JavaScript

```javascript
// Variables
let fullName = "Jane Hacker";
const pi = 3.14;

// Arrays ("lists" in Py)
let names = ["John", "Paul", "G"];

// Objects (similar to "dicts")
let translation = {
    ola: "Hello",
    oi: "Hi",
};

// For loop
for (let name of names) {
    console.log("name:", name);
}
// While loops
let x = 0;
while (x < 3) {
    console.log("X:", x);
    x++;
}
// If-statements
if (fullName === "Jane") {
    console.log("Hi, Jane!");
} else if (fullName === "Alice") {
    console.log("Hey Alice");
} else {
    console.log("Don't know you");
}
// Array processing (map & filter)
let longNames = names
    .filter(n => n.length > 3)
    .map(n => n.toUpperCase());

// Functions
function greeter(name) {
    console.log("Hi", name);
}
greeter("Bob")

// Arrow function expression
const dst = (x, y) => x*x + y*y;

// Conjunctions
if (age < 18 && drink === "beer") {
    console.log("Too young kiddo");
}
if (age > 18 || drink === "soda") {
    console.log("Great choice");
}
// Class syntax
class User extends BaseUser {
    constructor(name) {
        this.name = name;
        this.loggedIn = false;
    }
    logIn() {
        this.loggedIn = true;
    }
}
let user = new User("jqhacker");
```

Python

```python
# Variables
full_name = "Jane Hacker"
pi = 3.14

# lists ("Arrays" in JS)
names = ["John", "Paul", "G"]

# dicts (similar to "Objects")
translation = {
    "ola": "Hello",
    "oi": "hi",
}

# For loops
for name in names:
    print("name:", name)

# While loops
x = 0
while x < 3:
    print("X:", x)
    x += 1

# If-statements
if full_name == "Jane":
    print("Hi, Jane!")
elif full_name == "Alice":
    print("Hey Alice")
else:
    print("Don't know you")

# List comprehension
long_names = [
    name.upper() for name in names
    if len(name) > 3
]

# Functions
def greeter(name):
    print("Hi", name)
greeter("Bob")

# Lambda function
dst = lambda x, y: x*x + y*y


# Conjunctions
if age < 18 and drink == "beer":
    print("Too young kiddo")
if age > 18 or drink == "soda":
    print("Great choice")

# Class syntax
class User(BaseUser):
    def __init__(self, name):
        self.name = name
        self.logged_in = False

    def log_in(self):
        self.logged_in = True


user = User("jqhacker")
```

More Python types

sets "keys-only dict", with operations

```python
a = {"a", 1, 4, "b"}
b = {"a", "b"}
print(a - b)  #  {1, 4}
```

tuples immutable list (array)

```python
tup = ("a", "b", "c")
# can't do: tup["a"] = 3
```

Python features

template strings Python 3.6+ only

```python
phrase = f"name is {full_name}"
```

Ternary operator

```python
b = 3 if len(a) > 3 else 0
```

in operator can be used with any collection (dict, string, list, etc)

```python
if "Paul" in names:
    print("Found Paul")
```

Try / except Code should throw exceptions when necessary, and use try/catch to handle errors

```python
try:
    third_name = names[3]
except Exception as e:
    pass # Handle ANY exception
except IndexError as e:
    pass # Only one error type
```

General Differences

blocking Callbacks aren't used in Python. The blocking approach is to pause execution and use return values. Although not commonly in use, Python 3.5+ supports async via *await* and *async* syntax.

object oriented programming Python frameworks tend to use classic OOP more often than JS.

operator overloading Python allows custom classes to use operators. Example: define a method called `__add__` to allow the `+` operator.

Zen of Python

Simple is better than complex

There should only be one way to do it

Premature optimization is root of all evil