**Name: Session:**

**Programming I**

**Strings in Python**

Text in our programs is represented by the string data type. You can think of a string as a sequence of characters. Python allows you to designate strings with either a single of double quote. Most languages require double quotes but in Python you can use either. The only restriction is that they must be a matched set. For example, “hello’ would be illegal.

Strings may be stored in variables just as you would numbers.

|  |  |
| --- | --- |
| **Operator** | **Meaning** |
| + | Concatenation |
| \* | Repetition |
| <string>[ ] | Indexing |
| <string>[:] | Slicing |
| len(<string>) | Length |
| for <var> in <string> | Iteration through characters |

|  |  |
| --- | --- |
| **Method** | **Meaning** |
| **capitalize**( ) | Return a copy of the string with only its first character capitalized |
| **center**(*width*[*, fillchar*]) | Return centered in a string of length *width* |
| |  |  | | --- | --- | |  |  |   **count**(*sub*[*, start*[*, end*]]) | Return the number of occurrences of substring *sub* in string S[*start*:*end*] |
| **endswith**(*suffix*[*, start*[*, end*]]) | Return True if the string ends with the specified *suffix*, otherwise return False |
| **expandtabs**([*tabsize*]) | Return a copy of the string where all tab characters are expanded using spaces |
| **find**(*sub*[*, start*[*, end*]]) | Return the lowest index in the string where substring *sub* is found |
| **index**(*sub*[*, start*[*, end*]]) | Return the lowest index in the string where substring *sub* is found |
| **isalnum**( ) | Return true if all characters in the string are alphanumeric and there is at least one character, false otherwise |
| **isalpha**( ) | Return true if all characters in the string are alphabetic and there is at least one character, false otherwise |
| **isdigit**( ) | Return true if all characters in the string are digits and there is at least one character, false otherwise |
| **islower**( ) | Return true if all cased characters in the string are lowercase and there is at least one cased character, false otherwise |
| **isspace**( ) | Return true if there are only whitespace characters in the string and there is at least one character, false otherwise |
| **istitle**( ) | Return true if the string is a titlecased string and there is at least one character |
| **isupper**( ) | Return true if all cased characters in the string are uppercase and there is at least one cased character, false otherwise |
| **join**(*seq*) | Return a string which is the concatenation of the strings in the sequence *seq* |
| **ljust**(*width*[*, fillchar*]) | Return the string left justified in a string of length *width* |
| **lower**( ) | Return a copy of the string converted to lowercase |
| **lstrip**([*chars*]) | Return a copy of the string with leading characters removed |
| **partition**(*sep*) | Split the string at the first occurrence of *sep*, and return a 3-tuple containing the part before the separator, the separator itself, and the part after the separator |
| **replace**(*old, new*[*, count*]) | Return a copy of the string with all occurrences of substring *old* replaced by *new* |
| **rfind**(*sub*[*,start*[*,end*]]) | Return the highest index in the string where substring *sub* is found, such that *sub* is contained within s[start,end] |
| **rindex**(*sub*[*, start*[*, end*]]) | Return the highest index in the string where substring *sub* is found, such that *sub* is contained within s[start,end] |
| **rjust**(*width*[*, fillchar*]) | Return the string right justified in a string of length *width* |
| **rpartition**(*sep*) | Split the string at the last occurrence of *sep*, and return a 3-tuple containing the part before the separator, the separator itself, and the part after the separator |
| **rsplit**([*sep*[*,maxsplit*]]) | Return a list of the words in the string, using *sep* as the delimiter string |
| **rstrip**([*chars*]) | Return a copy of the string with trailing characters removed |
| **split**([*sep*[*,maxsplit*]]) | Return a list of the words in the string, using *sep* as the delimiter string |
| **splitlines**([*keepends*]) | Return a list of the lines in the string, breaking at line boundaries |
| **startswith**(*prefix*[*, start*[*, end*]]) | Return True if string starts with the *prefix*, otherwise return False |
| **strip**([*chars*]) | Return a copy of the string with the leading and trailing characters removed. The *chars* argument is a string specifying the set of characters to be removed |
| **swapcase**( ) | Return a copy of the string with uppercase characters converted to lowercase and vice versa. |
| **title**( ) | Return a titlecased version of the string: words start with uppercase characters, all remaining cased characters are lowercase |
| **upper**( ) | Return a copy of the string converted to uppercase |
| **zfill**(*width*) | Return the numeric string left filled with zeros in a string of length *width* |

**String Practice**

For each of the following expressions, indicate the value returned, or if the evaluation would lead to an error, write the word 'error'.

1. "a" + "bc"



correct

1. 3 \* "bc"



correct

1. "3" \* "bc"



correct

1. "abcd"[2]



correct

1. "abcd"[0:2]



correct

1. "abcd"[:2]



correct

1. "abcd"[2:]



For each of the expressions below, specify its type and value. If it generates an error, select type 'NoneType' and write the word 'error'. The Type will be either NoneType, int, float, Boolean, or string.

Assume we've made the following assignments:

> str1 = 'hello'

> str2 = ','

> str3 = 'world'

1. str1

Top of Form

Type

Bottom of Form



correct

1. str1[0]

Top of Form

 Type

Bottom of Form



correct

1. str1[1]

Top of Form

 Type

Bottom of Form



correct

1. str1[-1]

Top of Form

 Type

Bottom of Form



correct

1. len(str1)

Top of Form

 Type

Bottom of Form



correct

1. str1[len(str1)]

Top of Form

 Type

Bottom of Form



correct

1. str1 + str2 + str3

Top of Form

 Type

Bottom of Form



correct

1. str1 + str2 + ' ' + str3

Top of Form

 Type

Bottom of Form



correct

1. str3 \* 3

Top of Form

 Type

Bottom of Form



correct

1. 'hello' == str1

Top of Form

 Type

Bottom of Form



correct

1. 'HELLO' == str1

Top of Form

 Type

Bottom of Form



correct

1. 'a' in str3

Top of Form

 Type

Bottom of Form



correct

str4 = str1 + str3

'low' in str4

Top of Form

 Type

Bottom of Form



correct

1. str3[1:3]

Top of Form

 Type

Bottom of Form



correct

1. str3[:3]

Top of Form

 Type

Bottom of Form



correct

1. str3[:-1]

Top of Form

 Type

Bottom of Form



correct

1. str1[1:]

Top of Form

 Type

Bottom of Form



correct

1. str4[1:9]

Top of Form

 Type

Bottom of Form



correct

1. str4[1:9:2]

Top of Form

 Type

Bottom of Form



correct

1. str4[::-1]

Top of Form

 Type

Bottom of Form

