Python If, Else, Elif Statements: Takeaways

Syntax

- Append values with each iteration of a for loop:

```
apps_names = []
for row in apps_data[1:]:
    name = row[1]
    apps_names.append(name)
print(apps_names[:5])
```

- Use an if statement to control your code:

```
if True:
    print(1)
if 1 == 1:
    print(2)
    print(3)
```

- Return boolean values:

```
price = 0
print(price == 0)
print(price == 2)
```

- Execute code only when True follows if:

```
if True:
    print('First Output')
if False:
    print('Second Output')
if True:
    print('Third Output')
```

- Using the == and != operators with strings or lists:

```
print('Games' == 'Music')
print('Games' != 'Music')
print([1,2,3] == [1,2,3])
print([1,2,3] == [1,2,3,4])
```

Concepts

- We can use an `if` **statement** to implement a condition in our code.
- The `if` statement starts with `if`, it continues with a condition such as `price == 0.0`, and it ends with `:`.
- We use the `==` operator to check whether something is **equal** to something else. Don't confuse `==` with `=` (`=` is a variable assignment operator in Python; we use it to assign values to variables — it doesn't tell us anything about equality).

- We indent operations within the body of an `if` statement, such as `list.append(value)` or `print(value)`, four spaces to the right relative to the `if` statement.
- We call `True` and `False` **Boolean values** or **Booleans** — their data type is `bool` ("bool" is an abbreviation for "Boolean").
- Boolean values (`True` and `False`) are necessary parts of any `if` statement. One of the following must always follow `if` : (1) a Boolean value or (2) an expression that evaluates to a Boolean value.
- Indented code *only* executes when `True` follows `if` .
- We can use the `==` and `!=` operators with strings or lists.

Resources

- If Statements in Python