Standard Library Documentation:

Note 1: Methods are denoted in bold, to differentiate between what it does and the method name easily.

---sy--

let exit(code: int) -> never;

/*The exit function that is part of the system will allow the program to prematurely top functioning within the vm whenever the function is called. The only parameter the exit function will take in will be an int. The code of this exit function will signify whether the program terminated gracefully or if some error occurred. Non-zero codes indicate errors. The return type of this function is never, which is a function that never resolves to return any value at all, similar to how rust's exit function behaves.*/

--io--

let prints(message: string) -> unit;

/*The prints() function will take in a string as its only parameter and output the string onto the console, without including a "\n" escape character at the end of the string to be printed. This function's return type is unit. If this function wanted to work with integers as well, then the integer value has to be converted into a string instead.*/

let printsln(message: string) -> unit;

/*The io function of printsln() will take in a string as its parameters and outputs the string to the console window. Similar to the prins() function, except at the end of the outputted string, this function adds a "\n" at the end of the inputted argument. This function's return type is unit. If this function wanted to work with integers as well, then the integer value has to be converted into a string instead. Its return type is unit.*/

let printi(value: int) -> unit;

/* The io function of printi() will take in a value of type int as its parameters and output the integer value to the console window. Unlike the prints() function, this function will only work with integer values instead. Parsing an integer value to a string will not work with this function. Its return type is unit.*/

let printiln(value: int) -> unit;

/*The io function of printiln() will take in a value of type int as its parameters and output the integer value to the console window. Unlike the printsln() function, this function will only work with integer values instead. Parsing an integer value to a string will not work with this function. This function will add a "\n" escape character to the end of the outputted integer value. Its return type is unit.*/

let asks(prompt: string) -> string;

/* The io function of akss() will print out a string prompt to the console window that will inform the user of a prompt, and the user would have to type into the console window a string value that needs to be

entered. The return type of this function is a string, which needs to be stored in a corresponding variable or be used elsewhere. This is one of the console input methods that can be utilized, similar to how user input prompts in python work. */

let aski(prompt: string) -> int

/*The io funcion of aski() will print out a string prompt to the console window that will inform the user of a prompt, and the user would have to type into the console window a value that can be parsed into an int since all inputs and outputs from the console are treated as strings. Behind the scenes of this function, it converts the value entered into an integer. The return type of this function is an int, which would need to be stored into a corresponding variable to that type. This is another console input methods that can be used, similar to how input prompts in python work. If the user inputted value cannot be parsed to an integer, the function will call upon the exit function, passing in a non-zero code signaling the program to terminate gracefully. Additionally, inputs that are empty strings cannot be parsed and therefore the function will call upon the exit function, tell what the error is, and exit gracefully.*/

--string--

let pure strContains(self: string, needle:string) -> bool implies |needle| <= |self|

/*The string function will take in two strings, one will be the string being investigated, and the other will be the string that needs to be within the original string. The method will work if the length of the substring to be checked upon is less than or equal to the length of the original string. If the length of the substring is more than expected, then the function would not proceed with checking the substring, otherwise the method will implement its own algorithm of checking if a given string contains the substring to be checked for, if the substring exists then it will return true, otherwise it will return false.*/

let pure strStartsWith(self: string, prefix: string) -> bool implies strContains(self, needle)

/*The function will prematurely call upon the strContains function checking if the substring that was passed in exists in the original string. If the strContains method returns false, then the method returns false, as it is not feasible to find the substring needed to be checked over. If the strContains method is true, then the original string passed in gets to be separated by spaces since it is a delimeter of sorts. Once that has been done, the prefix string will be matched with each string from the splitted array for a valid prefix string. If a prefix has been found, then the method returns true, otherwise it will return false.*/

let pure strEndsWith(self: string, suffix: string) -> bool implies strContains(self, suffix);

/*The function will prematurely call upon the strContains function checking if the substring that was passed in exists in the original string. If the strContains function method returns false, then the method return false, as it is not feasible to find the substring needed to be checked over. If the strContains method returns true, then the original string passed in will be splitted by using an empty space as the

delimiter. Once the string has been splitted, each splitted string will be iterated through, and sliced on to check if a suffix exists. If the suffix is contained within one of these slices, then the method returns true, otherwise the original string has been exhausted and the method will end up returning false. */

let pure strAppend(self" string, suffix: string) -> string when |it| = |self| + |suffix| and strContains(it, self) and strContains(it, suffix);

/* The function will take in the self string and the prefix string, append the prefix to the self, and then call upon the strContains function on the new string to be returned and the original strings passed in, to make sure that the new string gets to be returned, and does not cause any unintended side effects to the suffix passed in. */

let pure strInsert(self: string, index: int when it >= 0 and it <= |self|, infix: string) -> string when |it| = |self| + |infix| and strContains(it, infix); /* method likely won't be implemented */

/* The function will take in three parameters, an original input string, an index, and an infix string to be appended to the original input string. The function will insert the infix string into the self passed in at that index, if the index is iwthin the bounds of the array. Once the new string has been created, it will implicitly call upon the strContains() function to make sure the returned string contains the infix string, and will return it once the method finishes its execution.*/

let pure strIndexOf(self: string, needle: string) -> (int when it >= 0 and it < |self|)? iff strContains(self, needle);

/* The function will take in an original string and a substring needed to be found, it will only begin with searching for the index iff the strContains() function returns true, since it would be possible to return a valid index. Once the substring has been discovered, it will return the index where the substring begins from the self string passed in. If the strContains() function were to return false, then the method would return a negative index, signifying failure. */

let pure strLastIndexOf(self: string, needle: string) -> (int when it >= 0 and it < |self|)? iff strContains(self, needle); // method likely won't be implemented

/*From the orginal string, the method will execute completely if the substring passed in exists in the original string somewhere. The method will return the last index where the first occurence of the substring is discovered at, unlike the indexOf method, this returns the index of the last substring's character. If no substring exits, then the method will just return a negative index value, indicating that the method has failed to locate the index. */

let pure strRepeat(self: string, times: int when it >= 0) -> string when |it| = |self| * times and strContains(it, self); // method likely won't be implemented

/* Method returns a new string that repeats and appended by the original string. If the number of times passed in were to be 0, then the method returns an empty string, so that the imposed condition were to be true. Method will assume that strContains will be true only when times is greater than 0 and that the original string contains this substring. */

let pure strReplaceFirst(self: string, needle: string, replacement: string) -> (string when |it| = |self| + |replacement| - |needle| and strContains(it, replacement))? iff strContains(self, needle);

/*The method will take in the orginal string, find the substring needed to be replaced, and the string that it is replaced by. The method will have to check if the substring to be replaced exists in the original string, and if so then it finds the first occurence of that substring and replaces it with a new string. Since it replaces the first occurence, the replacement needs to only occur once. This method will return a new string after the replacement process is over with. */

let pure strReplaceLast(self: string, needle: string, replacement: string) -> (string when |it| = |self| + |replacement| - |needle| and strContains(it, replacement))? iff strContains(self, needle); // likely won't be implemented

/* With the original string, substring, and replacement string, the method will replace the last occurence of the substring with the replacement and return the string after it is over with. If the substring doesn't exist in the original string, then the method fails to execute fully.*/

let pure strRemoveFirst(self: string, needle: string) -> (string when |it| = |self| - |needle|)? iff strContains(self, needle); // likely won't be implemented.

/* Method takes in the original string and the substring to remove, if the substring does not exist within the original string, then the original string gets returned. Otherwise the first occurence of the substring that needs to be removed will be thrown out of the original string, and any remaining characters will have to append to each other if applicable. Returns a new string with the substring removed. Multiple occurences of substring will prioritize the first occurence.*/

let pure strRemoveLast(self: string, needle: string) -> (string when |it| = |self| - |needle|)? iff strContains(self, needle); // likely won't be implemented.

/* Method takes in the original string and the substring to remove, if the substring does not exist within the original string, then the original string gets returned. Otherwise the last ccurence of the substring that needs to be removed will be thrown out of the original string, and any remaining characters will have to append to each other if applicable. Returns a new string with the substring removed. Multiple occurences of substring will prioritize the last occurence.*/

let pure strToLowercase(self: string) -> string when |it| = |self|;

/* Takes in an input string, converts all of its alphabetical character to lowercase, and returns the converted string as a new string. If any characters of the string are either: lowercase, special characters, or are numeric characters, then the method does nothing to them.*/

let pure strToUppercase(self: string) -> string when |it| = |self|;

/* Takes in an input string, converts all of its alphabetical characters to uppercase, and returns the converted string as a new string. If any chracters of the string are either: lowercase, special characters, or are numeric characters, then the method does nothing to them. */

let pure strParseInt(self: string, radix: int when it >= 2 && it <= 36) -> int?;

/*Takes in an input string, and a radix value as an integer. The radix will take in the string, attempt to convert the input string into an integer, and return an integer value represented from base to all the way to base 36. If the input string cannot be converted to an integer properly, then the method panics, and terminates gracefully with an exit code of -1. Alternatively, if the input string contains a set of characters and numbers that exist in the radix's numerical system, then the method will return the decimal representation of that number. /*

let pure strShowInt(value: int, radix: int when it >= 2 && it <= 36) -> string;

/*Takes in an integer value, and returns a string representing the sequence of characters that value makes up, depending on the radix system being used. Integers passed in would have to be in base 10 and the radix will be what base the number should be represented in, from base 2 to base 36.*/

let pure strParseDigit(self: string when |it| = 1, radix: int when it >= 2 && it <= 36) -> (int when it >= 0 && it < radix)?; // likely won't be implemented

/*Method will take in an input string, which will only have 1 character, and the radix in which the number should be represented by. Returns the numerical representation of the character in the specified radix system, as that acts the base the number is. The returned number from the method will be from 0 to radix-1. */

let pure strShowDigit(digit: int when it >= 0 && it < radix, radix: int when it >= 2 && it <= 36) -> string when |it| = 1; // likely won't be implemented

/* Method will take in a digit as an integer, and the radix what base system should that number be represented in. The integer passed in would have to be a positive number in base 10. Returns the string with only one character that is the base of the radix representation of that value. The digit would have to be a number from 0 to radix-1. If the values passed in are invalid, then the method panics and terminates gracefully. /*

let pure strIsEmpty(self: string) -> bool iff |self| = 0;

/* Returns true if the length of the string is 0, otherwise returns false.*/

let pure strReverse(self: string) -> string when |it| = |self|;

/* Returns a new string that is the reverse of the original string passed in. The lenght of the original string and the reversed string are equal to each other. */

let pure strSubstring(self: string, startInclusive: int when it >= 0 and it <= |self|, endExclusive: int when it >= startInclusive and it <= |self|) -> string when |it| = endExclusive - startInclusive and strContains(self, it);

/* Returns a new string starting from the beginning index which is inclusive and the ending index is exclusive. The starting index has to be within the bounds of the string and the ending index has to satisfy being between the starting index and the length of the string. The lenght of the string to be returned is calculated from (endExclusive - startInclusive). */

let pure strSubstring(self: string, startIndex: int when it >= 0 and it < |self|, length: int when it >= 0 and startIndex + it <= |self|) -> string when |it| = length and strContains(self, it);

/* Returns a new string starting from the startIndex specified and the substring that gets returned goes from the startingIndex all the way to whatever the lenght is. Both the startIndex and the length must be within the string's bounds otherwise an index error would likely occur. The length of the new string is the length that has been passed in, if the length passed in was 0, then the character at the starting index would only be returned. */