Ch06-Strings

August 7, 2020

1 6 Strings

- http://openbookproject.net/thinkcs/python/english3e/strings.html

1.1 Topics

- learn in-depth about string data types
- methods provided in string objects/data
- operators and various operations
- ways to traverse/step through characters in string

1.2 6.1 Introduction

- strings are text data; not numeric!
- breifely covered string while covering data and types in an earlier chapter
- unlike numbers, strings are compound data type; sequence of characters
- can work with string as a single thing
- string variables are objects with their own attributes and methods
- help(str) to see all the methods
- commonly used methods: upper(), lower(), swapcase(), capitalize(), endswith(), isdigit(), find(), center(), count(), split(), etc.

1.3 6.2 String methods and operations

- string objects/data come with dozens of methods you can invoke

[1]:
```
# let's see built-in documentation on str
help(str)
```

```
Help on class str in module builtins:

class str(object)
 |  str(object='') -> str
 |  str(bytes_or_buffer[, encoding[, errors]]) -> str
 |
 |  Create a new string object from the given object. If encoding or
 |  errors is specified, then the object must expose a data buffer
 |  that will be decoded using the given encoding and error handler.
 |  Otherwise, returns the result of object.__str__() (if defined)
```

```
|  or repr(object).
|  encoding defaults to sys.getdefaultencoding().
|  errors defaults to 'strict'.
|
|  Methods defined here:
|
|  __add__(self, value, /)
|      Return self+value.
|
|  __contains__(self, key, /)
|      Return key in self.
|
|  __eq__(self, value, /)
|      Return self==value.
|
|  __format__(self, format_spec, /)
|      Return a formatted version of the string as described by format_spec.
|
|  __ge__(self, value, /)
|      Return self>=value.
|
|  __getattribute__(self, name, /)
|      Return getattr(self, name).
|
|  __getitem__(self, key, /)
|      Return self[key].
|
|  __getnewargs__(…)
|
|  __gt__(self, value, /)
|      Return self>value.
|
|  __hash__(self, /)
|      Return hash(self).
|
|  __iter__(self, /)
|      Implement iter(self).
|
|  __le__(self, value, /)
|      Return self<=value.
|
|  __len__(self, /)
|      Return len(self).
|
|  __lt__(self, value, /)
|      Return self<value.
|
|  __mod__(self, value, /)
```

```
 |      Return self%value.
 |
 |  __mul__(self, value, /)
 |      Return self*value.
 |
 |  __ne__(self, value, /)
 |      Return self!=value.
 |
 |  __repr__(self, /)
 |      Return repr(self).
 |
 |  __rmod__(self, value, /)
 |      Return value%self.
 |
 |  __rmul__(self, value, /)
 |      Return value*self.
 |
 |  __sizeof__(self, /)
 |      Return the size of the string in memory, in bytes.
 |
 |  __str__(self, /)
 |      Return str(self).
 |
 |  capitalize(self, /)
 |      Return a capitalized version of the string.
 |
 |      More specifically, make the first character have upper case and the rest
lower
 |      case.
 |
 |  casefold(self, /)
 |      Return a version of the string suitable for caseless comparisons.
 |
 |  center(self, width, fillchar=' ', /)
 |      Return a centered string of length width.
 |
 |      Padding is done using the specified fill character (default is a space).
 |
 |  count(…)
 |      S.count(sub[, start[, end]]) -> int
 |
 |      Return the number of non-overlapping occurrences of substring sub in
 |      string S[start:end].  Optional arguments start and end are
 |      interpreted as in slice notation.
 |
 |  encode(self, /, encoding='utf-8', errors='strict')
 |      Encode the string using the codec registered for encoding.
 |
```

```
 |      encoding
 |        The encoding in which to encode the string.
 |      errors
 |        The error handling scheme to use for encoding errors.
 |        The default is 'strict' meaning that encoding errors raise a
 |        UnicodeEncodeError.  Other possible values are 'ignore', 'replace' and
 |        'xmlcharrefreplace' as well as any other name registered with
 |        codecs.register_error that can handle UnicodeEncodeErrors.
 |
 |  endswith(…)
 |      S.endswith(suffix[, start[, end]]) -> bool
 |
 |      Return True if S ends with the specified suffix, False otherwise.
 |      With optional start, test S beginning at that position.
 |      With optional end, stop comparing S at that position.
 |      suffix can also be a tuple of strings to try.
 |
 |  expandtabs(self, /, tabsize=8)
 |      Return a copy where all tab characters are expanded using spaces.
 |
 |      If tabsize is not given, a tab size of 8 characters is assumed.
 |
 |  find(…)
 |      S.find(sub[, start[, end]]) -> int
 |
 |      Return the lowest index in S where substring sub is found,
 |      such that sub is contained within S[start:end].  Optional
 |      arguments start and end are interpreted as in slice notation.
 |
 |      Return -1 on failure.
 |
 |  format(…)
 |      S.format(*args, **kwargs) -> str
 |
 |      Return a formatted version of S, using substitutions from args and
kwargs.
 |      The substitutions are identified by braces ('{' and '}').
 |
 |  format_map(…)
 |      S.format_map(mapping) -> str
 |
 |      Return a formatted version of S, using substitutions from mapping.
 |      The substitutions are identified by braces ('{' and '}').
 |
 |  index(…)
 |      S.index(sub[, start[, end]]) -> int
 |
 |      Return the lowest index in S where substring sub is found,
```

```
 |      such that sub is contained within S[start:end].  Optional
 |      arguments start and end are interpreted as in slice notation.
 |
 |      Raises ValueError when the substring is not found.
 |
 |  isalnum(self, /)
 |      Return True if the string is an alpha-numeric string, False otherwise.
 |
 |      A string is alpha-numeric if all characters in the string are alpha-
numeric and
 |      there is at least one character in the string.
 |
 |  isalpha(self, /)
 |      Return True if the string is an alphabetic string, False otherwise.
 |
 |      A string is alphabetic if all characters in the string are alphabetic
and there
 |      is at least one character in the string.
 |
 |  isascii(self, /)
 |      Return True if all characters in the string are ASCII, False otherwise.
 |
 |      ASCII characters have code points in the range U+0000-U+007F.
 |      Empty string is ASCII too.
 |
 |  isdecimal(self, /)
 |      Return True if the string is a decimal string, False otherwise.
 |
 |      A string is a decimal string if all characters in the string are decimal
and
 |      there is at least one character in the string.
 |
 |  isdigit(self, /)
 |      Return True if the string is a digit string, False otherwise.
 |
 |      A string is a digit string if all characters in the string are digits
and there
 |      is at least one character in the string.
 |
 |  isidentifier(self, /)
 |      Return True if the string is a valid Python identifier, False otherwise.
 |
 |      Use keyword.iskeyword() to test for reserved identifiers such as "def"
and
 |      "class".
 |
 |  islower(self, /)
 |      Return True if the string is a lowercase string, False otherwise.
```

```
 |
 |      A string is lowercase if all cased characters in the string are
lowercase and
 |      there is at least one cased character in the string.
 |
 |  isnumeric(self, /)
 |      Return True if the string is a numeric string, False otherwise.
 |
 |      A string is numeric if all characters in the string are numeric and
there is at
 |      least one character in the string.
 |
 |  isprintable(self, /)
 |      Return True if the string is printable, False otherwise.
 |
 |      A string is printable if all of its characters are considered printable
in
 |      repr() or if it is empty.
 |
 |  isspace(self, /)
 |      Return True if the string is a whitespace string, False otherwise.
 |
 |      A string is whitespace if all characters in the string are whitespace
and there
 |      is at least one character in the string.
 |
 |  istitle(self, /)
 |      Return True if the string is a title-cased string, False otherwise.
 |
 |      In a title-cased string, upper- and title-case characters may only
 |      follow uncased characters and lowercase characters only cased ones.
 |
 |  isupper(self, /)
 |      Return True if the string is an uppercase string, False otherwise.
 |
 |      A string is uppercase if all cased characters in the string are
uppercase and
 |      there is at least one cased character in the string.
 |
 |  join(self, iterable, /)
 |      Concatenate any number of strings.
 |
 |      The string whose method is called is inserted in between each given
string.
 |      The result is returned as a new string.
 |
 |      Example: '.'.join(['ab', 'pq', 'rs']) -> 'ab.pq.rs'
 |
```

```
 |  ljust(self, width, fillchar=' ', /)
 |      Return a left-justified string of length width.
 |
 |      Padding is done using the specified fill character (default is a space).
 |
 |  lower(self, /)
 |      Return a copy of the string converted to lowercase.
 |
 |  lstrip(self, chars=None, /)
 |      Return a copy of the string with leading whitespace removed.
 |
 |      If chars is given and not None, remove characters in chars instead.
 |
 |  partition(self, sep, /)
 |      Partition the string into three parts using the given separator.
 |
 |      This will search for the separator in the string.  If the separator is
found,
 |      returns a 3-tuple containing the part before the separator, the
separator
 |      itself, and the part after it.
 |
 |      If the separator is not found, returns a 3-tuple containing the original
string
 |      and two empty strings.
 |
 |  replace(self, old, new, count=-1, /)
 |      Return a copy with all occurrences of substring old replaced by new.
 |
 |        count
 |          Maximum number of occurrences to replace.
 |          -1 (the default value) means replace all occurrences.
 |
 |      If the optional argument count is given, only the first count
occurrences are
 |      replaced.
 |
 |  rfind(…)
 |      S.rfind(sub[, start[, end]]) -> int
 |
 |      Return the highest index in S where substring sub is found,
 |      such that sub is contained within S[start:end].  Optional
 |      arguments start and end are interpreted as in slice notation.
 |
 |      Return -1 on failure.
 |
 |  rindex(…)
 |      S.rindex(sub[, start[, end]]) -> int
```

```
 |
 |      Return the highest index in S where substring sub is found,
 |      such that sub is contained within S[start:end].  Optional
 |      arguments start and end are interpreted as in slice notation.
 |
 |      Raises ValueError when the substring is not found.
 |
 |  rjust(self, width, fillchar=' ', /)
 |      Return a right-justified string of length width.
 |
 |      Padding is done using the specified fill character (default is a space).
 |
 |  rpartition(self, sep, /)
 |      Partition the string into three parts using the given separator.
 |
 |      This will search for the separator in the string, starting at the end.
If
 |      the separator is found, returns a 3-tuple containing the part before the
 |      separator, the separator itself, and the part after it.
 |
 |      If the separator is not found, returns a 3-tuple containing two empty
strings
 |      and the original string.
 |
 |  rsplit(self, /, sep=None, maxsplit=-1)
 |      Return a list of the words in the string, using sep as the delimiter
string.
 |
 |        sep
 |          The delimiter according which to split the string.
 |          None (the default value) means split according to any whitespace,
 |          and discard empty strings from the result.
 |        maxsplit
 |          Maximum number of splits to do.
 |          -1 (the default value) means no limit.
 |
 |      Splits are done starting at the end of the string and working to the
front.
 |
 |  rstrip(self, chars=None, /)
 |      Return a copy of the string with trailing whitespace removed.
 |
 |      If chars is given and not None, remove characters in chars instead.
 |
 |  split(self, /, sep=None, maxsplit=-1)
 |      Return a list of the words in the string, using sep as the delimiter
string.
 |
```

```
 |      sep
 |        The delimiter according which to split the string.
 |        None (the default value) means split according to any whitespace,
 |        and discard empty strings from the result.
 |      maxsplit
 |        Maximum number of splits to do.
 |        -1 (the default value) means no limit.
 |
 |  splitlines(self, /, keepends=False)
 |      Return a list of the lines in the string, breaking at line boundaries.
 |
 |      Line breaks are not included in the resulting list unless keepends is
given and
 |      true.
 |
 |  startswith(…)
 |      S.startswith(prefix[, start[, end]]) -> bool
 |
 |      Return True if S starts with the specified prefix, False otherwise.
 |      With optional start, test S beginning at that position.
 |      With optional end, stop comparing S at that position.
 |      prefix can also be a tuple of strings to try.
 |
 |  strip(self, chars=None, /)
 |      Return a copy of the string with leading and trailing whitespace remove.
 |
 |      If chars is given and not None, remove characters in chars instead.
 |
 |  swapcase(self, /)
 |      Convert uppercase characters to lowercase and lowercase characters to
uppercase.
 |
 |  title(self, /)
 |      Return a version of the string where each word is titlecased.
 |
 |      More specifically, words start with uppercased characters and all
remaining
 |      cased characters have lower case.
 |
 |  translate(self, table, /)
 |      Replace each character in the string using the given translation table.
 |
 |        table
 |          Translation table, which must be a mapping of Unicode ordinals to
 |          Unicode ordinals, strings, or None.
 |
 |      The table must implement lookup/indexing via __getitem__, for instance a
 |      dictionary or list.  If this operation raises LookupError, the character
```

```
is
 |      left untouched.  Characters mapped to None are deleted.
 |
 |  upper(self, /)
 |      Return a copy of the string converted to uppercase.
 |
 |  zfill(self, width, /)
 |      Pad a numeric string with zeros on the left, to fill a field of the
given width.
 |
 |      The string is never truncated.
 |
 |  ----------------------------------------------------------------------
 |  Static methods defined here:
 |
 |  __new__(*args, **kwargs) from builtins.type
 |      Create and return a new object.  See help(type) for accurate signature.
 |
 |  maketrans(x, y=None, z=None, /)
 |      Return a translation table usable for str.translate().
 |
 |      If there is only one argument, it must be a dictionary mapping Unicode
 |      ordinals (integers) or characters to Unicode ordinals, strings or None.
 |      Character keys will be then converted to ordinals.
 |      If there are two arguments, they must be strings of equal length, and
 |      in the resulting dictionary, each character in x will be mapped to the
 |      character at the same position in y. If there is a third argument, it
 |      must be a string, whose characters will be mapped to None in the result.
```

[1]:
```
# let's use some of the methods provided for string objects/data
ss = "Hello there beautiful World!"
tt = ss.upper()
print(tt)
```

```
HELLO THERE BEAUTIFUL WORLD!
```

[2]:
```
print(ss)
```

```
Hello there beautiful World!
```

[3]:
```
print(tt.capitalize())
```

```
Hello there beautiful world!
```

[4]:
```
alist = tt.split()
print(alist)
```

```
['HELLO', 'THERE', 'BEAUTIFUL', 'WORLD!']
```

[5]:
```
# examples of some methods
ss.count('o')
```

[5]: 2

[6]:
```
print(ss.swapcase())
```

```
hELLO THERE BEAUTIFUL wORLD!
```

1.4 6.3 Operators on string

- * and + operators work on string data type as well
- + : concatenates
- * : repeats/multiplies

[7]:
```
a = "ABC"
```

[11]:
```
a = a + "D" + "E " + "F " + "1 " + "2" + a
```

[12]:
```
print(a)
```

```
DE F 1 2ABCDE F 1 2DE F 1 2ABC
```

[13]:
```
"A"*5
```

[13]: 'AAAAA'

[14]:
```
gene = "AGT"*10
print(gene)
```

```
AGTAGTAGTAGTAGTAGTAGTAGTAGT
```

[16]:
```
print("hello world!"*10)
```

```
hello world!hello world!hello world!hello world!hello world!hello world!hello
world!hello world!hello world!hello world!
```

1.5 6.4 working with parts of a string

- string can be sliced using [index] or [inclStartIndex : ExclEndIndex : step] bracket operator
- negative indices are allowed
- len( ) built-in function gives the length of a string

[3]:
```
# examples
s = "Pirates of the Caribbean"
```

[4]:
```
# access the second character
s[1]
```

[4]: 'i'

[5]:
```
s[:]
```

[5]: 'Pirates of the Caribbean'

[6]:
```
s[1:]
```

[6]: 'irates of the Caribbean'

[19]:
```
# print just the pirates
print(s[0:7])
```

```
Pirates
```

[7]:
```
# print "the" from string s
theIndex = s.find("the")
print('the startst at', theIndex)
print(s[theIndex:theIndex+4])
```

```
the startst at 11
the
```

[13]:
```
# TODO
# print Caribbean from string s - hint use find function
```

[ ]:
```
lastSpace = s.rfind(' ')
print(lastSpace)
```

[21]:
```
# print the last character
print(s[-1])
```

```
n
```

[22]:
```
# print string in reverse order
reversedS = s[-1::-1]
```

[23]:
```
ss = "Pirates of the Caribbean."
print(ss[len(ss)-1])
```

```
.
```

[24]:
```
# test whether a given string is palindrome
a= "racecar"
print ('palindrome') if a==a[::-1] else print('not palindrome')
```

```
palindrome
```

1.5.1 6.5 string is immutable

- string objects/variables can't be modified in place!
- you must reassign or make a copy to update strings

[1]:
```
a = 'hello'
```

[2]:
```
a[0] = 'H'
```

```
      ␣
↪---------------------------------------------------------------------------

       TypeError                                 Traceback (most recent call␣
↪last)

       <ipython-input-2-4903e04a4d6c> in <module>
   ----> 1 a[0] = 'H'


       TypeError: 'str' object does not support item assignment
```

1.6 6.6 string traversal

- it's a common practice to go through every character of a given string
- can use both for and while loop to traverse a string

[26]:
```
# example using for loop
# traversing string using index
for i in range(len(s)):
    print(s[i], end=' ')
```

```
P i r a t e s   o f   t h e   C a r i b b e a n
```

[25]:
```
# range-based loop traversing each character
for c in s:
    print(c, end=' ')
```

```
P i r a t e s   o f   t h e   C a r i b b e a n
```

[28]:
```
someStr = """afAdf@#456'"""
```

[29]:
```
# example using while loop
i = 0
while i < len(someStr):
    print(someStr[i], end=' ')
```

```
    i += 1
```

```
a f A d f @ # 4 5 6 '
```

1.7 6.7 string comparison

- strings can be compared using comparison operators
- comparison operators are ==, !=, <=, >=, <, >
- compares lexicographically using ASCII values
  - see ASCII table for values
- ord('c') provides ASCII value of the given character
- two strings are compared character by character in corresponding positions

[2]:
```
# find ascii values of lower a and upper A
print(ord('A'), ord('a'))
```

```
65 97
```

[33]:
```
# string comparison examples
print("apple" == "Apple")
```

```
False
```

[34]:
```
print("apple" >= "ball")
```

```
False
```

[17]:
```
# greater and less than returns True if first two corresponding
#characters have valid order
print("apple" >= "Apple")
```

```
True
```

[18]:
```
# since A is <= a; result is True
# eventhough b is not less than B or c
print('Abc' <= 'aBC')
```

```
True
```

[32]:
```
# for equality all characters have to match
print("apple" == "Apple") # false
```

```
False
```

[20]:
```
# for inequalify any one pair of unmatched characters will do
print('apple' != 'applE')
```

```
True
```

1.8 6.8 in and not in operators

- help quickly test for membership
- can be used to check if a substring is in a string

[36]:
```
print("p" in "apple")
```

```
True
```

[37]:
```
print("pe" in "apple")
```

```
False
```

[38]:
```
print("aple" not in "apple")
```

```
True
```

1.9 6.9 cleaning up strings

- often times working with strings involve removing punctuations and unwanted characters
- traverse the string by removing any encountered punctations

[39]:
```
# create a new string removing punctuations from the following string
ss = '"Well, I never did!", said Alice.'
print(ss)
```

```
"Well, I never did!", said Alice.
```

[8]:
```
# one solution is to use ASCII value of each character between A..Z and a..z
print(ord('a'), ord('z'), ord('A'), ord('Z'))
```

```
97 122 65 90
```

[40]:
```
newStr = ''
for c in ss:
    if ord(c) == ord(' '): # keep space
        newStr += c
    elif ord(c) >= ord('A') and ord(c) <= ord('Z'):
        newStr += c
    elif ord(c) >= ord('a') and ord(c) <= ord('z'):
        newStr += c
print(newStr)
```

```
Well I never did said Alice
```

[41]:
```
# convert newStr to lowercase for case insensitive operations
newStr1 = newStr.lower()
print(newStr1)
```

```
well i never did said alice
```

[42]:
```
# convert sentence into list of tokens/terms/words
words = newStr1.split()
print(words)
```

```
['well', 'i', 'never', 'did', 'said', 'alice']
```

[43]:
```
# traverse through list of words
for w in words:
    print(w)
```

```
well
i
never
did
said
alice
```

[10]:
```
# next solution using string library
# string library provides range of different types of characters as data
import string
help(string)
```

```
Help on module string:

NAME
    string - A collection of string constants.

MODULE REFERENCE
    https://docs.python.org/3.7/library/string

    The following documentation is automatically generated from the Python
    source files.  It may be incomplete, incorrect or include features that
    are considered implementation detail and may vary between Python
    implementations.  When in doubt, consult the module reference at the
    location listed above.

DESCRIPTION
    Public module variables:

    whitespace -- a string containing all ASCII whitespace
    ascii_lowercase -- a string containing all ASCII lowercase letters
    ascii_uppercase -- a string containing all ASCII uppercase letters
    ascii_letters -- a string containing all ASCII letters
    digits -- a string containing all ASCII decimal digits
    hexdigits -- a string containing all ASCII hexadecimal digits
    octdigits -- a string containing all ASCII octal digits
    punctuation -- a string containing all ASCII punctuation characters
    printable -- a string containing all ASCII characters considered printable
```

```
CLASSES
    builtins.object
        Formatter
        Template

    class Formatter(builtins.object)
     |  Methods defined here:
     |
     |  check_unused_args(self, used_args, args, kwargs)
     |
     |  convert_field(self, value, conversion)
     |
     |  format(*args, **kwargs)
     |
     |  format_field(self, value, format_spec)
     |
     |  get_field(self, field_name, args, kwargs)
     |      # given a field_name, find the object it references.
     |      #  field_name:   the field being looked up, e.g. "0.name"
     |      #                 or "lookup[3]"
     |      #  used_args:    a set of which args have been used
     |      #  args, kwargs: as passed in to vformat
     |
     |  get_value(self, key, args, kwargs)
     |
     |  parse(self, format_string)
     |      # returns an iterable that contains tuples of the form:
     |      # (literal_text, field_name, format_spec, conversion)
     |      # literal_text can be zero length
     |      # field_name can be None, in which case there's no
     |      #  object to format and output
     |      # if field_name is not None, it is looked up, formatted
     |      #  with format_spec and conversion and then used
     |
     |  vformat(self, format_string, args, kwargs)
     |
     |  ----------------------------------------------------------------------
     |  Data descriptors defined here:
     |
     |  __dict__
     |      dictionary for instance variables (if defined)
     |
     |  __weakref__
     |      list of weak references to the object (if defined)

    class Template(builtins.object)
     |  Template(template)
```

```
 |
 |  A string class for supporting $-substitutions.
 |
 |  Methods defined here:
 |
 |  __init__(self, template)
 |      Initialize self.  See help(type(self)) for accurate signature.
 |
 |  safe_substitute(*args, **kws)
 |
 |  substitute(*args, **kws)
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors defined here:
 |
 |  __dict__
 |      dictionary for instance variables (if defined)
 |
 |  __weakref__
 |      list of weak references to the object (if defined)
 |
 |  ----------------------------------------------------------------------
 |  Data and other attributes defined here:
 |
 |  braceidpattern = None
 |
 |  delimiter = '$'
 |
 |  flags = <RegexFlag.IGNORECASE: 2>
 |
 |  idpattern = '(?a:[_a-z][_a-z0-9]*)'
 |
 |  pattern = re.compile('\n    \\$(?:\n
(?P<escaped>\\$)…ced>(?a:[…

FUNCTIONS
    capwords(s, sep=None)
        capwords(s [,sep]) -> string

        Split the argument into words using split, capitalize each
        word using capitalize, and join the capitalized words using
        join.  If the optional second argument sep is absent or None,
        runs of whitespace characters are replaced by a single space
        and leading and trailing whitespace are removed, otherwise
        sep is used to split and join the words.

DATA
    __all__ = ['ascii_letters', 'ascii_lowercase', 'ascii_uppercase', 'cap…
```

```
    ascii_letters = 'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ'
    ascii_lowercase = 'abcdefghijklmnopqrstuvwxyz'
    ascii_uppercase = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
    digits = '0123456789'
    hexdigits = '0123456789abcdefABCDEF'
    octdigits = '01234567'
    printable = '0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTU…
    punctuation = '!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~'
    whitespace = ' \t\n\r\x0b\x0c'

FILE
    /Users/rbasnet/anaconda3/lib/python3.7/string.py
```

[11]:
```
# string library has data that can be useful, e.g.
string.punctuation
```

[11]:
```
'!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~'
```

[46]:
```
ss
```

[46]:
```
'"Well, I never did!", said Alice.'
```

[48]:
```
newStr = ''
for c in ss:
    if c in string.ascii_lowercase:
        newStr += c
    elif c in string.ascii_uppercase:
        newStr += c
    elif c == ' ':
        newStr += ' '

print(newStr)
```

```
Well I never did said Alice
```

[49]:
```
# write a function that removes all the punctations except for space
# returns new cleaned up string
def cleanUp(someStr):
    newStr = ''
    for c in someStr:
        if c.islower():
            newStr += c
        elif c.isupper():
            newStr += c
        elif c.isspace():
            newStr += c
```

```
    return newStr
```

[50]:
```
s = cleanUp(ss)
print(s)
```

```
Well I never did said Alice
```

1.10 6.10 string formatting

- format method
- use {} as replacement field
- numbers in curly braces are optional; determine which argument gets substituted
- each of the replace fields can also contain a format specification
  - < left alignment, > right, and ^ center, e.g. {1:<10}
  - 10 is width
  - type conversion such as f for float (.2f two decimal places), x for hex, etc.

[13]:
```
name = "Arthur"
age = 25
```

[14]:
```
s1 = "His name is {}!".format(name)
print(s1)
```

```
His name is Arthur!
```

[16]:
```
# newer syntax
print(f"His name is {name}!")
```

```
His name is Arthur!
```

[46]:
```
# note age and name are provided in reverse order
print("His name is {1} and {1} is {0} years old.".format(age, name))
```

```
His name is Arthur and Arthur is 25 years old.
```

[53]:
```
n1 = 4
n2 = 5.5
s3 = "{0} x {1} = {2} and {0} ^ {1} = {3:.2f}".format(n1, n2, n1*n2, n1**n2)
print(s3)
```

```
4 x 5.5 = 22.0 and 4 ^ 5.5 = 2048.00
```

[23]:
```
# formating decimal/float values to certian decimal points
print("Pi to three decimal places is {0:.3f}".format(3.1415926))
```

```
Pi to three decimal places is 3.142
```

[24]:
```
n1 = "Paris"
n2 = "Whitney"
n3 = "Hilton"
print("123456789 123456789 123456789 123456789 123456789 123456789")
print("|||{0:<15}|||{1:^15}|||{2:>15}|||Born in {3}|||"
        .format(n1,n2,n3,1981))
```

```
123456789 123456789 123456789 123456789 123456789 123456789
|||Paris          |||    Whitney    |||         Hilton|||Born in 1981|||
```

[34]:
```
# formatting decimal int to hexadecimal number
print("The decimal value {0} converts to hex value 0x{0:x}".format(16))
```

```
The decimal value 16 converts to hex value 0x10
```

[38]:
```
# formatting decimal int to binary number
print("The decimal value {0} converts to binary value 0b{0:b}".format(8))
```

```
The decimal value 8 converts to hex value 0b1000
```

[41]:
```
# formatting decimal int to octal number
print("The decimal value {0} converts to octal value 0o{0:o}".format(8))
```

```
The decimal value 8 converts to octal value 0o10
```

[42]:
```
letter = """
Dear {0} {2}.
 {0}, I have an interesting money-making proposition for you!
 If you deposit $10 million into my bank account, I can
 double your money ...
"""
print(letter.format("Paris", "Whitney", "Hilton"))
print(letter.format("Bill", "Warren", "Jeff"))
```

```
Dear Paris Hilton.
 Paris, I have an interesting money-making proposition for you!
 If you deposit $10 million into my bank account, I can
 double your money …


Dear Bill Jeff.
 Bill, I have an interesting money-making proposition for you!
 If you deposit $10 million into my bank account, I can
 double your money …
```

[51]:
```
layout = "{0:>4}{1:>6}{2:>6}{3:>8}{4:>13}{5:>24}"

print(layout.format("i", "i**2", "i**3", "i**5", "i**10", "i**20"))
for i in range(1, 11):
    print(layout.format(i, i**2, i**3, i**5, i**10, i**20))
```

```
   i  i**2  i**3    i**5        i**10                    i**20
   1     1     1       1           1                        1
   2     4     8      32        1024                  1048576
   3     9    27     243       59049                3486784401
   4    16    64    1024     1048576            1099511627776
   5    25   125    3125     9765625           95367431640625
   6    36   216    7776    60466176         3656158440062976
   7    49   343   16807   282475249        79792266297612001
   8    64   512   32768  1073741824     1152921504606846976
   9    81   729   59049  3486784401    12157665459056928801
  10   100  1000  100000 10000000000   100000000000000000000
```

1.11 6.11 Exercises

1. print a neat looking multiplication table like this:
2. Write a program that determines whether a given string is palindrome. Palindrome is a word, phrase, or sequence that reads the same backward as forward, e.g., madam or nurses run or race car.

2.1 Convert Exercise 2 into a function and write at least two test cases.

3. Write a program that calculates number of trials required to guess a 3 digit pass code 777 (starting from 000, 001, 002, 003, 004, 005..., 010, etc.) using some brute force technique.

3.1. Convert Exercise 3 into a function and write at least 3 test cases.

4. Write a function that calculates the run-length encoding of a given string. Run-length is a lossless data compression in which runs of data are stored as a single data value and count, rather than the original run. Assume that the data contains alphabets (upper and lowercase) only and are case insisitive. E.g.:
   - aaaabbc -> 4a2b1c
   - Abcd -> 1a1b1c1d

[18]:
```
# 4 solution
# Algorithm:
# for each character:
#   if the current character is same as the previous one
#        increment count
#   else
#        print the count and the previous character
#        reset count and previous character
#
```

```
def run_length_encoding(text):
    # check for corner case
    if not text: # if text is empty!
        return ''

    encoding = ''
    # FIXME: implement the algorithm


    return encoding
```

[20]:
```
# unit testing for run_length_encoding
assert run_length_encoding('') == ''
assert run_length_encoding('aaaabbc') == '4a2b1c'
assert run_length_encoding('abcd') == '1a2b3c4d'
assert run_length_encoding('zzaazyyyYY') == '2z2a1z5y'
# FIXME: Write few more test cases; what corner cases can you think of
# that would break run_length_encoding function?
```

```
␣
↪---------------------------------------------------------------------------

AssertionError                            Traceback (most recent call␣
↪last)

<ipython-input-20-5d7c792797c3> in <module>
  1 # unit testing for run_length_encoding
  2 assert run_length_encoding('') == ''
----> 3 assert run_length_encoding('aaaabbc') == '4a2b1c'
  4 assert run_length_encoding('abcd') == '1a2b3c4d'
  5 assert run_length_encoding('zzaazyyyYY') == '2z2a1z5y'


AssertionError:
```

5. Write a function that decodes the given run-length encoded compressed data, i.e. decompresses the compressed data to the original string. e.g.
   - '' -> ''
   - '1a2b3c' -> 'abbccc'
   - '10a' -> 'aaaaaaaaaa'

1.12 Kattis problems

1. Avion - https://open.kattis.com/problems/avion

- Apaxiaans - https://open.kattis.com/problems/apaxiaaans

- Hissing Microphone - https://open.kattis.com/problems/hissingmicrophone
- Reversed Binary Numbers - https://open.kattis.com/problems/reversebinary
- Kemija - https://open.kattis.com/problems/kemija08
- Simon Says - https://open.kattis.com/problems/simonsays
- Simon Says - https://open.kattis.com/problems/simon
- Quite a Problem - https://open.kattis.com/problems/quiteaproblem
- Eligibility - https://open.kattis.com/problems/eligibility
- Charting Progress - https://open.kattis.com/problems/chartingprogress
- Pig Latin - https://open.kattis.com/problems/piglatin
- Battle Simulation - https://open.kattis.com/problems/battlesimulation
- Palindromic Password - https://open.kattis.com/problems/palindromicpassword
- Image Decoding - https://open.kattis.com/problems/imagedecoding

[ ]: