CS101 - String, Set, Dictionary and Image Processing

Lecture 7

School of Computing
KAIST

Roadmap

Last week we learned

- Sequences
  - Lists
  - Strings
  - Tuples

Roadmap

Last week we learned

- Sequences
  - Lists
  - Strings
  - Tuples

This week we will learn

- Data structures
  - String
  - Set
  - Dictionary
- Image processing

Formatting

We often want to produce nicely formatted output:

```
print("Max between " + str(x0) + " and " +
      str(x1) + " is " + str(val))
```

Formatting

We often want to produce nicely formatted output:

```
print("Max between " + str(x0) + " and " +
      str(x1) + " is " + str(val))
```

The string formatting operator `%` makes this much easier:

```
print("Max between %d and %d is %g" % (x0, x1, val))
```

Formatting

We often want to produce nicely formatted output:

```
print("Max between " + str(x0) + " and " +
      str(x1) + " is " + str(val))
```

The string formatting operator `%` makes this much easier:

```
print("Max between %d and %d is %g" % (x0, x1, val))
```

Formatting operator:

```
format_string % (arg0, arg1, .... )
```

Formatting

We often want to produce nicely formatted output:

```
print("Max between " + str(x0) + " and " +
      str(x1) + " is " + str(val))
```

The string formatting operator `%` makes this much easier:

```
print("Max between %d and %d is %g" % (x0, x1, val))
```

Formatting operator:

```
format_string % (arg0, arg1, .... )
```

Tuple has one element for each place holder in the *format_string*.
Place holders are:

- `%d` for integers in decimal
- `%g` for float
- `%.2f` for float with fixed precision (2 digits after period)
- `%s` for anything (like `str(x)`)

Formatting

If there is only one place holder, tuple is not necessary:

```
print("Maximum is %g" % val)
```

Formatting

If there is only one place holder, tuple is not necessary:

```
print("Maximum is %g" % val)
```

We can align table by using field width:

```
print("%3d ~ %3d: %10g" % (x0, x1, x2))
```

Formatting

If there is only one place holder, tuple is not necessary:

```
print("Maximum is %g" % val)
```

We can align table by using field width:

```
print("%3d ~ %3d: %10g" % (x0, x1, x2))
```

A value can be left-aligned in its field:

```
print("%3d ~ %-3d: %-12g" % (x0, x1, x2))
```

Strings

Strings are sequences:

```
def is_palindrome(s):
    for i in range(len(s) // 2):
        if s[i] != s[len(s) - i - 1]:
            return False
    return True
```

Strings

Strings are sequences:

```
def is_palindrome(s):
    for i in range(len(s) // 2):
        if s[i] != s[len(s) - i - 1]:
            return False
    return True
```

Strings are immutable.

Strings

Strings are sequences:

```
def is_palindrome(s):
    for i in range(len(s) // 2):
        if s[i] != s[len(s) - i - 1]:
            return False
    return True
```

Strings are immutable.

The `in` operator for strings:

```
>>> "abc" in "01234abcdefg"
True
>>> "abce" in "01234abcdefg"
False
```

Different from the `in` operator for lists and tuples, which tests whether something is equal to an element of the list or tuple.

String methods

String objects have many useful methods:

- `upper()`, `lower()` and `capitalize()`
- `isalpha()` and `isdigit()`
- `startswith(prefix)` and `endswith(suffix)`
- `find(str1)`, `find(str1, start)` and `find(str1, start, end)`
- `replace(str1, str2)`
- `rstrip()`, `lstrip()` and `strip()`
- `split()` and `split(sep)`
- `join(list1)`

All methods are described in the Python document.

Set

Python has a data type to implement sets in mathematics. A set is a collection of distinct objects, and therefore there can be no duplicated elements in a set.

Set

Python has a data type to implement sets in mathematics. A set is a collection of distinct objects, and therefore there can be no duplicated elements in a set. To create a set object, we can use curly braces or the `set()` function.

```
>>> odds = {1, 3, 5, 7, 9}
>>> evens = {2, 4, 6, 8, 10}
>>> emptyset = set() # {} creates an empty dictionary
>>> randomset = {4, 6, 2, 7, 5, 2, 3} # Duplicated ele.
```

Set

Python has a data type to implement sets in mathematics. A set is a collection of distinct objects, and therefore there can be no duplicated elements in a set. To create a set object, we can use curly braces or the `set()` function.

```
>>> odds = {1, 3, 5, 7, 9}
>>> evens = {2, 4, 6, 8, 10}
>>> emptyset = set() # {} creates an empty dictionary
>>> randomset = {4, 6, 2, 7, 5, 2, 3} # Duplicated ele.

>>> odds
{9, 3, 5, 1, 7}
>>> evens
{8, 10, 2, 4, 6}
>>> emptyset
set()
>>> randomset
{2, 3, 4, 5, 6, 7}
```

Set

We can convert a list to a set

```
>>> gold = [0, 4, 5, 10, 3, 0, 2, 1, 4, 8, 1, 0, 1,
 ↪  0, 0, 8, 11, 4, 13, 1, 2, 3, 2, 6, 1, 9]
>>> gold
[0, 4, 5, 10, 3, 0, 2, 1, 4, 8, 1, 0, 1,
 0, 0, 8, 11, 4, 13, 1, 2, 3, 2, 6, 1, 9]
>>> goldset = set(gold)
>>> goldset
{0, 1, 2, 3, 4, 5, 6, 8, 9, 10, 11, 13}
>>> type(goldset)
<class 'set'>
```

Set

We can convert a list to a set

```
>>> gold = [0, 4, 5, 10, 3, 0, 2, 1, 4, 8, 1, 0, 1,
 ↪  0, 0, 8, 11, 4, 13, 1, 2, 3, 2, 6, 1, 9]
>>> gold
[0, 4, 5, 10, 3, 0, 2, 1, 4, 8, 1, 0, 1,
 0, 0, 8, 11, 4, 13, 1, 2, 3, 2, 6, 1, 9]
>>> goldset = set(gold)
>>> goldset
{0, 1, 2, 3, 4, 5, 6, 8, 9, 10, 11, 13}
>>> type(goldset)
<class 'set'>
```

We can also convert a string to set

```
>>> set("Good morning!")
{'G', 'm', 'i', 'd', 'o', '!', 'g', 'n', 'r', ' '}
```

Set

A set does not have ordering, so indexing is not supported.

```
>>> odds[1]
TypeError: 'set' object does not support indexing
```

Set

A set does not have ordering, so indexing is not supported.

```
>>> odds[1]
TypeError: 'set' object does not support indexing
```

We use `in` operator for sets

```
>>> 3 in odds
True
>>> 2 in odds
False
>>> for num in odds:
...     print(num)
9
3
5
1
7
```

Set methods

The set objects `s` have the following methods

- `s.add(v)` : adds an element `v`
- `s.remove(v)` : removes an element `v`
- `s.pop()` : removes and returns an arbitrary element
- `s.intersection(k)` : returns the intersection between the sets `s` and `k` (i.e., $s \cap k$)
- `s.union(k)` : returns the union of the sets `s` and `k` (i.e., $s \cup k$)
- `s.difference(k)` : removes elements found in a set `k` (i.e., $s \cap k^c$)

Set methods

Examples of using the set methods

```
>>> randomset
{2, 3, 4, 5, 6, 7}
>>> randomset.add(9)
>>> randomset
{2, 3, 4, 5, 6, 7, 9}
>>> randomset.remove(7)
>>> randomset
{2, 3, 4, 5, 6, 9}
>>> randomset.pop()
2
>>> randomset
{3, 4, 5, 6, 9}
```

Set methods

Examples of using the set methods - continued

```
>>> randomset
{3, 4, 5, 6, 9}
>>> randomset.intersection(odds)
{9, 3, 5}
>>> randomset.union(evens)
{2, 3, 4, 5, 6, 8, 9, 10}
>>> randomset.difference(odds)
{4, 6}
>>> odds.difference(randomset)
{1, 7}
>>> randomset.difference(odds, evens)
set()
```

Dictionary

Another useful data structure in Python is *dictionary*.

Similar to lists and sets, a dictionary is a collection of values. However, a dictionary can be accessed by using multiple types of indexes (i.e., not only integers, but also strings and any immutable types of objects).
Indexes used for a dictionary are called *keys*, and a key is associated with a *value*. This is called a *key-value pair*.

Dictionary

Another useful data structure in Python is *dictionary*.

Similar to lists and sets, a dictionary is a collection of values. However, a dictionary can be accessed by using multiple types of indexes (i.e., not only integers, but also strings and any immutable types of objects).
Indexes used for a dictionary are called *keys*, and a key is associated with a *value*. This is called a *key-value pair*.

To create a dictionary, we can use curly braces or the `dict()` function.

```
majors = {"CS": "Computer Science",
          "EE": "Electrical Engineering",
          "MAS": "Mathematical Sciences",
          "ME": "Mechanical Engineering"}

d1 = dict() # an empty dictionary
d2 = {} # an empty dictionary
```

Dictionary

A dictionary does not have ordering, and only the keys that are defined in a dictionary can be used as an index.

```
>>>majors[0]
KeyError: 0
```

Dictionary

A dictionary does not have ordering, and only the keys that are defined in a dictionary can be used as an index.

```
>>>majors[0]
KeyError: 0
```

We can add a key with a value to a dictionary.

```
>>> majors["PH"] = "Physic"
>>> majors["PH"]
'Physic'
```

Dictionary

A dictionary does not have ordering, and only the keys that are defined in a dictionary can be used as an index.

```
>>>majors[0]
KeyError: 0
```

We can add a key with a value to a dictionary.

```
>>> majors["PH"] = "Physic"
>>> majors["PH"]
'Physic'
```

We can also change the value via the key in the dictionary.

```
>>> majors["PH"] = "Physics"
>>> majors["PH"]
'Physics'
```

Dictionary methods

A dictionary object `d` has the following methods and operators

- `len(d)`: returns the number of elements in `d`
- `key in d`: returns `True` if `d` has the `key`, otherwise returns `False`
- `d.get(key, default=None)`: Returns the value that corresponds to the `key`, or returns the `default` value if the key is not defined in `d`)
- `d.keys()`: returns a list of keys in `d`
- `d.values()`: returns a list of values in `d`
- `d.items()`: returns a list of key-value pairs in `d`
- `del d[key]`: removes the key-value pair that corresponds to the `key`

The objects that are returned from `keys()`, `values()` and `items()` are not list objects.
They have elements like lists, but they cannot be modified and do not have an `append()` method.

Dictionary methods

KAIST

Examples of using the dictionary methods

```
>>> majors
{0: 0.001, 'CS': 'Computer Science', 'PH': 'Physics',
'ME': 'Mechanical Engineering', 'EE': 'Electrical Engineering',
'MAS': 'Mathematical Sciences'}
>>> len(majors)
6
>>> del majors[0]
>>> majors
{'CS': 'Computer Science', 'PH': 'Physics',
'ME': 'Mechanical Engineering', 'EE': 'Electrical Engineering',
'MAS': 'Mathematical Sciences'}
>>> len(majors)
5
>>> "CS" in majors
True
>>> "AI" in majors
False
```

Dictionary methods

KAIST

Examples of using the dictionary methods - continued

```
>>> majors.keys()
dict_keys(['CS', 'PH', 'ME', 'EE', 'MAS'])
>>> majors.values()
dict_values(['Computer Science', 'Physics',
'Mechanical Engineering', 'Electrical Engineering',
'Mathematical Sciences'])
>>> majors.items()
dict_items([('CS','Computer Science'), ('PH','Physics'),
('ME','Mechanical Engineering'), ('EE','Electrical Engineering'),
('MAS','Mathematical Sciences')])
```

Loop in a dictionary

To loop over the keys in a dictionary, we can use the **in** operator

```
>>> for key in majors:
...     print("%s is %s." % (key, majors[key]))
CS is Computer Science.
PH is Physics.
ME is Mechanical Engineering.
EE is Electrical Engineering.
MAS is Mathematical Sciences.
```

Loop in a dictionary

To loop over the keys in a dictionary, we can use the `in` operator

```
>>> for key in majors:
...     print("%s is %s." % (key, majors[key]))
CS is Computer Science.
PH is Physics.
ME is Mechanical Engineering.
EE is Electrical Engineering.
MAS is Mathematical Sciences.
```

To loop over both keys and values in a dictionary, we can use `items()`

```
>>> for key, value in majors.items():
...     print("%s is %s." % (key, value))
CS is Computer Science.
PH is Physics.
ME is Mechanical Engineering.
EE is Electrical Engineering.
MAS is Mathematical Sciences.
```

List, Set and Dictionary

When do we use list, set or dictionary?

- If we need to manage an ordered sequence of objects
  –> Use a List
- If we need to manage an unordered set of values
  –> Use a Set
- If we need to associate values with keys, so that we can easily look up the values by the keys
  –> Use a Dictionary

List, Set and Dictionary

Using a set is more efficient than using a list when we check membership of a value.

```
import time
large_list = list(range(10000000))
large_set = set(large_list)

st = time.time()
for num in range(100000):
    if num not in large_list:
        print("What?!")
print("Running time for list: %f sec" % (time.time() - st))

st = time.time()
for num in range(100000):
    if num not in large_set:
        print("What?!")
print("Running time for set: %f sec" % (time.time() - st))
```

List, Set and Dictionary

Using a set is more efficient than using a list when we check membership of a value.

```
import time
large_list = list(range(10000000))
large_set = set(large_list)

st = time.time()
for num in range(100000):
    if num not in large_list:
        print("What?!")
print("Running time for list: %f sec" % (time.time() - st))

st = time.time()
for num in range(100000):
    if num not in large_set:
        print("What?!")
print("Running time for set: %f sec" % (time.time() - st))
```

Result:

```
Running time for list: 78.066966 sec
Running time for set: 0.010978 sec
```

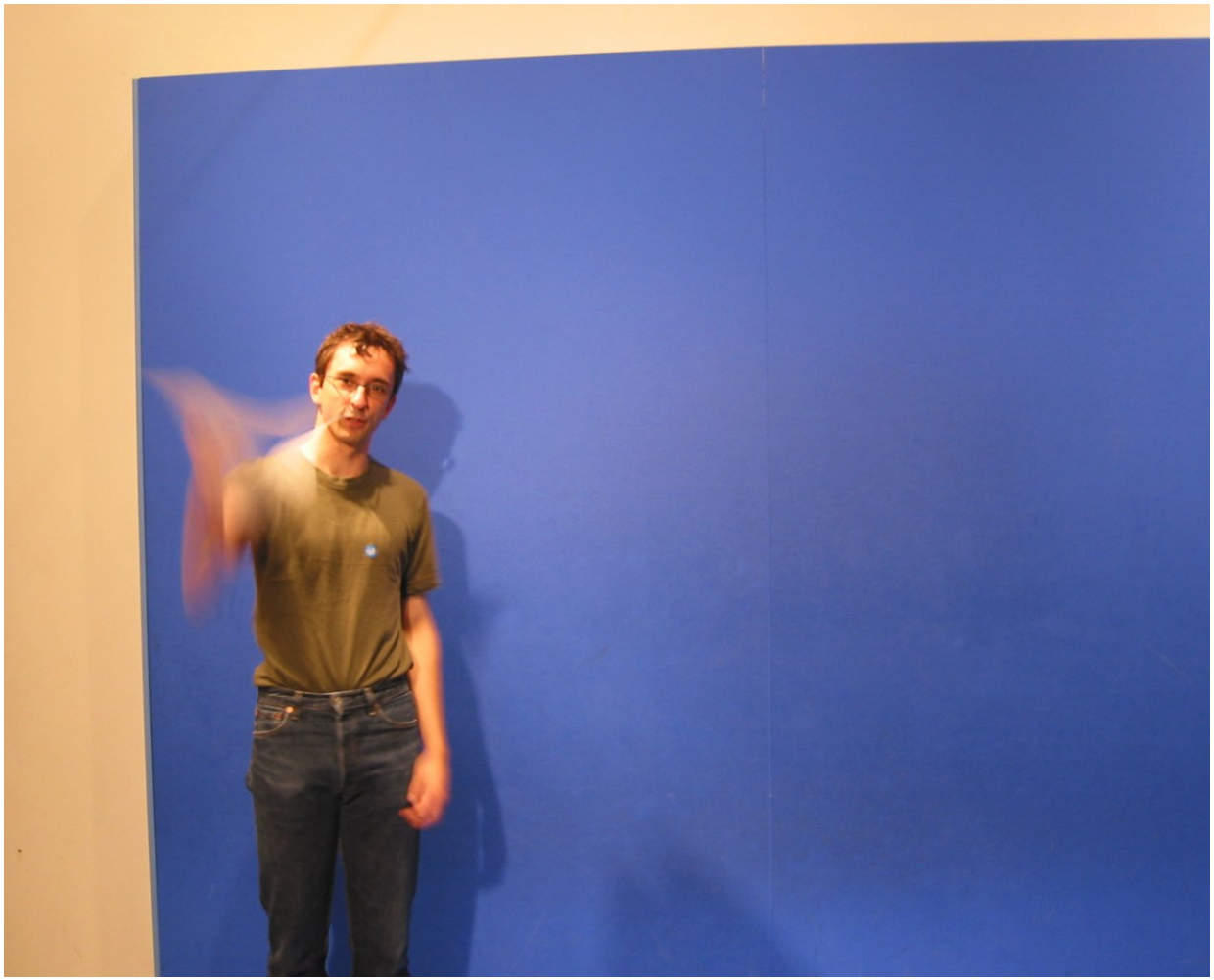
Copy and paste

Let's put the KAIST statue on a nice background:

```
def paste(canvas, img, x1, y1):
  w, h = img.size()
  for y in range(h):
    for x in range(w):
      canvas.set(x1 + x, y1 + y, img.get(x, y))
```

Chromakey

Chromakey is a technique to overlay one scene on top of another one. It is commonly used for weather maps.



Color distance

Actually, the background is not exactly blue - just blueish.
We need a function to decide how similar two colors are:

```
def dist(c1, c2):
    r1, g1, b1 = c1
    r2, g2, b2 = c2
    return math.sqrt((r1-r2)**2 + (g1-g2)**2 +
                     (b1-b2)**2)
```

Color distance

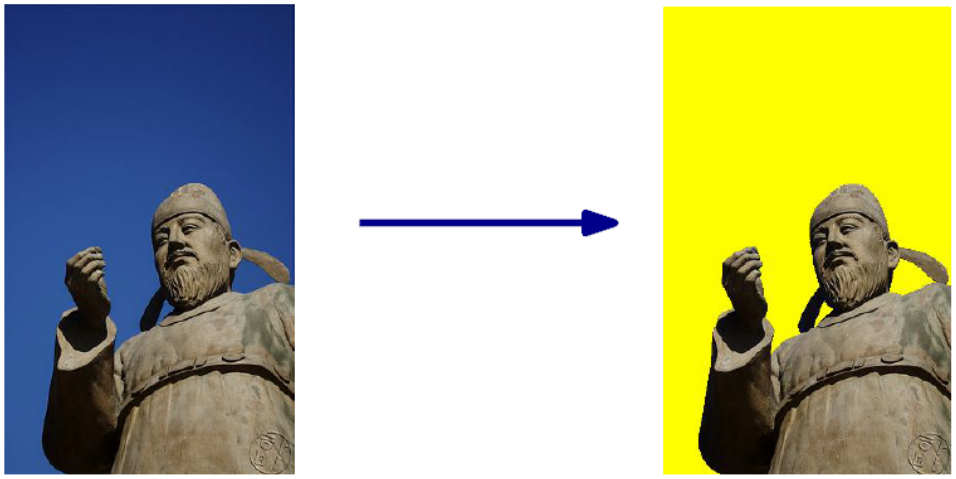
Actually, the background is not exactly blue - just blueish.
We need a function to decide how similar two colors are:

```
def dist(c1, c2):
    r1, g1, b1 = c1
    r2, g2, b2 = c2
    return math.sqrt((r1-r2)**2 + (g1-g2)**2 +
                     (b1-b2)**2)
```

This is just the Euclidean distance in $\mathbb{R}^3$.

Chromakey

```
def chroma(img, key, threshold):
  w, h = img.size()
  for y in range(h):
    for x in range(w):
      p = img.get(x, y)
      if dist(p, key) < threshold:
        img.set(x, y, Color.yellow)
```



Chromakey

Now all we need is a paste function that skips the color-coded background:

```
def chroma_paste(canvas, img, x1, y1, key):
  w, h = img.size()
  for y in range(h):
    for x in range(w):
      p = img.get(x, y)
      if p != key:
        canvas.set(x1 + x, y1 + y, p)
```

Information hiding

Humans cannot perceive a small change in light intensity or color value. We can use this to hide information inside images.

Information hiding

Humans cannot perceive a small change in light intensity or color value. We can use this to hide information inside images.

Here is an algorithm to hide a black/white image secret in an image `img`:

- For all pixels `(r, g, b)` of `img`, if `r` is odd then subtract one from `r`
- For each black pixel of secret, add one to the red value of the same pixel in `img`.

Information hiding

Humans cannot perceive a small change in light intensity or color value. We can use this to hide information inside images.

Here is an algorithm to hide a black/white image secret in an image `img`:

- For all pixels `(r, g, b)` of `img`, if `r` is odd then subtract one from `r`
- For each black pixel of secret, add one to the red value of the same pixel in `img`.

To decode the secret, we look at all pixels `(r,g,b)` of the image, and turn it black if `r` is odd, and white otherwise.