🐍 Python for Data Science Crash Course

Welcome!

---

1. The Basics: Operations, Variables & Types

In [25]:
```
#Comments starting with a hash symbol are ignored by the Python interpreter.

# This is a comment
print("Hello, World!")  # This comment is inline with code

'''This is a multi-line comment
that spans several lines.'''
```

```
Hello, World!
```

Out[25]: `'This is a multi-line comment\nthat spans several lines.'`

In [26]:
```
#Arithmetic operations

1+1 # addition
2*3 # multiplication
4/2 # division
5-3 # subtraction
2**3 # exponentiation
10%3 # modulus
10//3 # floor division
```

Out[26]: `3`

In [27]:
```
#you can see only the last line's output in a Jupyter notebook cell
#if you want to see the output of multiple lines, use the print() function
print(1+1) # addition
print(2*3) # multiplication
print(4/2) # division
print(5-3) # subtraction
print(2**3) # exponentiation
print(10%3) # modulus
print(10//3) # floor division
```

```
2
6
2.0
2
8
1
3
```

In [28]:
```python
#functions like print() are built-in functions in Python that perform specific
#you can find the list of built-in functions in the Python documentation.

#important built-in functions for now:
# len()       returns the length of an object
# type()      returns the type of an object
# str()       converts an object to a string
# int()       converts an object to an integer
# float()     converts an object to a floating-point number
# print()     prints output to the console
# input()     reads input from the user
# range()     generates a sequence of numbers
```

In [29]:
```python
#Variables are used to store data values. In Python, you don't need to declare
# Assigning values to variables is done using the equals sign (=).



# 1. Integer (Whole Numbers)
sales = 50



# 2. Float (Decimal Numbers - use a dot!)
price = 19.99
#floats are sometimes tricky due to how they are represented in memory (precis
print("float precision issue:", 1.1 + 2.2 == 3.3) # This  returns False due to


# 3. String (Text - use quotes)
product_name = "Wireless Mouse"

# Lets do some operations with these variables
total_revenue = sales * price # Calculate total revenue and store it in a new
print("Total Revenue: $", total_revenue) # Print the total revenue

#mixing data types in operations can lead to errors or unexpected results. For
#Also the behavior of certain operations may vary based on the data types invo
```

```
float precision issue: False
Total Revenue: $ 999.4999999999999
```

In [30]:
```python
#string operations
greeting = "Hello"
name = "Alice"
full_greeting = greeting + ", " + name + "!"  # Concatenation
print(full_greeting)  # Output: Hello, Alice!

# You can also use f-strings for formatted output
age = 30
print(f"{name} is {age} years old.")  # Output: Alice is 30 years old.

#more string operations
message = "  Welcome to Python Programming!  "
print(message.lower())        # Convert to lowercase
print(message.upper())        # Convert to uppercase
```

```
print(message.strip())        # Remove leading and trailing whitespace
print(message.replace("Python", "Data Science"))  # Replace substring
print(message.split())       # Split the string into a list of words
print(len(message))          # Get the length of the string

#when we do string operations, they always return a new string and do not modi

#so if you want to keep the changes, you need to assign the result back to a v
message_stripped = message.strip()  # Now message has the trimmed string
print("Original message:", message)  # Original string remains unchanged
print("Stripped message:", message_stripped)  # New variable has the trimmed s
```

```
Hello, Alice!
Alice is 30 years old.
  welcome to python programming!
  WELCOME TO PYTHON PROGRAMMING!
Welcome to Python Programming!
  Welcome to Data Science Programming!
['Welcome', 'to', 'Python', 'Programming!']
34
Original message:   Welcome to Python Programming!
Stripped message: Welcome to Python Programming!
```

In [31]:
```
#more built-in functions for data types
print(type(sales))          # type() returns the type of an object
print(str(price))           # str() converts an object to a string
print(int(sales))          # int() converts an object to an integer
print(float(sales))        # float() converts an object to a floating-point nu
print(range(5))            # range() generates a sequence of numbers
 # len() returns the length of an object
```

```
<class 'int'>
19.99
50
50.0
range(0, 5)
```

---

2. Sequences (Data Structures)

In Data Science, we rarely work with single numbers. We work with **lists** of numbers (like an Excel column).

- **Important:** Python starts counting at **0**.

In [32]:
```
# Creating a list of daily users
daily_users = [120, 150, 90, 200, 180]

# Accessing data
print(f"First day users (Index 0): {daily_users[0]}")
print(f"Third day users (Index 2): {daily_users[2]}")
```

```python
# first Index is always 0 and last is always length -1

# Adding new data
daily_users.append(210)
print("Updated List:", daily_users)

#more list operations
print("Length of the list:", len(daily_users))  # Get the length of the list
print("Maximum users in a day:", max(daily_users))  # Get the maximum value
print("Minimum users in a day:", min(daily_users))  # Get the minimum value
print("Sum of users over the week:", sum(daily_users))  # Get the sum of all v
daily_users.sort()  # Sort the list in ascending order
print("Sorted List:", daily_users)
daily_users.reverse()  # Reverse the list
print("Reversed List:", daily_users)

#unlike strings, lists are mutable, meaning you can change their content witho
print(daily_users) #is reversed since this was the last operation done on it
```

```
First day users (Index 0): 120
Third day users (Index 2): 90
Updated List: [120, 150, 90, 200, 180, 210]
Length of the list: 6
Maximum users in a day: 210
Minimum users in a day: 90
Sum of users over the week: 950
Sorted List: [90, 120, 150, 180, 200, 210]
Reversed List: [210, 200, 180, 150, 120, 90]
[210, 200, 180, 150, 120, 90]
```

In [33]:

```python
#lists cam store mixed data types
mixed_list = [1, "Hello", 3.14, True]
print(mixed_list)

#multidimensional lists (lists of lists)
matrix = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
]
print("Matrix:", matrix)
print("Element at row 1, column 2:", matrix[1][2])  # Accessing element '6'

#tuples are similar to lists, but they are immutable (cannot be changed after
coordinates = (10.0, 20.0)
print("Coordinates:", coordinates)
# Trying to modify a tuple will raise an error
# therefore we don't have methods like append() or remove() for tuples
```

```
[1, 'Hello', 3.14, True]
Matrix: [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
Element at row 1, column 2: 6
Coordinates: (10.0, 20.0)
```

In [34]:
```
#Dictionaries store data in key-value pairs.
student = {
    "name": "John Doe",
    "age": 21,
    "major": "Computer Science"
}
# Accessing values
print("Student Name:", student["name"])

# Modifying values
student["age"] = 22  # Update age
print("Updated Age:", student["age"])
# Adding new key-value pair
student["graduation_year"] = 2024
print("Updated Student Info:", student)
# Removing a key-value pair
del student["major"]
print("After Deletion:", student)
```

```
Student Name: John Doe
Updated Age: 22
Updated Student Info: {'name': 'John Doe', 'age': 22, 'major': 'Computer Scienc
e', 'graduation_year': 2024}
After Deletion: {'name': 'John Doe', 'age': 22, 'graduation_year': 2024}
```

In [35]:
```
#Sets are unordered collections of unique items.
fruits = {"apple", "banana", "orange"}
print("Fruits Set:", fruits)
# Adding an item
fruits.add("grape")
print("After Adding Grape:", fruits)
# Removing an item
fruits.remove("banana")
print("After Removing Banana:", fruits)
# Sets automatically handle duplicates
fruits.add("apple")  # Trying to add a duplicate
print("After Adding Duplicate Apple:", fruits)
#Sets support mathematical operations like union, intersection, and difference
more_fruits = {"banana", "kiwi", "mango"}
all_fruits = fruits.union(more_fruits)
print("Union of Fruits:", all_fruits)
common_fruits = fruits.intersection(more_fruits)
print("Intersection of Fruits:", common_fruits)
```

```
Fruits Set: {'apple', 'orange', 'banana'}
After Adding Grape: {'grape', 'apple', 'orange', 'banana'}
After Removing Banana: {'grape', 'apple', 'orange'}
After Adding Duplicate Apple: {'grape', 'apple', 'orange'}
Union of Fruits: {'apple', 'orange', 'kiwi', 'grape', 'mango', 'banana'}
Intersection of Fruits: set()
```

3. Logic: Loops & Conditions

We use computers to automate boring tasks.

- **For-Loop:** "Do this for every item in the list."
- **If-Condition:** "Only do this if X is true."

In [36]:
```
#for loops are used to iterate over a sequence (like a list, tuple, dictionary
# They allow you to execute a block of code multiple times, once for each item
for i in range(5):
    print(f"Iteration {i}")

# In this example, the loop iterates over a sequence of numbers generated by r
# For each iteration, the current number is printed.
```

```
Iteration 0
Iteration 1
Iteration 2
Iteration 3
Iteration 4
```

In [37]:
```
#if statements are used for conditional execution of code blocks.

x=10
if x > 5:
    print(f"{x} is greater than 5")

#you can add an else statement to execute code when the condition is false.
if x > 15:
    print(f"{x} is greater than 15")
else:
    print(f"{x} is not greater than 15")

#multiple conditions can be checked using elif statements.
if x > 15:
    print(f"{x} is greater than 15")
elif x < 5:
    print(f"{x} is less than 5")
else:
    print(f"{x} is between 5 and 15")
```

```
10 is greater than 5
10 is not greater than 15
10 is between 5 and 15
```

In [38]:
```
#whats happening is that x>5 is evaluated to True.
# True is a boolean value, which is a data type that can be either True or Fal

print(1==1) # True --> the single equal sign (=) is used for assignment, while
print(1==2) # False
print(1!=2) # True
print(5 > 3)  # True
```

```python
print(2 < 1)  # False
print(3 >= 3) # True
print(2 <= 1) # False
# You can combine multiple conditions using logical operators: and, or, not
print((5 > 3) and (2 < 4))  # True
print((5 > 3) or (2 > 4))   # True
print(not (5 > 3))          # False
```

```
True
False
True
True
False
True
False
True
True
False
```

In [39]:

```python
#scenario: Find all days with high traffic (> 140 users)

for user_count in daily_users:
    # Check the condition
    if user_count > 140:
        print(f"High traffic detected: {user_count} users")
```

```
High traffic detected: 210 users
High traffic detected: 200 users
High traffic detected: 180 users
High traffic detected: 150 users
```

In [40]:

```python
#nested loops
for i in range(3):  # Outer loop
    for j in range(2):  # Inner loop
        print(f"i: {i}, j: {j}")
```

```
i: 0, j: 0
i: 0, j: 1
i: 1, j: 0
i: 1, j: 1
i: 2, j: 0
i: 2, j: 1
```

In [41]:

```python
#While loops are used to repeatedly execute a block of code as long as a speci

i = 0
while i < 5:
    print(f"Iteration {i}")
    i += 1

# if the condition never becomes false, the loop will continue indefinitely, l
```

```
Iteration 0
Iteration 1
Iteration 2
Iteration 3
Iteration 4
```

In [42]:
```python
#list comprehensions provide a concise way to create lists.
squares = [x**2 for x in range(10)] # structure: [expression for item in itera
print(squares)  # Output: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]

#you can have multiple conditions and expressions as well.
x = [1, 2]
y = [5, 6]

# Using two for loops inside a list comprehension
ans = [a+b for a in x for b in y if (a + b) % 2 == 0]

print(ans)
```

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
[6, 8]
```

---

4. Functions

DRY Principle: Don't Repeat Yourself. Functions are reusable code blocks (like a custom formula in Excel).

In [43]:
```python
# Define the function
def calculate_vat(amount):
    vat_rate = 0.20 # 20% Tax
    tax = amount * vat_rate
    return tax

# Use the function
price_laptop = 1000
price_phone = 500

print(f"Tax for Laptop: {calculate_vat(price_laptop)}")
print(f"Tax for Phone: {calculate_vat(price_phone)}")
# Functions help in organizing code into reusable blocks, making it easier to
# They can take inputs (parameters) and return outputs (return values).
```

```
Tax for Laptop: 200.0
Tax for Phone: 100.0
```

In [44]:
```python
# Functions can also have default parameter values, allowing you to call them


def calculate_vat_with_default(amount, vat_rate=0.20):
    tax = amount * vat_rate
```

```
    return tax
print(f"Tax for Laptop with default VAT: {calculate_vat_with_default(price_lap
print(f"Tax for Phone with custom VAT: {calculate_vat_with_default(price_phone
```

```
Tax for Laptop with default VAT: 200.0
Tax for Phone with custom VAT: 75.0
```

---

5. Data Science Power: Pandas

Now we use **Pandas**. This is the standard tool for analyzing tabular data (DataFrames).

In [45]:
```
import pandas as pd # Import the library
```

In [47]:
```
#if you don't have pandas installed, you can install it using pip:
# %pip install pandas

# if we want to use function from a module we need to write the module name fo
```

In [48]:
```
# Creating a dataset
data = {
    'Date': ['Mon', 'Tue', 'Wed', 'Thu', 'Fri'],
    'Sales': [1200, 1500, 900, 2000, 1800],
    'Visitors': [300, 350, 200, 500, 450]
}

# Convert to DataFrame (Table)
df = pd.DataFrame(data)

# Show the table
display(df)
```

	Date	Sales	Visitors
0	Mon	1200	300
1	Tue	1500	350
2	Wed	900	200
3	Thu	2000	500
4	Fri	1800	450

In [49]:
```
# 1. Quick Statistics
print("Statistics:")
display(df.describe())

# 2. Calculation (Conversion Rate)
```

```python
# We can do math on whole columns at once!
df['Conversion_Rate'] = df['Sales'] / df['Visitors']

print("\nUpdated Table:")
display(df)
```

```
Statistics:
```

	Sales	Visitors
count	5.00000	5.000000
mean	1480.00000	360.000000
std	443.84682	119.373364
min	900.00000	200.000000
25%	1200.00000	300.000000
50%	1500.00000	350.000000
75%	1800.00000	450.000000
max	2000.00000	500.000000

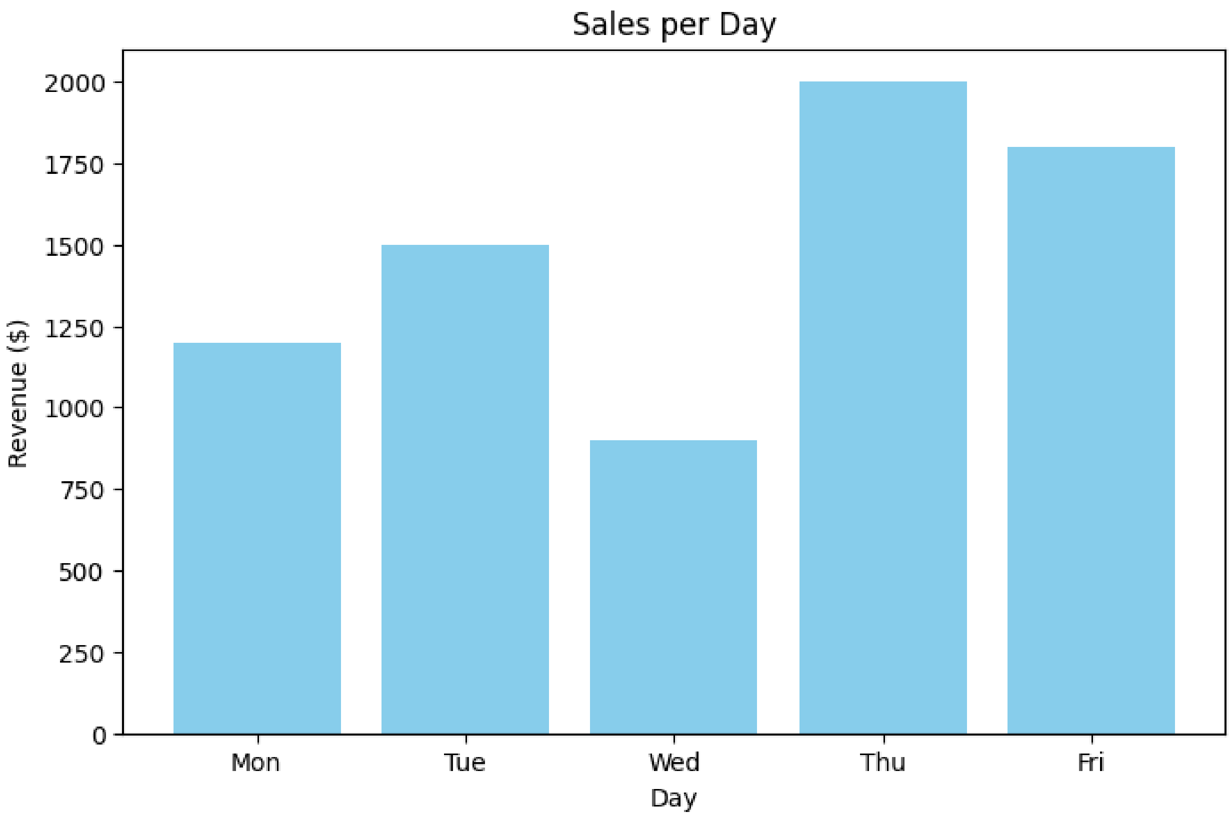
```
Updated Table:
```

	Date	Sales	Visitors	Conversion_Rate
0	Mon	1200	300	4.000000
1	Tue	1500	350	4.285714
2	Wed	900	200	4.500000
3	Thu	2000	500	4.000000
4	Fri	1800	450	4.000000

In [52]:

```python
# 3. Visualization with Matplotlib
import matplotlib.pyplot as plt

plt.figure(figsize=(8, 5))
plt.bar(df['Date'], df['Sales'], color='skyblue')
plt.title("Sales per Day")
plt.xlabel("Day")
plt.ylabel("Revenue ($)")
plt.show()
```



In [53]:
```
#save the DataFrame to a CSV file
df.to_csv('sales_data.csv', index=False)

#load a CSV file into a DataFrame
df = pd.read_csv('sales_data.csv')
display(df)
#be aware that the CSV file should be in the same directory as your script or
```

	Date	Sales	Visitors	Conversion_Rate
0	Mon	1200	300	4.000000
1	Tue	1500	350	4.285714
2	Wed	900	200	4.500000
3	Thu	2000	500	4.000000
4	Fri	1800	450	4.000000

Congratulations! 🚀

You just completed your first data science workflow.