String Formatting

Strings

- sequence (of characters) data type
- single, double, triple quotes

```
>>> x = ‘Mozzarella’
>>> y = “Gouda”
>>> z = “”””Brie,
            Cheddar””””
```

Frames Objects

Global frame	
x	"Mozzarella"
y	"Gouda"
z	"Brie, Cheddar"

- every character is mapped to an integer
- Python uses UTF-8 encoding
- *ord*() / *chr*() - forward / reverse mapping

Escape Characters in Print

- want to print special characters
- tab (‘\t’), bell (‘\a’), newline (‘\n’)
- use backslash
  string with space as separator

```
>>> print(‘Mozzarella\n\tGouda\t\tBrie’)
Mozzarella
        Gouda                Brie
```

- \\ to display single \

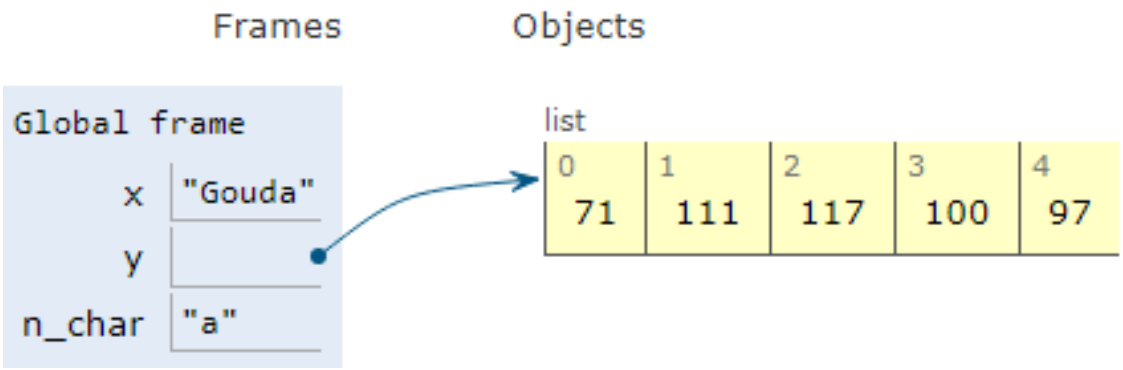
```
>>> print(“C:\\users\\downloads”)
C:\users\downloads
```

- \” to print double quote

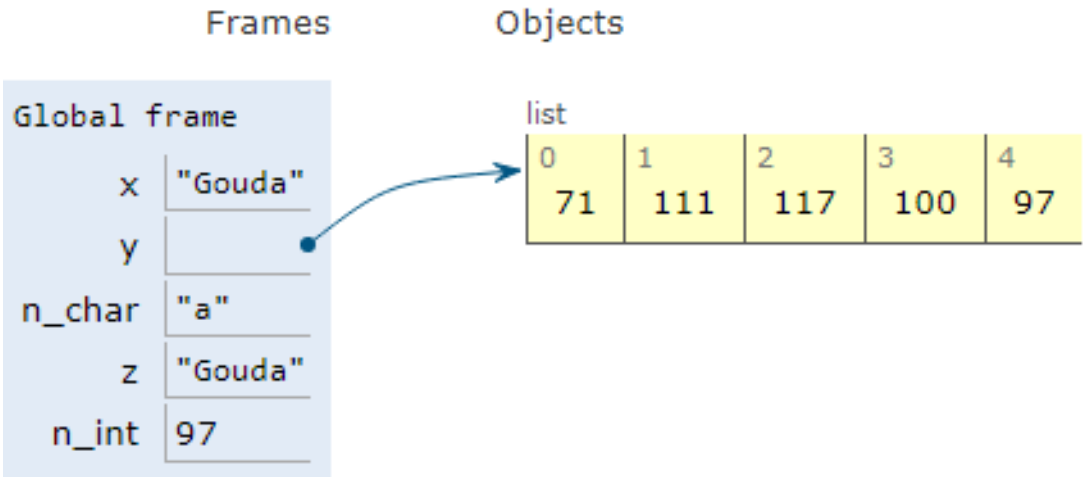
```
>>> print(‘\”hello\”’)
“hello”
```

ord() / *chr*() Functions

```
>>> x = 'Gouda'; y = []
>>> for n_char in x:
        y.append(ord(n_char))
```

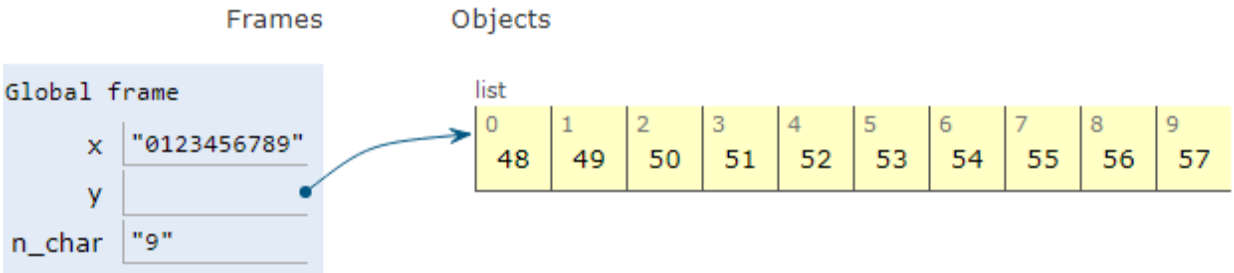



```
>>> z = ""
>>> for n_int in y:
        z = z + chr(n_int)
```

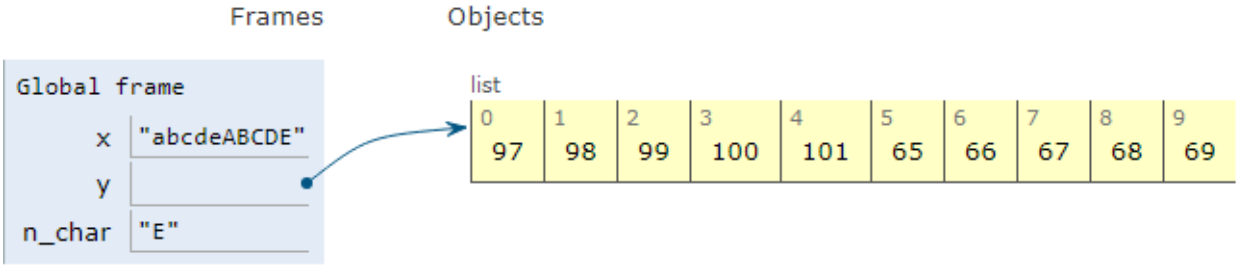


Exampled of UTF-8 Encoding

```
>>> x = '0123456789' y = []
>>> for n_char in x:
        y.append(ord(n_char))
```




```
>>> x = 'abcdeABCDE'; y = []
>>> for n_char in x:
        y.append(ord(n_char))
```



- note that '012' < 'ABC' < 'abc'

Copying Strings

- strings are immutable, same id

```
>>> x = 'Mozzarella';  id_x = id(x)
>>> y = x[ : ];        id_y = id(y)
```

Frames Objects

Global frame	
x	"Mozzarella"
id_x	140626392183856
y	"Mozzarella"
id_y	140626392183856

- modify string and extract: new id

```
>>> x = 'Mozzarella';  id_x = id(x)
>>> z = (x + '.')[: -1];  id_z = id(z)
```

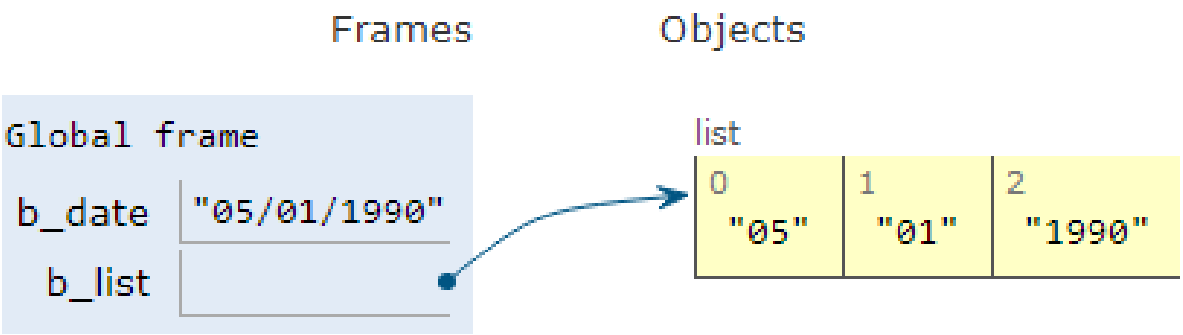
Frames Objects

Global frame	
x	"Mozzarella"
id_x	140298444702640
y	"Mozzarella"
id_y	140298444702640
z	"Mozzarella"
id_z	140298352556656

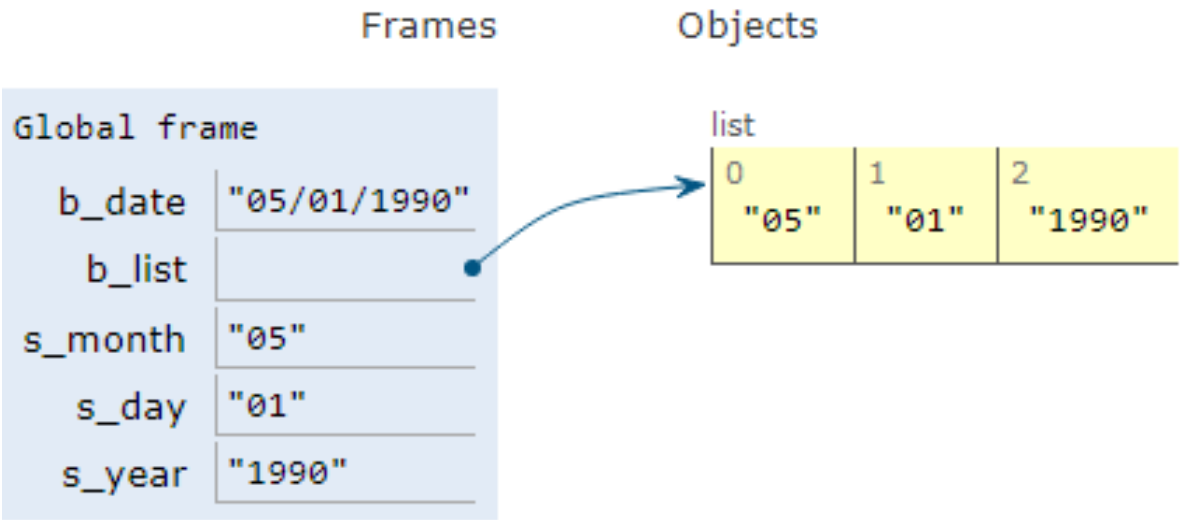
Example: Parsing Birthdate

- prompts to enter birthdata: mm/dd/yyyy

```
>>> b_date=input('birthdate mm/dd/yyyy')
>>> b_list = b_date.split('/')
```




```
>>> s_month = b_list[0]
>>> s_day = b_list[1]
>>> s_year = b_list[2]
```



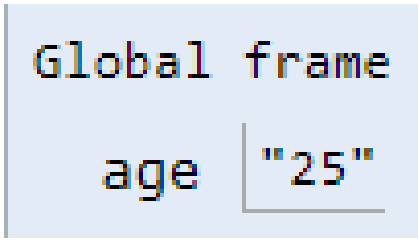
Input Function

- prompts to enter data

>>> age = *input*(‘how old are you?’)

- user responds: value and ENTER

Frames Objects



print Function

- uses string with space as separator

```
>>> print('example', 'in', 'Python')
example in Python
```

- can specify separators with *sep* args

```
>>> print('example', 'in', 'Python', sep='$')
example$in$Python
```

- print executes newline at the end
- can be changes with *end* argument

```
>>> print('example'); print('in')
example
in

>>> print('example',end="!"); print('in')
example!in
```

ljust(), rjust(), center()

- return strings of specified width
- (left/right/center) justify output in print

```
>>>print(‘01234567890123456789’);
print('Mozzarella'.rjust(15));
print('Cheddar'.rjust(15))
01234567890123456789
          Mozzarella
             Cheddar
>>>print(‘01234567890123456789’);
print('Mozzarella'.center(15));
print('Cheddar'.center(15))
01234567890123456789
      Mozzarella
        Cheddar
```

format() Function

```
>>> x = 'This is CS-{} Python {}'.format(521, 'class')
```

Frames Objects

```
Global frame
x   "This is CS-521 Python class"
```

- creates new string
- {} elements are replaced by format args
- replacement is done in order

Basic Formatting

- x is old style, y is new style

```
>>> x = '%s %s' % ('Gouda', 'Brie')
>>> y = '{} {}'.format('Gouda', 'Brie')
```

Frames Objects

Global frame	
x	"Gouda Brie"
y	"Gouda Brie"

```
>>> x = '%d %' % (5, 10)
>>> y = '{} {}'.format(5, 10)
```

Frames Objects

Global frame	
x	"5 10"
y	"5 10"

- new style can re-arrange order

```
>>> y = '{1} {0}'.format(5, 10)
```

Frames Objects

Global frame	
y	"10 5"

Format: Alignment

- <, >, ^ specify alignment

```
>>> x = '01234567890123456789 0'
>>> y = '{:>10s} has {:<3d} students'.
format('Class', 50)
```

Frames Objects

```
Global frame
x   "01234567890123456789 0"
y   "     Class has 50  students"
```

- alignment arguments

left	<
center	^
right	>

Format: Precision and Width

- can specify both precision and width

```
>>> third  = 0.333333    # 6 decimal points
>>> x = '01234567890123456789'
```

- precision is 4 decimal points

```
>>> y = 'third is {:.4f}'.format(third)
```

- precision 2 decimal points, length 10

```
>>> z = 'third is {:10.2f}'.format(third)
```

Frames Objects

Global frame

third	0.3333
x	"01234567890123456789"
y	"third is 0.3333"
z	"third is 0.33"

Format Descriptors: Types

- Inside curly brackets specify types

```
>>> x = '01234567890123456789 0'
>>> y = '{:>10s} has {:<3d} students'.
format('Class', 50)
```

Frames Objects

```
Global frame
x  "01234567890123456789 0"
y  "     Class has 50  students"
```

- common types

string	**s**
decimal	**d**
floating point (decimal)	**f**
floating point (exponential)	**e**
floating point (percent)	**%**

Format Descriptors: Width

- inside brackets specify width

```
>>> x = '01234567890123456789 0'
>>> y = '{:>10s} has {:<3d} students'.
format('Class', 50)
```

Frames Objects

```
Global frame
x   "01234567890123456789 0"
y   "     Class has 50  students"
```

- generic bracket specification:

{:align width .precision}

1. align (left, right, center)
2. width (# characters)
3. precision (for floating numbers)

Format: Detailed Example

- inside brackets specify width

```
>>> for n in range(0, 11):
        print('{:8d} -- {:4d}'.format(n,n**3))
```

```
01234567890123456789 0
       0 --    0
       1 --    1
       2 --    8
       3 --   27
       4 --   64
       5 --  125
       6 --  216
       7 --  343
       8 --  512
       9 --  729
      10 -- 1000
```

Frames Objects

Global frame

n	10

Format: Detailed Example

- Modify alignment to left justify

```
>>> for n in range(0, 11):
        print('{:8d} -- {:<4d}'.format(n,n**3))
```

```
01234567890123456789 0
       0 -- 0
       1 -- 1
       2 -- 8
       3 -- 27
       4 -- 64
       5 -- 125
       6 -- 216
       7 -- 343
       8 -- 512
       9 -- 729
      10 -- 1000
```

Frames Objects

Global frame

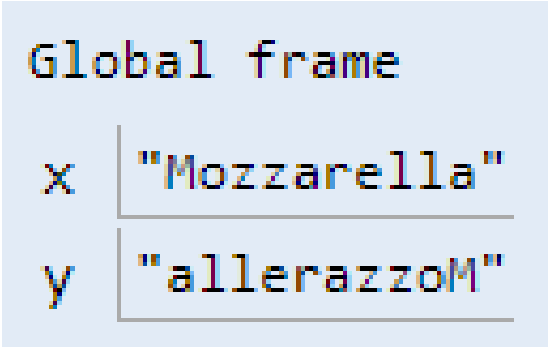
n	10

Reversing Strings

- “Pythonic” way to reverse:

```
>>> x = ‘Mozzarella’
>>> y = x[ :: -1 ];
```

Frames Objects



x[start: finish: countBy=-1]

- another alternative:

```
>>> z =””.join(reversed(x))
```

Example Palindrome

- any sequence of characters which reads the same forward or backwards
- palindromes: “noon’, ‘civic’, ‘kayak’

```
def check_palindrome(x):   # word only
    if x == x[::-1]:
        print('{} is palindrome'.format(x))
    else:
        print('{} is not palindrome'.format(x))
```

```
>>> check_palindrome('noon')
>>> check_palindrome('morning')
```

```
noon is a palindrome
morning is not a palindrome
```