	PYTHON	JAVASCRIPT
1. Compiled vs Interpreted Implementation	Usually **Interpreted**	Usually **Interpreted**
	Interpreted/Compiled is not a property of a language but a property of implementation. In most languages, the implementation falls under one category, however there are exceptions	
	Interpreted Program	
	- code is executed line by line by an interpreter	
	Compiled Program	
	- all code is converted/translated into a lower level machine code before it is run	
2. Statically or Dynamically Typed Language	**Dynamically** Typed	**Dynamically** Typed
	- perform **type checking** at **run-time**	
	- no need to declare variables before you use them	
	Statically Typed	
	- perform type checking at **compile-time**	
	- must declare variables before you use them	
	Type Checking	
	- verifiying if the data types are compatible with the operands being used on them	
	Ex. String + Number ("2 + 3)	
3. Strongly or Weakly Typed Language	**Strongly**-Typed	**Weakly**-Typed
	does NOT allow implicit conversions between unrelated data types	DOES allow implicit conversions between unrelated data types (ex. numbers -> strings)
	Ex.	Ex.
	score = 21	let score = 21;
	score + "3" # TypeError!	score + "3"; //=> "213"
4. Objects	**Everything** is an object	**Almost** everything is an object
		Not Objects: 1) **String**, 2) **Number**, 3) **Boolean**, 4) **Null**, 5) **Undefined**, 6) **Symbol**, 7) **Big Int**
		JS objects are more like Python classes (even though they syntactically look like python dictionaries)
5. Data Types	5 Main Categories:	2 Main Categories:
	1) **Numeric**: **Integer** (ex. 13, -1), **Float** (1.0), **Complex**	1) **Primitives**
	2) **Dictionary (ex. { })**	2) **Objects**
	3) **Boolean** (ex. True, False)	
	4) **Set**	
	5) **Sequence**: **String** ("yes", 'yes'), **List (ex. [1, 2, "a"])**, **Tuple**	
6. Primitive vs Non-Primitive Data Types	No such thing as "primitives" (in the conventional Java / JavaScript sense)	Primitives are the basic building blocks for other data types, and contain a single "value"
		Immutable data types are values that cannot be changed once they are created
		Primitives: 1) **String**, 2) **Number**, 3) **Boolean**, 4) **Null**, 5) **Undefined**, 6) **Symbol**, 7) **Big Int**
		Non Primitives: **Objects**
7. Immutable Data Types	Data type values cannot be changed once they are created	Data type values cannot be changed once they are created
	Immutable Objects:	**Immutable**: 1) **String**, 2) **Number**, 3) **Boolean**, 4) **Null**, 5) **Undefined**, 6) **Symbol**, 7) **Big Int**
	1) **Integer** (ex. 5, -5)	
	2) **Float** (ex. -5.0)	
	3) **Complex**	

	PYTHON	JAVASCRIPT
	4) **Tuple**	
	5) **String** (ex. "ye", 'ye')	
	6) **Bytes**	
	7) **Frozen Set**	
	Attempting to change the value of an immutable data type results in error!	Attempting to change the value of an immutable data type does NOT result in error!
	Ex.	Ex.
	name = "max"	let name = "max";
	name[0] = "T" # TypeError! 'str' object does not support item assignment	name[0] = "T"; // no error!
		console.log(name); //=> "max"
8. Variable Declaration/Assignment	No need to declare variable types like in C++ (Ex. int myNum;)	No need to declare variable types like in C++ (Ex. int myNum;)
	No declaration of variable before assignment!	No need to declare variable before assigment, but you can
	Ex.	Ex.
	my_num = 5	let number;
		number = 5;
	No keywords when declaring like in JS (let, const, var)	You can use a keyword before variable (let, const var), to control scope of variable
		let = block scope, reassignable
		const = block scope
		var = function scope, reassignable, redeclarable, hoisted
		no keyword = function scope, reassignable
9. Variable Naming	Same as JS	upper/lowercase letters, numbers, and _
		name cant begin with number
10. Multi Variable Assignment	Ex.	Ex.
	a, b, c = 1, 2, 3	[a, b, c] = [1, 2, 3];
11. Constant Variables	convention is to use all uppercase	use keyword "const"
	constant variables CAN be reassigned	constant variables can NOT be reassigned or redeclared
	Ex.	Ex.
	MY_NUM = 5	const myNum = 5;
	MY_NUM = 10 # ok!	myNum = 10; // TypeError!!!
12. None Data Type	None data type is equivalent to JS "null" data type	
	Ex.	Ex.
	count = None	count = null;
13. Function Hoisting	functions are NOT hoisted	function declarations ARE hoisted
		function expressions are NOT hoisted
14. List Data Type	List data type is equivalent to JS "array"	
	Ex.	Ex.
	numbers = [1, 2, 3]	let numbers = [1, 2, 3];

	PYTHON	JAVASCRIPT
	negative indexing to get items starting from end of list	NO negative indexing supported!
	Ex.	
	numbers = [1, "A", 3]	let numbers = [1, "A", 3];
	print(numbers[-1]) #=> 3	console.log(numbers[-1]); //=> undefined