DATA SCIENCE BOOTCAMP

PYTHON SESSION

TABLE OF CONTENT

WHAT LANGUAGE CAN WE USE IN DATA SCIENCE TO SOLVE PROBLEMS?

IN DATA SCIENCE, WE USE PROGRAMMING LANGUAGES LIKE PYTHON TO SOLVE PROBLEMS. FOR EXAMPLE, IF YOU WANT TO KNOW THE AVERAGE SCORE OF STUDENTS IN A CLASS, YOU CAN WRITE A SMALL PYTHON PROGRAM TO CALCULATE IT. YOU WOULD USE PYTHON TO ADD UP ALL THE SCORES AND DIVIDE BY THE NUMBER OF STUDENTS. PYTHON IS GREAT BECAUSE IT HAS TOOLS (CALLED LIBRARIES) THAT MAKE TASKS LIKE THIS EASIER, SUCH AS PANDAS FOR HANDLING DATA. SIMILARLY, WE USE SQL TO GET DATA FROM DATABASES, LIKE FINDING ALL STUDENTS WHO SCORED ABOVE 80.

INTRODUCTION TO PYTHON

PYTHON HAS BECOME AN INDISPENSABLE TOOL IN THE FIELD OF DATA SCIENCE DUE TO ITS VERSATILITY, READABILITY, AND EXTENSIVE LIBRARIES. ITS INTUITIVE SYNTAX AND EASE OF LEARNING MAKE IT ACCESSIBLE TO BOTH SEASONED PROGRAMMERS AND NEWCOMERS, ALLOWING FOR RAPID PROTOTYPING AND EXPERIMENTATION WITH DATA-DRIVEN PROJECTS. ADDITIONALLY, PYTHON BOASTS A RICH ECOSYSTEM OF LIBRARIES, SUCH AS NUMPY, PANDAS, MATPLOTLIB, AND SCIKIT-LEARN, WHICH PROVIDE POWERFUL FUNCTIONALITIES FOR DATA MANIPULATION, ANALYSIS, VISUALIZATION, AND MACHINE LEARNING. FOR INSTANCE, A DATA SCIENTIST MIGHT USE NUMPY TO PERFORM COMPLEX MATHEMATICAL OPERATIONS ON LARGE DATASETS, PANDAS TO CLEAN AND PREPROCESS DATA, MATPLOTLIB TO CREATE INFORMATIVE VISUALIZATIONS, AND SCIKIT-LEARN TO BUILD AND TRAIN MACHINE LEARNING MODELS.

WHAT ALL THINGS WE NEED TO LEARN PYTHON?

TOPICS COVERED:

VARIABLES AND DATA TYPES

TYPE CONVERSIONS

LISTS AND DICTIONARIES

CONTROL STRUCTURES

LOOPS

FUNCTIONS

LIST COMPREHENSIONS

VARIABLES AND DATA TYPES

VARIABLES IN PYTHON ARE CONTAINERS FOR STORING DATA VALUES.
THE TYPE OF A VARIABLE DEPENDS ON THE DATA ASSIGNED TO IT.

INT: WHOLE NUMBERS (E.G., 10, -5).
FLOAT: DECIMAL NUMBERS (E.G., 3.14, 2.5).
STR: TEXT (E.G., "HELLO", "WORLD").

```
X = 10    # INT
Y = 3.14  # FLOAT
NAME = "ZEESHAN"  # STRING
PRINT(X, Y, NAME)
```

```
PRINT(TYPE(X))   # OUTPUT: <CLASS 'INT'>
PRINT(TYPE(Y))   # OUTPUT: <CLASS 'FLOAT'>
PRINT(TYPE(NAME)) # OUTPUT: <CLASS 'STR'>
```

TYPE CONVERSIONS

SOMETIMES DATA NEEDS TO BE CONVERTED FROM ONE TYPE TO ANOTHER
(E.G., FROM A STRING TO AN INTEGER)

```
A = "5"
B = INT(A)    # CONVERT STRING TO INT
C = FLOAT(A)  # CONVERT STRING TO FLOAT
PRINT(B, C)   # OUTPUT: 5 5.0
```

IMPORTANT FUNCTIONS:
INT(): CONVERTS TO INTEGER
FLOAT(): CONVERTS TO FLOAT
STR(): CONVERTS TO STRING

```
AGE = INPUT("ENTER YOUR AGE: ")  # USER INPUTS STRING
AGE = INT(AGE)  # CONVERT TO INTEGER
PRINT("YOUR AGE IS", AGE)
```

LISTS AND DICTIONARIES

LISTS AND DICTIONARIES ARE USED TO STORE MULTIPLE ITEMS.

LISTS STORE ITEMS IN A SPECIFIC ORDER, WHILE DICTIONARIES STORE KEY-VALUE PAIRS.

CREATING AND USING LISTS

```
FRUITS = ["APPLE", "BANANA", "CHERRY"]
PRINT(FRUITS[0])  # OUTPUT: APPLE
FRUITS.APPEND("ORANGE")  # ADDING NEW ELEMENT
PRINT(FRUITS)  # OUTPUT: ['APPLE', 'BANANA', 'CHERRY', 'ORANGE']
```

CREATING AND USING DICTIONARIES

```
PERSON = {"NAME": "ZEESHAN", "AGE": 25, "CITY": "LONDON"}
PRINT(PERSON["NAME"])  # OUTPUT: ZEESHAN
PERSON["JOB"] = "DATA SCIENTIST"  # ADDING NEW KEY-VALUE PAIR
PRINT(PERSON)
```

CONTROL STRUCTURES

CONTROL STRUCTURES ALLOW YOU TO MAKE DECISIONS IN YOUR CODE.

```
AGE = 18
IF AGE >= 18:
    PRINT("YOU ARE AN ADULT.")
ELIF AGE >= 13:
    PRINT("YOU ARE A TEENAGER.")
ELSE:
    PRINT("YOU ARE A CHILD.")
```

```
AGE = 18
IF AGE < 18:
    PRINT("MINOR")
ELIF AGE == 18:
    PRINT("JUST TURNED 18!")
ELSE:
    PRINT("ADULT")
```

LOOPS

LOOPS ARE USED TO REPEAT A BLOCK OF CODE MULTIPLE TIMES.

FOR LOOP: ITERATE OVER A SEQUENCE.

```
FRUITS = ["APPLE", "BANANA", "ORANGE"]
FOR FRUIT IN FRUITS:
    PRINT(FRUIT)

FOR I IN RANGE(5):
    PRINT(I)  # OUTPUT: 0 1 2 3 4
```

WHILE LOOP: REPEAT AS LONG AS A CONDITION IS TRUE.

```
COUNT = 0
WHILE COUNT < 5:
    PRINT(COUNT)
    COUNT += 1
```

FUNCTIONS

FUNCTIONS ARE REUSABLE BLOCKS OF CODE.

DEFINING AND CALLING A FUNCTION

```
DEF GREET(NAME):
    RETURN "HELLO, " + NAME
PRINT(GREET("ZEESHAN"))  # OUTPUT: HELLO, ZEESHAN
```

FUNCTION WITH MULTIPLE ARGUMENTS

```
DEF ADD(A, B):
    RETURN A + B
PRINT(ADD(3, 5))  # OUTPUT: 8
```

LIST COMPREHENSIONS

LIST COMPREHENSION IS A SHORTCUT TO CREATE LISTS IN A SIMPLER AND FASTER WAY. IT'S LIKE TAKING A SHORTCUT WHEN YOU WANT TO DO SOMETHING QUICKLY.

INSTEAD OF WRITING A LOT OF LINES TO CREATE OR CHANGE A LIST, YOU CAN DO IT IN ONE SIMPLE LINE USING LIST COMPREHENSION

```
NUMBERS = [1, 2, 3, 4, 5]
DOUBLED_NUMBERS = []
FOR NUM IN NUMBERS:
    DOUBLED_NUMBERS.APPEND(NUM * 2)

PRINT(DOUBLED_NUMBERS)  # OUTPUT: [2, 4, 6, 8, 10]


DOUBLED_NUMBERS = [NUM * 2 FOR NUM IN [1, 2, 3, 4, 5]]
  PRINT(DOUBLED_NUMBERS)  # OUTPUT: [2, 4, 6, 8, 10]
```

Q&A

THANK YOU!