	PYTHON	JAVASCRIPT
1. for __ in sequence :	__ = iterator variable can be named anything (only accessible in this block scope)	__ = iterator variable can be named anything (only accessible in this block scope)
	__ iterator variable refers to **ELEMENT** in sequence (list, string)	__ iterator variable refers to stringified **INDEX** in iteratable (array, string)
	Ex.	Ex.
	letters = ["a", "b", "c"]	let letters = ["a", "b", "c"];
	for char in letters:	for (let char in letters) {
	print(char) # "a", "b", "c"	console.log(char); // "0", "1", "2"
		}
2. for __ in dictionary:	__ iterator variable refers to KEY in dictionary	__ iterator variable refers to KEY in object
	- order of key/values in loop not guaranteed!	- order of key/values in loop not guaranteed!
	Ex.	Ex.
	pantry = { "apple": 2, "mango":1, "pear: 3 }	let pantry = { "apple": 2, "mango":1, "pear: 3 };
	for fruit in pantry:	for (let key in pantry) {
	print(fruit) # "mango", "pear", "apple"	console.log(key); // "apple", "mango", "pear"
		}
3. range(*start*, stop, *step*)	params:	
	- start (int) = optional number where to start list, default 0	
	- stop (int) = required number where to end list (exclusive)	
	- step (int) = optional number how to increment list, default 1	
	returns (list) python v2	
	returns (range) python v3 - less memory	
	Ex1. default step 1	
	x = range(3, 6)	
	print(x) # [3, 4, 5]	
	Ex2. custom step 2	
	x = range(3, 6, 2)	
	print(x) # [3, 5]	
	Ex3. only 1 param- end	
	x = range(4)	
	print(x) # [0, 1, 2, 3]	
	Ex4. negative param end	
	x = range(-3)	
	print(x) # [ ]	
	Ex5. descending numbers	
	x = range(3, -1, -1)	
	print(x) # [3, 2, 1, 0]	

	PYTHON	JAVASCRIPT
4. for __ in range(start, stop):	can use this loop to create C++ or JS style for loops. Ex. for (int i = 0; i < 9; i++) {}	Ex1. loop 4 times from 0 to 3 (inclusive) in steps of 1
		for (let i = 0; i < 4; i++) {
	Ex1. loop 4 times from 0 to 3 (inclusive) in steps of 1	console.log(i); // 0, 1, 2, 3
	for i in range(0, 4):	}
	print(i) # 0, 1, 2, 3	
5. WHILE Loop	Ex1. loop 4 times from 0 to 3 (inclusive) in steps of 1	Ex1. loop 4 times from 0 to 3 (inclusive) in steps of 1
	i = 0	let i = 0;
	while i < 4:	while (i < 4) {
	print(i) # 0, 1, 2, 3	console.log(i); // 0, 1, 2, 3
	i += 1	i++;
		}
6. open( )	open(file, *mode*) #=> file obj	
	- file (string) = path relative or absolute	
	- mode (string) = optional, default "r" read	
	- other modes: "w" write, "a" append, "x" create	
	- file obj = doesn't have actual file contents, but acts like a placeholder to file contents	
	Ex.	
	f = open("stock_prices.csv", "r")	
7. file.read( )	file.read(*n*) #=> str	
	- file (obj) = file object created from open( )	
	- n (int) = optional, refers to number of characters or bytes to read from file, if blank, default entire file	
	- str (string) = entire file as a string, usually separated by "\n" for csv files	
8. string.split( )	string.split(*delimeter*) #=> list	same as python, except see example 4 below
	- delimeter (string) = optional what you want to separate each string with. ex "," or "\n". default is whitespace	
	- list = list of values separated by the delimeter	
	Ex1. delimeter included	Ex1. delimeter included
	str = "a,b,c"	let str = "a,b,c";
	str.split(",") #=> ["a", "b", "c"]	str.split(",") //=> ["a", "b", "c"]
	Ex2. no delimeter	Ex2. no delimeter
	str = "a,b,c"	str = "a,b,c"
	str.split() #=> ["a,b,c"]	str.split() //=> ["a,b,c"]
	Ex3. delimeter not found	Ex3. delimeter not found
	str = "a,b,c"	str = "a,b,c"
	str.split("#") #=> ["a,b,c"]	str.split("#") //=> ["a,b,c"]
	Ex4. empty string delimeter- **ValueError**!!!	Ex4. empty string delimeter- **NO ERROR!**
	str = "a,b,c"	str = "a,b,c"
	str.split("") #=> ValueError!!!!!!!	str.split("") //=> ['a', ',', 'b', ',', 'c']