CIS 1100

Sets!

Python
Spring 2025
University of Pennsylvania

Any questions from last time?

main

So far in this class we have put all our code directly in the py file.

Now that we know functions, we should follow better practices.

Starting with HW03 onwards, all code should go in a function.

- import statements should still be at the top of the py file outside of functions
- Comments can be outside of functions

Hello World w/ main()

From now on we will have a function called `main()` that will be where we keep the code we previously did not put in a function

We also need to add an if statement to the bottom of our code. Do not forget it!

For example:

```
def main():
    print("Hello World!")

if __name__ == "__main__":
    main()
```

main() practice (C16)

rewrite the following code so that we use the new style that uses:

`def main():` and

`if __name__ == "__main__":`

```
import sys

# Prints all command line args
def print_args():
  for arg in sys.argv:
    print(arg)

print("Hello!")
print_args()
```

Why `main()`?

Writing our code in a `main()` function has a few benefits:

- Usually means our code is easier to read and better organized
- We can now import functions from the python files we write
  - this is the most important point
  - means we can import our own code into separate files for testing (or for the REPL)
- This is also just a style convention followed by most programming languages

Inputs, Parameters, Arguments

Often times in this class we call the inputs to functions "Inputs"

```
def add(x, b):
    return x + b

zinc = 64
add(3, zinc)
```

Previously we would call `x`, `b`, `3`, `zinc` all "inputs"

Technically we would make the distinction where

`x` and `b` are **parameters** (The variables defined in the function header)

`3` and `zinc` are **arguments** (The actual data passed into the function)

You do not need to memorize this distinction, this is just for your future use and so you know what we mean when we say parameter and argument of a function.

Inputs should be inputs

Quick: What gets printed from this code? (S10)

```
def mystery(x):
    x += 5

def plank(name):
    name.upper()

def main():
    number = 3
    artist = "Kuji"
    mystery(number)
    plank(artist)

    print(number)
    print(artist)
```

```
if __name__ == "__main__":
    main()
```

Inputs should be inputs

When we pass most things to a function, we pass in a copy of it or we pass something that cannot be mutated.

We generally want our inputs to be inputs only. If we get something from a function it should be returned

Lists as parameters

Consider this code though:

```
def modify_list(nums):
    nums.append(3)

def main():
    my_numbers = [2, 5]
    modify_list(my_numbers)
    print(my_numbers) # prints "[2, 5, 3]"

if __name__ == "__main__":
    main()
```

Lists (and sets and dicts) are special in that they can be modified when used as a function input. Why? (more later)

Be careful not to modify a list in a function that was taken as input. Our homework code checks to make sure you don't do this.

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",
  "seal": "arf",
  "fox": "woof"
}
```

(S9)

```
last_year_movies = {"Anora", "Conclave", "A Real Pain"}
```

(S10)

```
recs = {201:["Zwe", "Hannah"], 203: ["Adi", "Sofia"],
        204: ["Jana", "Hannah"], 201: ["Hannah", "Zwe"]}
```

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