Python CS-521

Eugene Pinsky
Department of Computer Science
Metropolitan College, Boston University
Boston, MA 02215
email: epinsky@bu.edu

May 11, 2020

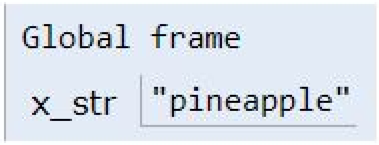
Abstract

This course will present an effective approach to help you learn Python. With extensive use of graphical illustrations, we will build understanding of Python and its capabilities by learning through many simple examples and analogies. The class will involve active student participation, discussions, and programming exercises. This approach will help you build a strong foundation in Python that you will be able to effectively apply in real-job situations and future courses.

STRINGS

A Python String

```
x_str = 'pineapple'
```



Global frame

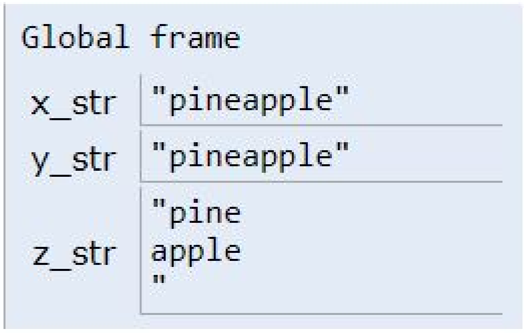
x_str "pineapple"

0	1	2	3	4	5	6	7	8
p	i	n	e	a	p	p	l	e

- object (not just an array)
- ordered and immutable
- many built-in methods

Defining Strings

```
x_str = ’pineapple’  # single quote
y_str = "pineapple"  # double quote
# triple quotes allow multi-line strings
z_str = """pine
apple
"""
```



Global frame

x_str	"pineapple"
y_str	"pineapple"
z_str	"pine apple "

Exercise(s):

- show three ways to define the following (old English proverb) string *x_str*:

```
"after
meat
comes
mustard"
```

- how may newline characters are there in *x_str*?

String Encoding

- every character is mapped to an integer
- past: ASCII code for each charcater
- now: UTF variable length encoding

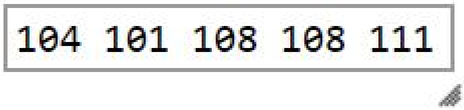
(a) international alphabets

(b) memory efficiency

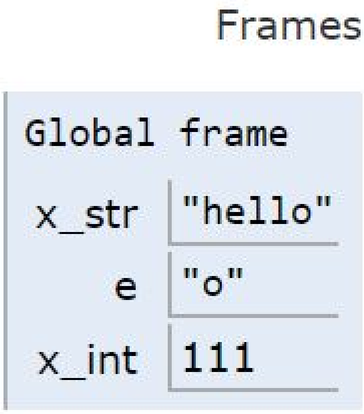
- *ord*() and *chr*() for forward and reverse mapping

ord() Function

```
# print integer values for each character
x_str = ’hello’
for e in x_str:
    x_int = ord(e)
    print(x_int, end = " ")
```



```
104 101 108 108 111
```



Frames

Global frame

x_str	"hello"
e	"o"
x_int	111

- *ord*(): maps character to its integer ”value”

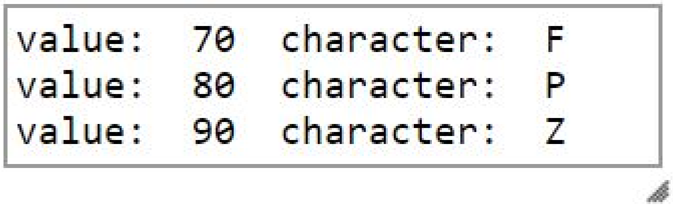
Exercise(s):

- use *ord*() to print integer values for each character in string *x_str*:

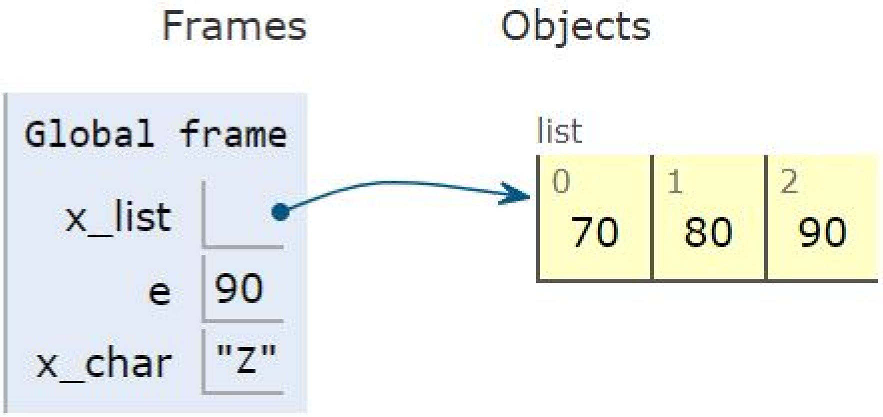
```
x_str = "Boston University"
```

chr() Function

```
x_list = [ 70, 80, 90 ]
for e in x_list:
    x_char = chr(e)
    print('value: ', e, ' character: ', x_char)
```



```
value:  70  character:  F
value:  80  character:  P
value:  90  character:  Z
```



- *chr*(): maps integer value to corresponding character

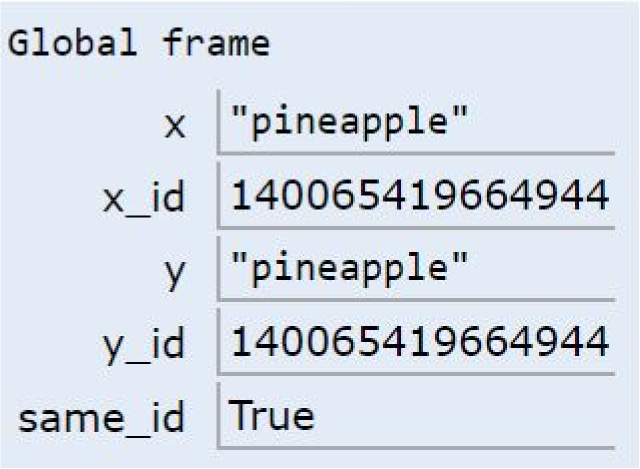
Exercise(s):

- use *chr*() to print characters for integers from 75 to 85

String Immutability

```
x = "pineapple"
x_id = id(x)
y = ’pine’ + ’apple’
y_id = id(y)
same_id = (x_id == y_id)
```

- Python strings are immutable

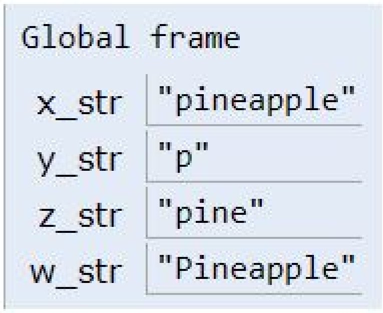


Global frame

x	"pineapple"
x_id	140065419664944
y	"pineapple"
y_id	140065419664944
same_id	True

Examples of String Methods

```
x_str   = 'pineapple'
y_str   = x_str[6]        # indexing
z_str   = x_str[0 : 4]    # slicing
w_str   = x_str.title()   # capitalize first
```



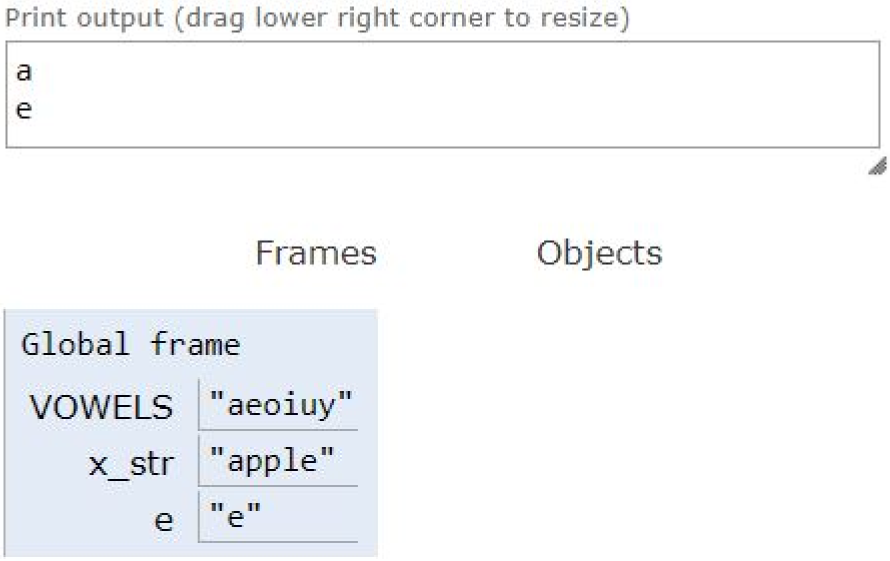
Global frame

x_str	"pineapple"
y_str	"p"
z_str	"pine"
w_str	"Pineapple"

0	1	2	3	4	5	6	7	8
p	i	n	e	a	p	p	l	e

Membership & Iteration

```
# print vowels in a string
VOWELS    = ’aeoiuy’
x_str = ’apple’

for e in x_str:
    if e in VOWELS:
        print(e)
```



```
a
e
```

Global frame

VOWELS	"aeoiuy"
x_str	"apple"
e	"e"

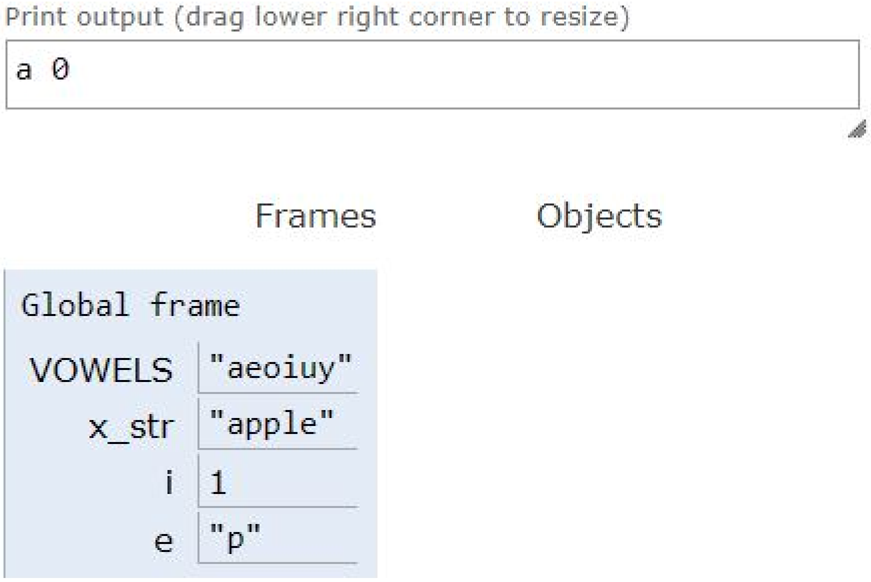
Exercise(s):

- print all consonants in string *x_str*:

```
"after
meat
comes
mustard"
```

Iteration: *enumerate*()

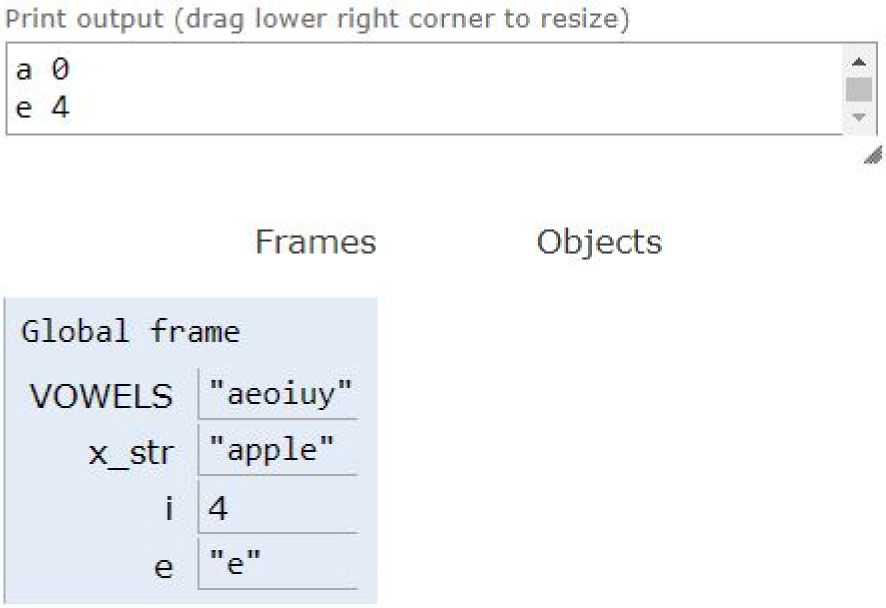
```
# print vowels and positions from string
VOWELS = 'aeoiuy'
x_str  = 'apple'

for i,e in enumerate(x_str):
    e = x_str[i]
    if e in VOWELS:
        print(e,i)
```



```
a 0
```

VOWELS	"aeoiuy"
x_str	"apple"
i	1
e	"p"

Iteration: *enumerate*() (cont'd)



```
a 0
e 4
```

Global frame	
VOWELS	"aeoiuy"
x_str	"apple"
i	4
e	"e"

- get both index and element
- use in strings, lists, tuples

Exercise(s):

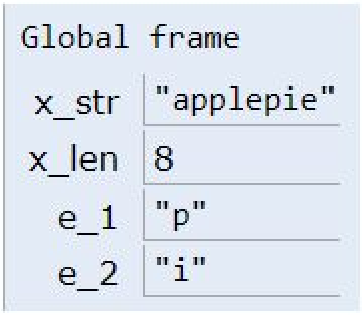
- print all consonants and positions in string *x_str*:

```
"after
meat
comes
mustard"
```

- print vowels and positions in string *x_str* without using *enumerate*()

String Indexing

```
x_str = 'applepie'
x_len = len(x_str)
e_1   = x_str[1]
e_2   = x_str[-2]
```



Global frame

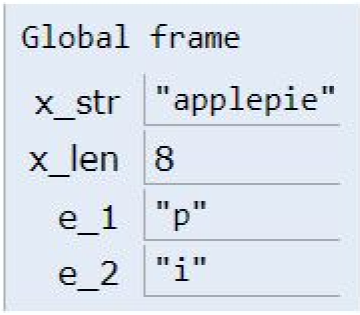
x_str	"applepie"
x_len	8
e_1	"p"
e_2	"i"

0	1	2	3	4	5	6	7
a	p	p	l	e	p	i	e
−8	−7	−6	−5	−4	−3	−2	−1

- positive and negative indices

Indexing (cont'd)

```
x_str = 'applepie'
x_len = len(x_str)
e_1   = x_str[1]
e_2   = x_str[-2]
```



Global frame

x_str	"applepie"
x_len	8
e_1	"p"
e_2	"i"

0	1	2	3	4	5	6	7
a	p	p	l	e	p	i	e
−8	−7	−6	−5	−4	−3	−2	−1

- positive = negative + length

Exercise(s):

- use positive and negative indices to extract ”7” from *x_str*:

```
x_str = "3456789abcdefg"
```

- print positive and negative indices for even digits in *x_str*:

String Slicing

```
x_str = 'applepie'
```

Global frame

x_str "applepie"

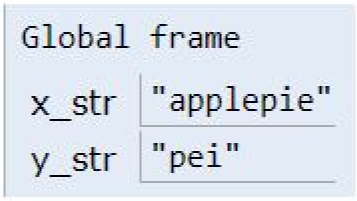
0	1	2	3	4	5	6	7
a	p	p	l	e	p	i	e
-8	-7	-6	-5	-4	-3	-2	-1

$$[\text{start} : \text{end} + 1 : \text{step}]$$

- use both pos & neg indices
- negative step for reversals

Slicing (cont'd)

```
x_str = 'applepie'

y_str = x_str[ 2 :  7 : 2]
y_str = x_str[-6 : -1 : 2]
y_str = x_str[ 2 : -1 : 2]
y_str = x_str[-6 :  7 : 2]
```



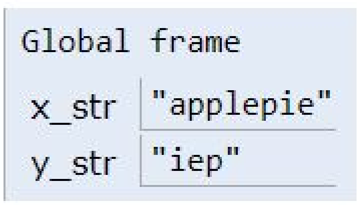
Global frame

x_str	"applepie"
y_str	"pei"

0	1	2	3	4	5	6	7
a	p	p	l	e	p	i	e
−8	−7	−6	−5	−4	−3	−2	−1

Slicing (cont’d)

```
x_str = ’applepie’

y_str = x_str[ 6 :  1 : -2]
y_str = x_str[-2 : -7 : -2]
y_str = x_str[ 6 : -7 : -2]
y_str = x_str[-2 :  1 : -2]
```



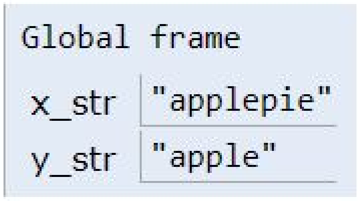
Global frame

x_str	"applepie"
y_str	"iep"

0	1	2	3	4	5	6	7
a	p	p	l	e	p	i	e
−8	−7	−6	−5	−4	−3	−2	−1

Slicing (cont'd)

```
x_str = 'applepie'

y_str = x_str[0 : 5 : 1]
y_str = x_str[  : 5 : 1]  # assume defaults
y_str = x_str[  : 5]
```



Global frame

x_str	"applepie"
y_str	"apple"

0	1	2	3	4	5	6	7
a	p	p	l	e	p	i	e
−8	−7	−6	−5	−4	−3	−2	−1

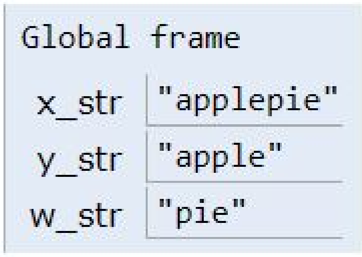
Exercise(s):

- show four different ways to extract ”wash” from *x_str*:

```
x_str = "dishwasher"
```

Slicing with Defaults

```
x_str = ’applepie’

y_str = x_str[   : 5 ]
w_str = x_str[ 5 :   ]
```



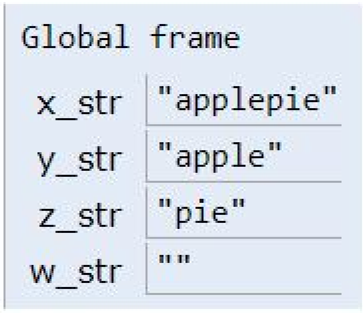
Global frame

x_str	"applepie"
y_str	"apple"
w_str	"pie"

0	1	2	3	4	5	6	7
a	p	p	l	e	p	i	e
−8	−7	−6	−5	−4	−3	−2	−1

”Out-of-Bound” Slicing

```
x_str = ’applepie’
y_str = x_str[-100  : 5]
z_str = x_str[5 : 500 ]
w_str = x_str[400 : 500]
```



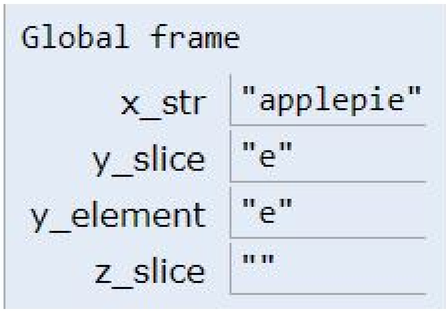
Global frame

x_str	"applepie"
y_str	"apple"
z_str	"pie"
w_str	""

- ”largest” sub-list
- no error!

Slicing vs. Indexing

```
x_str     = ’applepie’
y_slice   = x_str[4:5]
y_element = x_str[4]
z_slice   = x_str[100:101]
z_element = x_str[100]       # error
```



Global frame

x_str	"applepie"
y_slice	"e"
y_element	"e"
z_slice	""

0	1	2	3	4	5	6	7
a	p	p	l	e	p	i	e
−8	−7	−6	−5	−4	−3	−2	−1

Exercise(s):

- what is the result of the following slices from x_str?

```
"two plus two is four"
```

(a) $x\_str[10]$

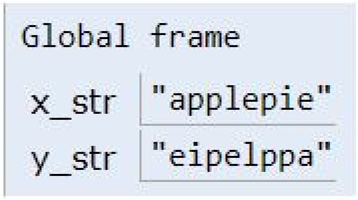
(b) $x\_str[10 : 11]$

(c) $x\_str[10 : 2000]$

(d) $x\_str[2000 : 2001]$

String Reversal

```
x_str = ’applepie’
y_str = x_str[ : : -1]
```



```
Global frame
x_str  "applepie"
y_str  "eipelppa"
```

- check if a palindrome

```
x_str = ’never odd or even’
y_str = x_str.replace(’ ’, ’’)
if y_str == y_str[ : : -1]:
    print(x_str, ’ is a palindrome’)
else:
    print(x_str, ’ is not a palindrome’)
```

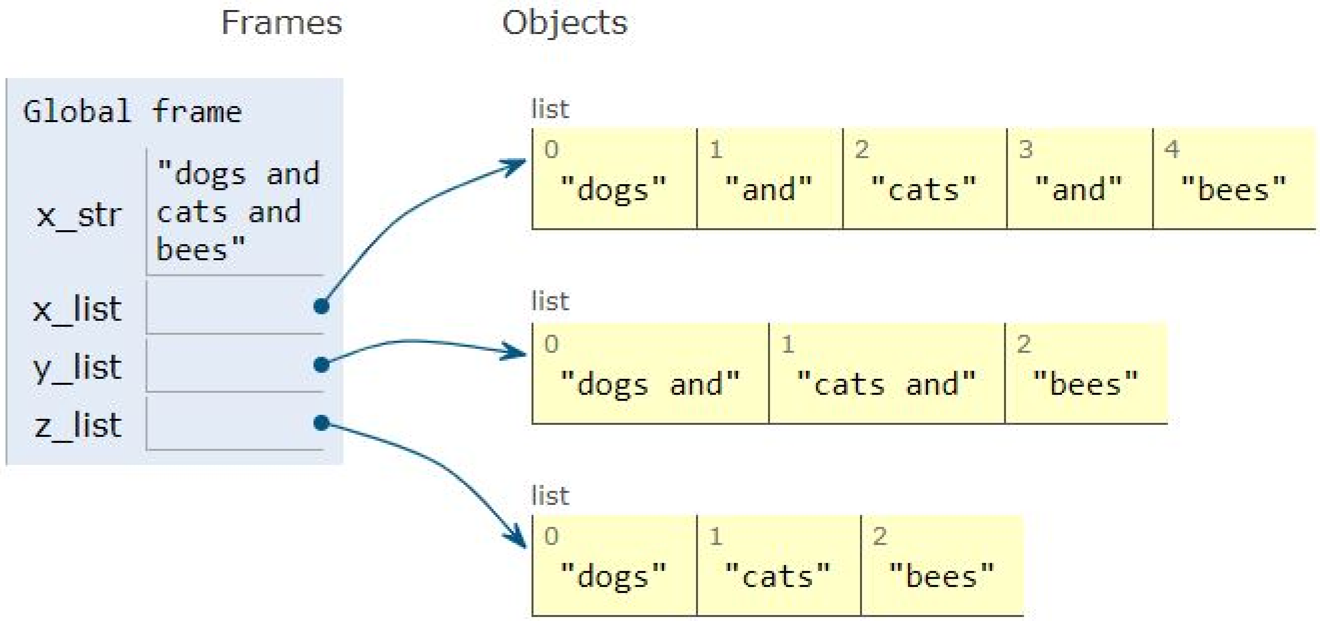
Exercise(s):

- reverse the string *x_str*:

```
"after meat comes mustard"
```

String *split*() Function

```
# split string using a separator
x_str = """dogs and
cats and
bees"""

x_list = x_str.split()
y_list = x_str.split(sep = ’\n’)
z_list = x_str.split(’ and\n’)
```



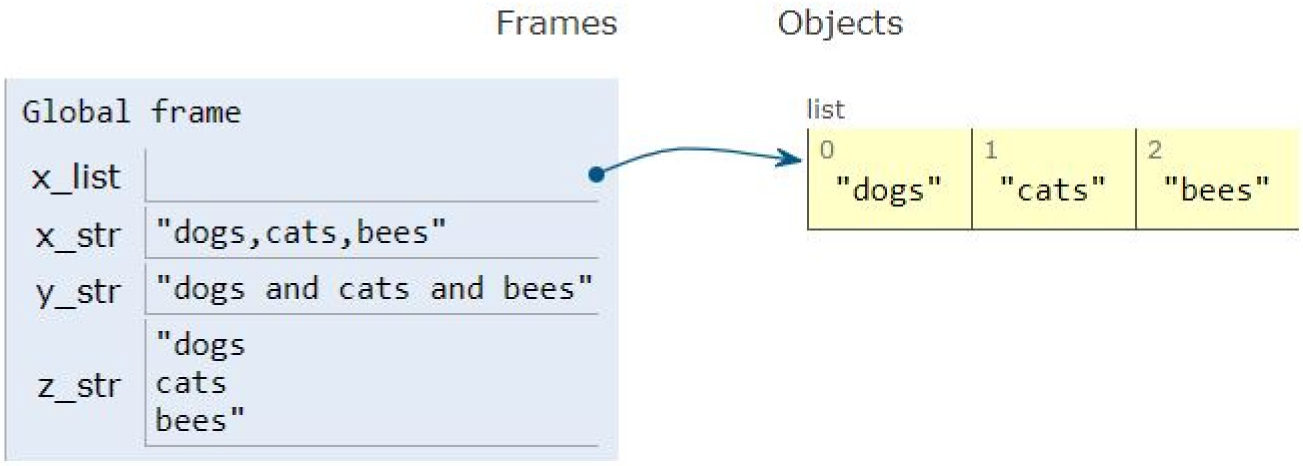
Exercise(s):

- convert a string of words x_str into a list of words x_list

```
"after meat comes mustard"
```

String *join*() Function

```
# join strings in list with separator

x_list = [’dogs’, ’cats’, ’bees’]

x_str = ’,’.join(x_list)

y_str = ’ and ’.join(x_list)

z_str = ’\n’.join(x_list)
```



Global frame

x_list

x_str "dogs,cats,bees"

y_str "dogs and cats and bees"

z_str "dogs
cats
bees"

"dogs" "cats" "bees"

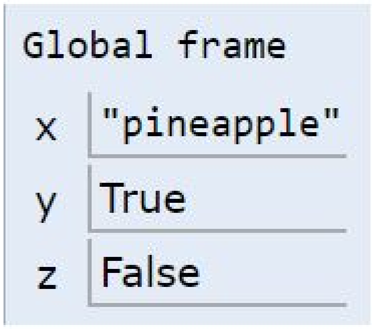
Exercise(s):

- using *split*() and *join*() replace spaces with '$' in the string *x_str*:

```
"after meat comes mustard"
```

String Methods

```
x = "pineapple"
y = x.startswith("pi")
z = x.endswith("LE")
```



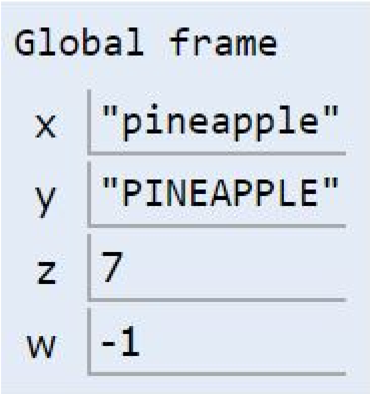
Global frame

x	"pineapple"
y	True
z	False

- many methods (around 50) available
- this makes Python very useful to use for text processing

String Methods (cont'd)

```
x = "pineapple"
y = x.upper()
z = y.find("L")
w = y.find("L", 0, 5)
```



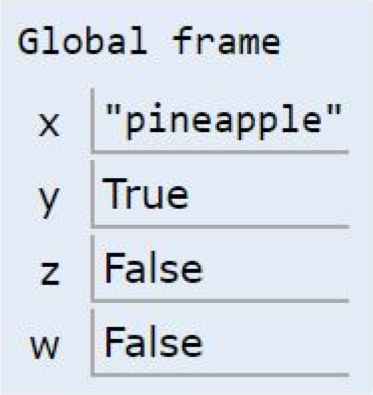
Global frame

x	"pineapple"
y	"PINEAPPLE"
z	7
w	-1

- find position
- can specify substring

String Methods (cont'd)

```
x = "pineapple"
y = x.islower()
z = x.isupper()
w = x.isdigit()
```



Global frame

x	"pineapple"
y	True
z	False
w	False

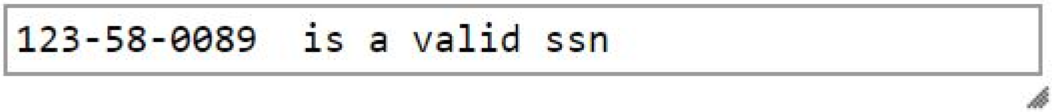
- many methods to check formats

String Methods (cont’d)

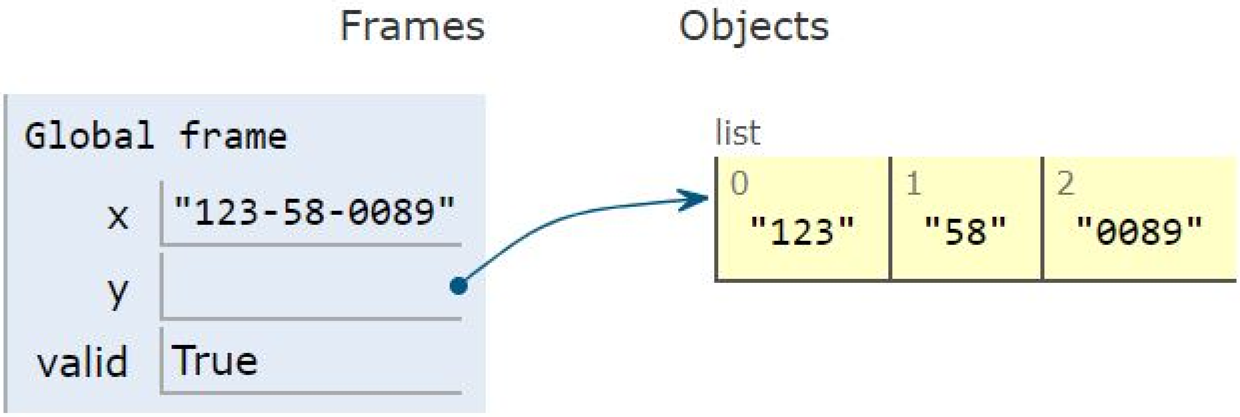
```
x = "123-58-0089";
y = x.split("-")
valid = False
if len(y)==3:
    if (y[0].isdigit() is True) and \
       (y[1].isdigit() is True) and \
       (y[2].isdigit() is True):
           valid = True
if valid is True:
    print(x,’ is a valid ssn’)
else:
    print(x, ’is not valid ssn’)
```

- verify format for social security numbers

String Methods (cont'd)



```
123-58-0089  is a valid ssn
```



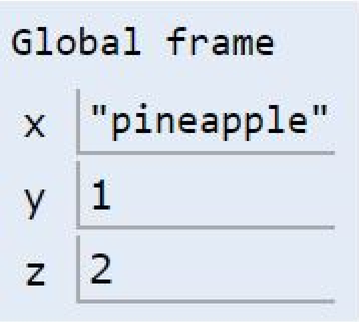
Exercise(s):

- verify that only numeric values are entered for a date

```
x_date = "09/08/1988"
```

String Methods (cont'd)

```
x = "pineapple"
y = x.count("apple")
z = x.count("e")
```



Global frame

x	"pineapple"
y	1
z	2

- easy frequency counting

Exercise(s):

- consider string *x_str*:

`"after meat comes mustard"`

(a) count the number of times character *"m"* appears

(b) compute position of the first character *"m"*

(c) compute position of the second character *"m"*

(d) replace *"mustard"* with *"dessert"*