CIS 1100

Sets!

Python
Fall 2024
University of Pennsylvania

Any questions from last time?

Keyword Arguments

Sometimes we want our functions to be able to take **default values** for their inputs. We can do this with *keyword arguments*.

```
def divide(a, b, rounding=False):
    result = a / b
    if rounding:
        return round(result)
    else:
        return result
```

`rounding` is a keyword argument that is defined by its **name** as well as the **default value** that it takes if it is not replaced.

Keyword Argumennts

```
def divide(a, b, rounding=False):
    result = a / b
    if rounding:
        return round(result)
    else:
        return result
```

We can do any of the following:

```
>>> divide(3422, 194)
17.63917525773196
>>> divide(3422, 194, rounding=True)
18
>>> divide(3422, 194, True)
18
>>> divide(3422, 194, False)
17.63917525773196
```

Rules of Keyword Arguments

Signatures:

- All keyword parameters have to be provided AFTER all the positional ones
- A keyword parameter is defined by writing `identifier=<default_value>`
- Can have as many as you want, including ONLY keyword parameters

Calls:

- All keyword arguments have to be passed in AFTER all positional inputs, but from there can be in any order
- Keyword arguments can be given positionally or by name, but you should always just give thme by name

Good or Bad?

```
def fun(a, b, c=13, d):
    pass
```

Good or Bad?

```
def fun(a, b, c=13, d):
    pass
```

BAD!

Good or Bad?

```
def fun(a=13, n="haha"):
    pass
```

Good or Bad?

```
def fun(a=13, n="haha"):
    pass
```

GOOD!

Good or Bad?

```
def fun(a, b, c=, d=13):
    pass
```

Good or Bad?

```
def fun(a, b, c=, d=13):
    pass
```

BAD!

Good or Bad?

```
def fun(x, y, z=0):
    pass
```

then,

```
...
fun(3, 4, 0)
...
```

Good or Bad?

```
def fun(x, y, z=0):
    pass
```

then,

```
...
fun(3, 4, 0)
...
```

OK, but redundant?

Good or Bad?

```
def fun(x, y, z=0):
    pass
```

then,

```
...
fun(z=0, 3, 4)
...
```

Good or Bad?

```
def fun(x, y, z=0):
    pass
```

then,

```
...
fun(z=0, 3, 4)
...
```

BAD!

Good or Bad?

```
def fun(x, y, z=0):
    pass
```

then,

```
...
fun(3, 4, z=x+y)
...
```

Good or Bad?

```
def fun(x, y, z=0):
    pass
```

then,

```
...
fun(3, 4, z=x+y)
...
```

BAD!

Good or Bad?

```
def fun(x, y, z=0):
    pass
```

then,

```
...
fun(3, 4)
...
```

Good or Bad?

```
def fun(x, y, z=0):
    pass
```

then,

```
...
fun(3, 4)
...
```

Good!

Review: Sets

Sets are an **unordered** container for data.

Unordered means:

- We can still use `for` to loop over every element
- can still use `in` to see if something is in it
- can **not** access into it with an index and `[]` or slice

Review: Sets

Sets look similar to lists, but with `{}` instead of `[]`

- `{"this"}`
- `{"howdy", "partner"}`

Cannot store lists, dicts or other sets within a set

Most important part of a set: it enforces uniquness of its elements. An element can only be in the set once or not at all.

More Set Review

There are a few common features of a set:

- `.add()` adds an item to the set
- `.remove()` removes an item from a set, if the item is not in the set then crash
- `.discard()` removes an item from a set, ignore if the item is already not in the set
- set comprehension work like list comprehensions, use {} instead of []

What is the final value of `my_set` after running this code? (S7)

```
my_set = {"moons", "farming", "tormented"}
my_set.add("agility")
my_set.add("agility")
my_set.remove("agility")
my_set.discard("Farming")
my_set.remove("farming")
```

Practice

Put both of these in (C12)

```
def remove_all(words, filter):
    # given a list of strings, return a new list of strings except all words
    # that are in the input set "filter" are not in the output
    # remove_all(["Hi", "There"], {"Hi"}) -> ["There"]
```

```
def count_unique_words(words):
    # given a list of strings, return the number of unique strings in that list
```

Set Operations

Name	Meaning	Method	Operator
Union	Create a new set with all elements from both	`s.union(t)`	`s | t`
Intersection	Create a new set with only elements that appear in both sets	`s.intersection(t)`	`s & t`
Difference	Create a new set with only elements in `s` that don't appear in `t`	`s.difference(t)`	`s - t`
Symmetric Difference	Create a new set with elements that appear in only one set *but not both*	`s.symmetric_difference(t)`	`s ^ t`

Set Operations Practice

Put both of these in (C14)

Implement both an intersection function and a union function without using the built-in intesection or union operators or functions.

```
def set_union(s1, s2):
    # given two sets, return a new set that has all elements of both input sets
    # set_union({"hi","ho"}, {"bad","hi"}) -> {"hi", "ho", "bad"}
```

```
def set_intersection(s1, s2):
    # given two sets, return a new set that only has the elements that are in both input sets
    # set_intersection({"hi","ho"}, {"bad","hi"}) -> {"hi"}
```

Dictionaries

Dictionaries (also called "dicts") are the much more commonly used **unordered** collection

- Associates **keys** to **values**
- Allow for looking up some information associated with a search key
- Keys must be unique, values do not need to be unique

What is a Mapping?

Any association from **keys** (things you can search by) to **values** (information you might want to know.)

The Penn Directory, for example:

```
Name : Email
Harry Smith : sharry@seas
Travis McGaha : tqmcgaha@seas
...
```

Here, the names are keys and the emails are values.

Dict Syntax

Dict literals are defined with curly braces (`{}`) and separate keys and values with a colon.

- `{3, 10, 15}`
  - is a **set** with three elements
- `{"Harry" : "sharry", "Travis" : "tqmcgaha"}`
  - is a **dict** with two elements (key-value pairs)
- `{}` is an empty dict
  - writing just `dict()` gets the same result

Dictionary Practice: Reading

Given the following dictionaries, which ones are legal dictionaries? (Legal / Illegal)

(S8)

```
speak = {
  "dog": "woof",
  "cat": "meow",
  "fish": "blub",
  "seal": "ow ow ow",
  "fox": "woof"
}
```

(S9)

```
faves = {"The Wall", "Her", "Princess Mononoke"}
```

(S10)

```
friends = {"Jamie":["Tampa"], "Hunter": ["Tampa", "Orlando"], "Zack": ["Tampa"], "Jamie": ["NYC"]}
```

Next time

- More on dictionaries!
  - I only just barely started talking about them
  - These are a really important data structure in Python and Programming in general