Bit Project

Introduction to Python

Module 1: Part 1

Introduction to Data Science & Programming

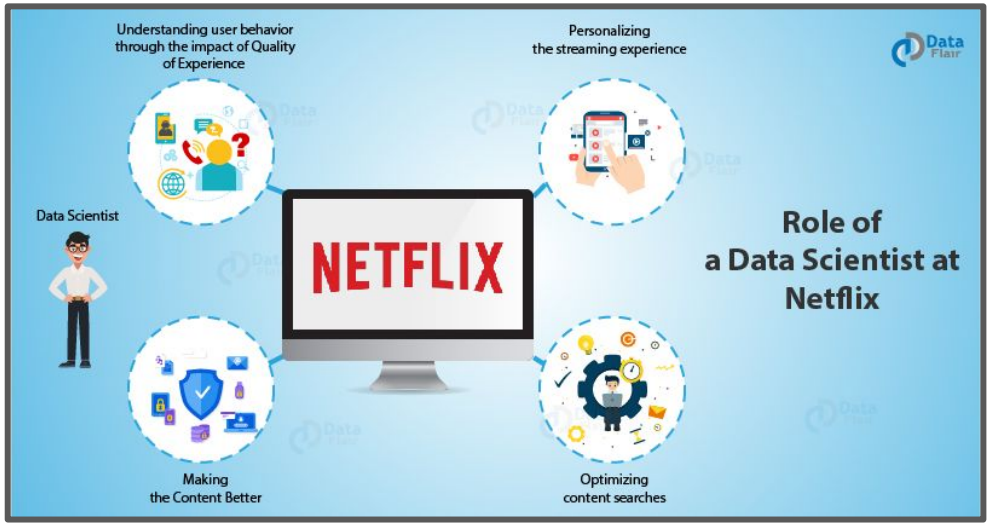
Why is it important?

What is Data Science?

- Learn programming tools to **analyze data**
  - **Sort** and **filter data**
  - Generate **visualizations**
  - Form **meaningful conclusions** and **analyses**
- Can be applicable to everyday life

Real-Life Applications

- Netflix
- Healthcare Providers
- Stockbrokers
- More!



What is Programming?

- Easily and efficiently analyze data
  - Filter out irrelevant data
  - Select specific data
  - Visualize text-based information
- Examples of Programming languages:

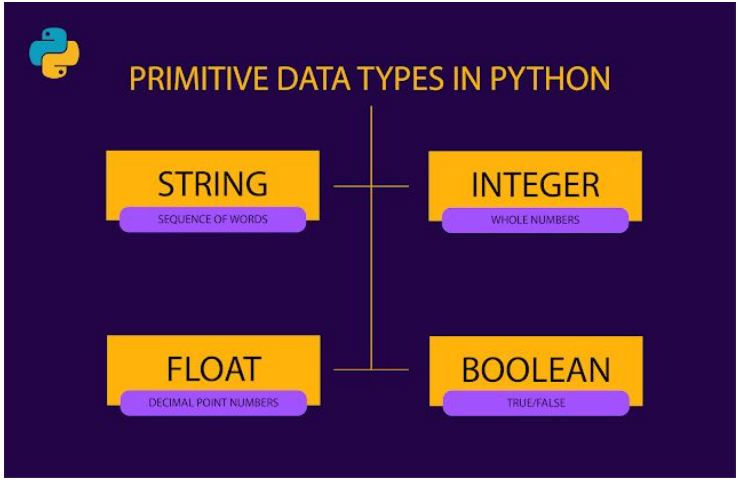
Why, Where, and How we use Python

- Popular, **beginner-friendly** programming language
  - **Easy to read / write**
  - Many **data science libraries**
- Used for data science, software engineering, web development, etc

Introduction to Python

Let’s get started!

Overview



1. Numbers & Variables
2. Strings
3. Booleans & Comparison Operators
4. If-Else Statements
5. Lists

Numbers

Types of Numbers

- **Integers** are whole numbers
  - `3 ,  100 ,  -2000`
- **Floats** are numbers with decimals
  - `-3.14 ,  2.917 ,  1.1`

Basic Arithmetic

- Addition
- Subtraction
- Multiplication
- Division

```
4 + 5       # 9

10 - 3      # 7

7 * 2       # 14

36 / 9      # 4
```

Basic Arithmetic

- **Floor Division**
  - Returns **quotient** as a **whole number**

```
17 // 7          # 17 // 7 = 2

21 % 4           # 21 % 4 = 1
```

- **Modulo**
  - Returns **remainder**

Variables

Variable Assignment

- **Variables** are used to **store data**

- **Variable assignment** is done with the **“=” operator**
  - **Variable reassignment** changes an existing variable’s value

```
a = 10        # Assignment
b = 20
c = 30

a = 20        # Reassignment
```

Operations with Variables

- We can perform **arithmetic operations** with **variables**

```
a = 10        # Assignment
a + a         # Doesn't change a's value

a = a + a         # Now reassigned a = 20
```

Variable Naming Rules

1. Names **can't start with a number.** (E.g. `123name` is invalid)
2. There **can't be any spaces**. (E.g. `my name` is invalid)
    a. Use **underscores** instead. (E.g. `my_name` is valid)
3. Can't use any of these **restricted symbols**: `'",<>/?|\()!@#$%^&*~-+`
4. **Avoid** using the characters **'`l`'** (lowercase letter el), **'`O`'** (uppercase letter oh), or **'`I`'** (uppercase letter eye) as **single letter** variable names.
5. Avoid using words that have **special meaning** in Python like "`list`" and "`str`".
6. Using **lowercase names** are best practice.

Variable Naming Example

- Use **meaningful variable names!**

```
income = 1000
tax_rate = 0.2
taxes = income * tax_rate

print("Total taxes:", taxes)
# Total taxes: 200
```

A Brief Note About Errors

- Tools to deal with errors:
  - Refer to prior experience
    - What did you do when you encountered this error last time?
  - Manual detection within code
    - Search through code to find the error
  - Google it!
    - Google the error message
- Don’t be discouraged, errors happen!

Strings

Creating Strings

- **Strings** are a combination of **characters**
  - **Characters** are singular **letters, numbers,** or **symbols**
- **Strings** are defined by a **set of quotes**

```
# A word
'hi'

# A phrase
'A string can even be a sentence like this'

# Using double quotes
"Strings can also be defined by double quotes"

# Be careful with contractions or apostrophes!
'I'm using single quotes, but this will create an error'
"This shouldn't create an error now"
```

String Basics

- Python strings have many **built-in properties**
  - **len():** returns length of string
  - **print():** prints content to console

```
len('apple')            # 5
len('Good morning')     # 12

print('Good afternoon')     # Good afternoon
```

String Indexing

- **Indexing** allows us to work with individual characters within a string
- Python strings follow **zero-based indexing**
  - Consider the string `'university'`
  - The indexing is as follows:

Index	0	1	2	3	4	5	6	7	8	9
Character	u	n	i	v	e	r	s	i	t	y

String Indexing

Index	0	1	2	3	4	5	6	7	8	9
Character	u	n	i	v	e	r	s	i	t	y

```
my_string = 'university'

# Forwards indexing
my_string[0]        # u
my_string[1]        # n
my_string[2]        # i
...
my_string[9]        # y

# Reverse indexing
my_string[-1]       # y
my_string[-2]       # t
...
```

String Slicing (1)

- **Format:** `samplestring [ begin_index : end_index ]`

Index	0	1	2	3	4	5	6	7	8	9
Character	u	n	i	v	e	r	s	i	t	y

```
my_string = 'university'

# Grab all the letters
my_string[:]            # university

# Grab all the letters up until index 5
my_string[ :5]          # unive

# Grab everything up until the last letter
my_string[ :-1]         # universit
```

String Slicing (2)

Index	0	1	2	3	4	5	6	7	8	9
Character	u	n	i	v	e	r	s	i	t	y

```
my_string = 'university'

# Grab from index 3 to the end
my_string[3: ]                  # versity

# Grab a segment from index 2 up until index 7
my_string[2:7]                  # ivers
```

String Step Sizes

- **Format:** `samplestring [ begin_index : end_index : step_size ]`

Index	0	1	2	3	4	5	6	7	8	9
Character	u	n	i	v	e	r	s	i	t	y

```
my_string = 'university'

# Grab everything, but go in step sizes of 1
my_string[ : :1]                # university

# Grab everything, but go in step sizes of 2
my_string[0: : 2]               # uiest

# Reverse the string
my_string[ : :-1]               # ytisrevinu
```

String Immutability

- Strings are **immutable**, so **cannot be changed** after creation.

```
my_string = 'university'

# Attempt to change the first letter
my_string[0] = 'a'          # THIS CAUSES AN ERROR
```

String Concatenation

- **Concatenation** combines strings.

```
my_string = 'university'

# Attempt concatenation
print(my_string + 'Cali')           # universityCali
print(my_string)                    # university

# Combine strings through concatenation and reassignment
my_string = my_string + 'Cali'      # universityCali
print(my_string)                    # universityCali
```

Basic Built-In String Methods

- **Built-in methods** usually use **dot notation**
  - Format: `variable.method()`

```
my_string = 'University Of California'

# Make all letters become uppercase
my_string.upper()                # UNIVERSITY OF CALIFORNIA

# Make all letters become lowercase
my_string.lower()                # university of california
```

Now Try This

1. Given the string `'Amsterdam'`, write a Python statement that displays the letter `'d'`.
   a. (Hint: This requires indexing)

2. Reverse the string `'Amsterdam'` using slicing.

3. Given the string `'Amsterdam'`, display the letter `'r'` using negative indexing.

Booleans & Comparison Operators

Creating Booleans

- **Booleans** are a data type that represent **True** or **False**

```
# Booleans can be assigned to variables

game_status = True

game_status = False

# Use None as a placeholder for unknown status
game_status = None
```

Comparison Operators

Operator	Description	Example
==	Checks if two numbers are EQUAL	(a == b) is not true
!=	Checks if two numbers are NOT EQUAL	(a != b) is true
>	Checks if the first number is GREATER THAN the second	(a > b) is not true
<	Checks if the first number is LESS THAN the second	(a < b) is true
>=	Checks if the first number is GREATER OR EQUAL TO the second	(a >= b) is not true
<=	Checks if the first number is LESS OR EQUAL TO the second	(a <= b) is true

Boolean Exercises

```
# Equal
4 == 4                                  # True
1 == 0                                  # False

# Not Equal
4 != 5                                  # True
1 != 1                                  # False

# Greater Than
8 > 3                                   # True
1 > 9                                   # False

# Less Than
3 < 8                                   # True
7 < 0                                   # False

# Greater Than or Equal To
7 >= 7                                  # True
9 >= 4                                  # True

# Less Than or Equal To
4 <= 4                                  # True
1 <= 3                                  # True
```

If - Else Statements

if-else Statements

- Use **If-Else statements** to perform actions under different **conditions**
    - **Let's walk through the syntax**

```
age = 18

if age >= 18:
  print("You can vote!")

else:
  print("You can't vote yet!")
```

elif Statements

- Use **elif statements** for **multiple conditions**

```
age = 18

if age >= 35:
  print("You can run for president.")

elif age >= 18:
  print("You can vote!")

else:
  print("You can't vote yet!")
```

Multiple Conditions

- **Case 1: Both or All conditions must be true** before executing some code
    - Use **AND** operand
- **Case 2: At least one condition must be true** before executing some code
    - Use **OR** operand

Multiple Conditions

- **Case 1: Both or All conditions must be true** before executing some code

```
age = 18

if age >= 18 and citizen == True:
  print("You can vote!")

else:
  print("You can't vote yet!")
```

Multiple Conditions

- **Case 2: At least one condition must be true** before executing some code

```
age = 18

if age < 18 of citizen != True:
  print("You can't vote yet!")

else:
  print("You can vote!")
```

Now Try This

Write a simple program that decides whether you stay dry or wet when going outside. Here's what should happen:

1. If it is raining outside and you have a jacket, print "You can go outside!"
2. If it is raining outside and you don't have a jacket, print "You're gonna get wet!"
3. If it is not raining outside, print "It's a beautiful day!"

Use the following boolean variables:

```
raining = False
jacket = False
```

Lists

Creating Lists

- **Lists** are a data type that can hold other types of data, not just letters or characters
    - Format: `[ a , b , c ]`

```
# Create a list and assign to a variable
my_list = [1, 2, 3]          # List of integers

# Create a list storing a string, integer,
float, and character
my_list = ['A string', 23, 100.234, 'o']
```

List Properties

- **Lists** share many of the **same properties** with **strings**

```
my_list = ['one', 'two', 'three', 4, 5]

len(my_list)              # 5
my_list[0]                # 'one'
my_list[1: ]              # 'two', 'three', 4, 5
my_list[ :3]              # 'one', 'two'

my_list = my_list + ['new item']
    # ['one', 'two', 'three', 4, 5, 'new item']
```

List Mutability

- **Lists** are **mutable,** and can be **changed after creation**

```
my_list = ["Hello", 1.2, "o", True, 5]

# Replacing 'True' with 'False'
my_list[3] = False

print(my_list)      # ["Hello", 1.2, "o", False, 5]
```

Basic List Methods (1)

```python
my_list = [1, 2, 3]

# Add a string to the end of the list
my_list.append('append me!')        # [1, 2, 3, 'append me!']

# Remove the first item from the list
my_list.pop(0)                      # [2, 3, 'append me!']

# Attempt to access element at index 100
my_list[100]                # ERROR: 100th index doesn't exist
```

Basic List Methods (2)

```
list_1 = ['a', 'e', 'x', 'b', 'c']
list_2 = [3, 2, 9, 6, 4, 0]

# Sort the list in alphabetical order
list_1.sort()                  # ['a', 'b', 'c', 'e', 'x']

# Sort the list in ascending numerical order
list_2.sort()                  # [0, 2, 3, 4, 6, 9]

# Sort the list in descending numerical order
list_2.sort(reverse = True)    # [9, 6, 4, 3, 2, 0]

# Reverse the order of the list
list_1.reverse()               # ['x', 'e', 'c', 'b', 'a']
```

Nesting Lists

- **Lists** can be **nested** to store **lists within other lists**
    - This creates a **matrix**

```
# Build three lists
list_1 = [1, 2, 3]
list_2 = [4, 5, 6]
list_3 = [7, 8, 9]

# Nest lists to create a matrix
matrix = [list_1, list_2, list_3]
# [
#   [ 1, 2, 3 ],
#   [ 4, 5, 6 ],
#   [ 7, 8, 9 ],
# ]
```

Nested Lists Indexing

- **Multiple indexes** are needed to access elements in nested lists

```
# Build three lists
list_1 = [1, 2, 3]
list_2 = [4, 5, 6]
list_3 = [7, 8, 9]

# Nest lists to create a matrix
matrix = [list_1, list_2, list_3]
# [
#   [ 1, 2, 3 ],
#   [ 4, 5, 6 ],
#   [ 7, 8, 9 ],
# ]

# Grab the first element in matrix (list_1)
matrix[0]                          # [ 1, 2, 3 ]

# Grab the first element of list_1
matrix[0][0]                       # 1
```

Now Try This

1. Build the list `[0, 0, 0]`.

2. Given `my_nest_list`, modify it with multiple indexing to replace `'hello'` with `'goodbye'`.

```
my_nest_list = [ 1, 2, [ 3, 4, 'hello' ] ]
```

3. Sort the list `my_sort_list` using a built-in method.

```
my_sort_list = [5, 3, 4, 6, 1]
```

Follow Us!

@bitprojectorg

@bitprj

@BitProject

@BitProject

join.bitproject.org