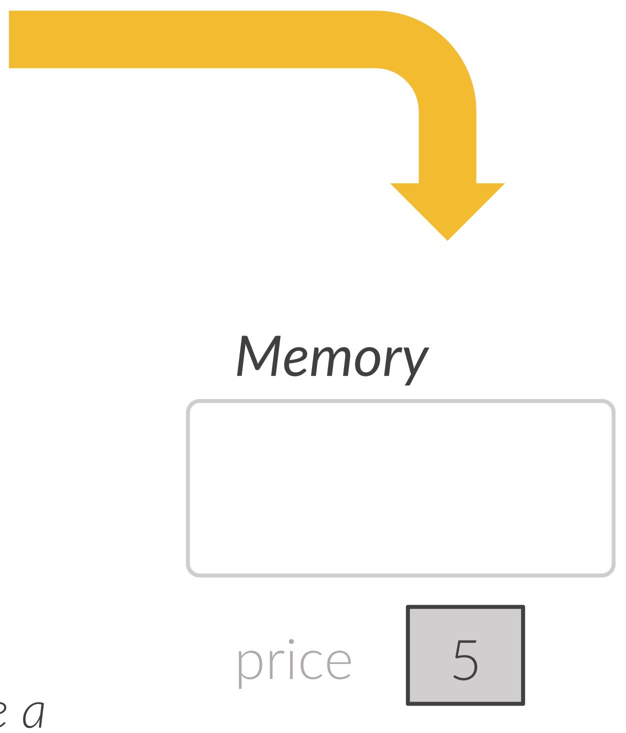
```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
/var/folders/f8/075hbnj13wb0f9yzh9k4nyz00000gn/T/ipykernel_36120/432900761.py in <module>
      2 del price
      3
----> 4 print(price)

NameError: name 'price' is not defined
```

*Python stores the value of 5 in memory when it is assigned to **price***

*Then **del** removes it from memory*

*When we try to print **price**, we get an **Error**, as the variable no longer exists*

***NameError** occurs when we reference a variable or object name that isn't defined*



TIP: Deleting objects is generally unnecessary, and mostly used for large objects (like datasets with 10k+ rows); in most cases, reassign the variables instead and Python will get rid of the old value!

NAMING RULES

Variables have some basic **naming rules**

Variable names **can**:

- Contain letters (case sensitive!)
  Contain numbers
- Contain underscores
- Begin with a letter or underscore

Variable names **cannot**:

- Begin with a number
- Contain spaces or other special characters (*, &, ^, -, *etc.*)
- Be reserved Python keywords

  like **del**or **list**

TIP:"Snake case" is the recommended naming style for Python variables, which is all lowercase with words separated by underscores (first_second_third, new_price, etc.)

NAMING RULES

Variable Assignment

Overwriting & Deleting

Naming Conventions

Tracking Variables

Variables have some basic **naming rules**

Valid variable names:

- price_list_2019
- _price_list_2019
- PRICE_LIST_2019
- pl2019

Invalid variable names

2019_price_list	*(starts with a number)*
price_list-2019	*(has special characters)*
2019 price list	*(has spaces)*
list	*(reserved Python keyword)*

TIP: Give your variables intuitive names to make understanding your code easier –instead of PL19, consider something like price_list_2019 or prices_2019

TRACKING VARIABLES

Use “%who” and “%whos” to **track the variables** you’ve created

```
price = 10
product = 'Super Snowboard'
Date = '10-Jan-2021'
dimensions = [160, 25, 2]
```

```
%who
```

```
Date        dimensions        price    product
```

%who *returns variable names*

```
%whos
```

```
Variable      Type    Data/Info
------------------------------
Date          str     10-Jan-2021
dimensions    list    n=3
price         int     10
product       str     Super Snowboard
```

%whos *returns variable names, types, and information on the data contained*

Magic commands that start with “%” only work in the iPythonenvironments, which applies to Jupyterand Colab

KEY TAKEAWAYS

✓ Variables are containers used to **store values** from any data type

These values are stored in memory and can be referenced repeatedly in your code

✓ **Overwrite variables** by assigning new values to them

The old value will be lost unless it's assigned to another variable

✓ Variable names must follow Python's **naming rules**

"Snake case" is the recommended naming style (all lowercase with underscores separating each word)

✓ Give variables **intuitive** names

Even though you can use magic commands to track variables, good names save a lot of time & confusion