CS101: Intro to Computing
Fall 2015

Lecture 12

Administrivia

- Homework 10 is assigned
  - Due ***Monday***
- Midterm results coming Monday

REVIEW

```
s="WTE"
t="ANY"
u=[]
for a,b in zip(s,t):
  u.append(a+b)
x=''.join(u)
```

What is the final value of x?

a) ""

b) "AWNTYE"

c) "WTEANY"

d) "WATNEY"

```
s="234"
x=0
for a,b in enumerate(s):
  x+=int(b)+a
```

What is the final value of x?

a) 12

b) 237

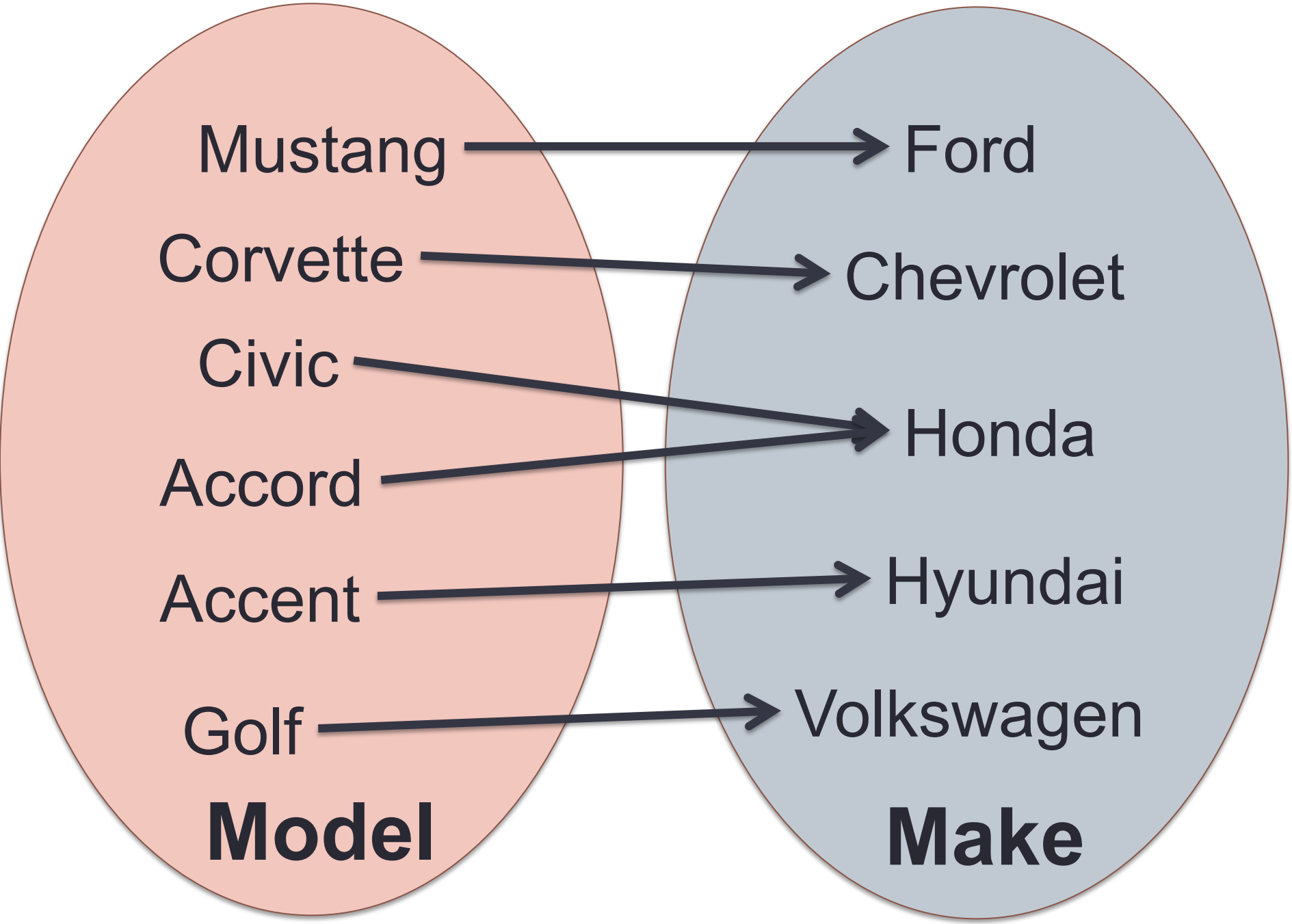
c) 9

d) 15

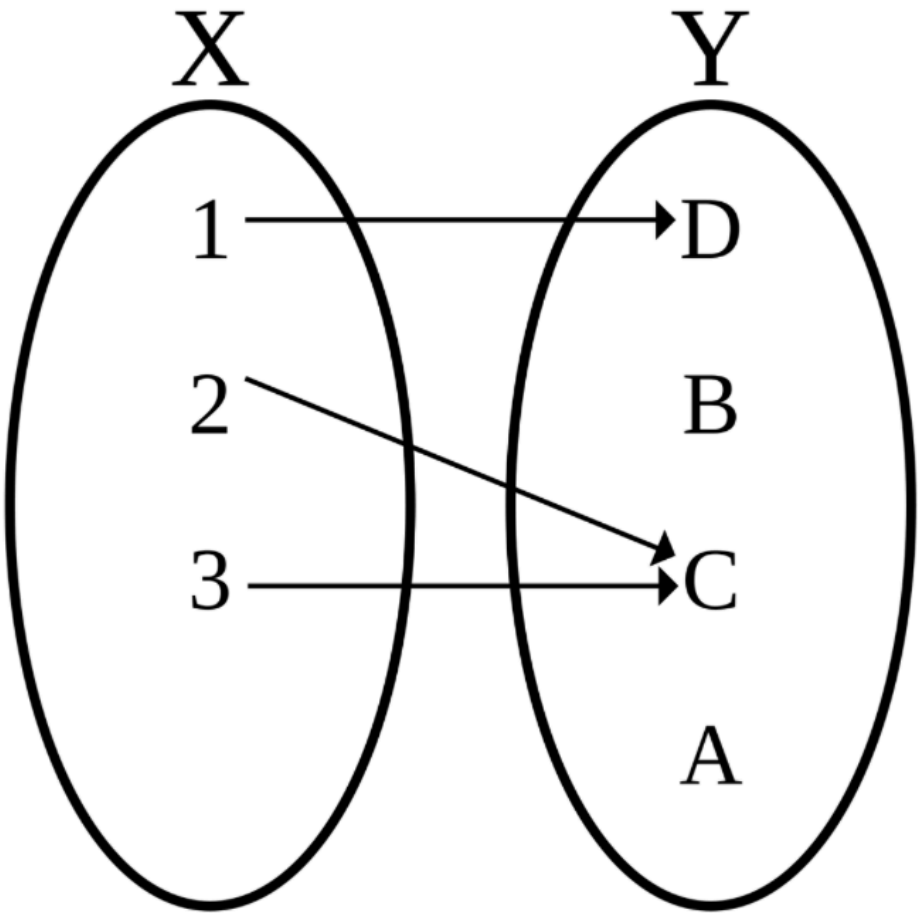
DICTIONARIES

Types we've learned

- Lists and tuples are *ordered*
- Lists and tuples are *indexed using an integer*
- It's natural to associate data with an identifier:
  - Person → birthday, gender, parents
  - Country → flag, median income


Mustang
Corvette
Civic
Accord
Accent
Golf
Model
Ford
Chevrolet
Honda
Hyundai
Volkswagen
Make

Mapping



Dictionaries

- Represents an ***unordered*** collection of items or elements
- A ***container*** type
  - Contains other values of ***any type***
  - **NOTE*:*** elements don't have to be the same type
- Can be indexed with ***any type***
- Map ***keys*** to ***values***

Dictionary literals

- We create an ***dictionary*** by typing:

1. an open curly bracket {
2. a key, a colon, and its associated value
3. key:value pairs separated by commas
4. a closing curly bracket }

Dictionary

```
model={“Civic”:”Honda”,
   ”Mustang”:”Ford”,
   ”Corvette”:”Chevy”,
   ”Accord”:”Honda”,
   ”Accent”:”Hyundai”}
```

Dictionary Operations

```
d={“one”:1,”two”:2,”three”:3}
print d[“one”]
d[“four”]=4
del d[“four”]
“five” in d
for key in d:
  print key,d[key]
```

NO GUARANTEE ON ORDER!

```
d={“a”:2,”c”:3,”b”:1}
x=d[“a”]+d[“c”]
```

What is the final value of x?

a) 3

b) “ac”

c) 4

d) 5

```
d={}
for i,j in enumerate(“ABC”):
  d[j]=i
```

What is the final value of d?

a) {“A”:0,”B”:1,”C”:2}

b) {0:”A”,1:”B”,2:”C”}

c) {“A”:1,”B”:2,”C”:3}

d) {1:”A”,2:”B”,3:”C”}

```
d={}
for i,j in zip(“WAT”,”NEY”):
  d[(i,j)]=j
```

Which of these expressions evaluates to “E”?

a) `d[“A”]`

b) `d[“E”]`

c) `d[“A”,”E”]`

d) `d[“EA”]`

USES FOR DICTIONARIES

Dictionaries to Encode

- We can use dictionaries to encode/decode data
- We can use dictionaries to translate from one representation to another

Dictionaries to Encode

```
x="ABCDEFGHIJKLMNOPQRSTUVWXYZ"
y="BCDEFGHIJKLMNOPQRSTUVWXYZA"
e={}
for i in range(len(x)):
  e[x[i]]=y[i]
encoded=""
for c in "HELLO":
  encoded+=e[c]
```

Dictionaries to Encode

```
x="ABCDEFGHIJKLMNOPQRSTUVWXYZ"
y="BCDEFGHIJKLMNOPQRSTUVWXYZA"
d={}
for i in range(len(x)):
  d[y[i]]=x[i]
decoded=""
for c in encoded:
  decoded+=d[c]
```

Exercise

- Encipher all of the words in a file with the Caesar cypher
- Decode all of the words in the file

Dictionaries as Accumulators

- We can use dictionaries as a collection of counters for many things at once

```
x="ABBACAB"
d={}
for c in x:
  if c not in d:
    d[c]=0
  d[c]+=1
```

Exercise

- Count category frequencies in Jeopardy questions
- Count bigram frequencies in Jeopardy clues

Dictionaries to Join/Merge Data

- We can link data based on a common field

```
zip={“Bill”:60644,
  ”Jim”:41073,”Beth”:63103}
city={60644:”Chicago”,
  41073:”Cincinnati”,
  63103:”St. Louis”}
for name in zipcode:
  print name,city[zipcode[name]]
```

MODULES

Modules

- A collection of Python specialized functions, variables, and even types
- We need to ***import*** the module

```
import math
```

- Can then access things within the module using ***attribute operator***

```
math.sqrt(math.pi)
```

From

- Can choose what to import with ***from***

```
from cmath import phase
phase(1+1j)
```

READABLE CODE

Writing readable code

- We should always strive to write code that is easy to read.
  - Our variables should have ***descriptive*** names.
  - We should also ***annotate*** our code.
- **REMEMBER**: A program is set of instructions a computer executes ***to achieve a goal***.

Commenting

- ***Comments*** are text that the interpreter ignores
- Comments help ***a person*** read a program
- The # symbol indicates a comment
  - Anything after that symbol is ignored

```
# Hello, I am a comment
```

Docstring

- A string literal that behaves like a comment
- Use triple quotes
- Especially useful after function definition

```
"""Hello, I am a docstring."""
```

```
s="ABCD"
s+="1"
#s+="2"
"""s+=3"""
```

What is the final value of x?

a) ABCD

b) ABCD1

c) ABCD12

d) ABCD123

Why comment/document?

- Allows us to ***explain*** our code to others.
- But mostly… to ourselves.
- Yes, ***ourselves.***

Documenting Modules

- Every script (.py) file you write is a module.
- Your modules should have a docstring at the beginning describing them and you.

```
"""
CS101 class demonstration
Author: Ryan Cunningham
"""
```

Documenting Functions

- Use doc string and describe what function does.
- Describe all parameters by name.
- Describe all return values.

Main function

- Allows our module to be imported OR run from the command line as a script
- Put the “starting point” code in a function called “main”
- This test checks if running on command line:

```
if __name__ == ‘__main__ ’:
   main()
```

DEBUGGING

“My code doesn’t work.”

“My code doesn’t work.”

“My code doesn’t work.”

1. How do I know it isn’t working?
2. What do I expect it to do?
3. What is my code doing instead? Why?

Error messages

1. Read message and ***think*** about it.

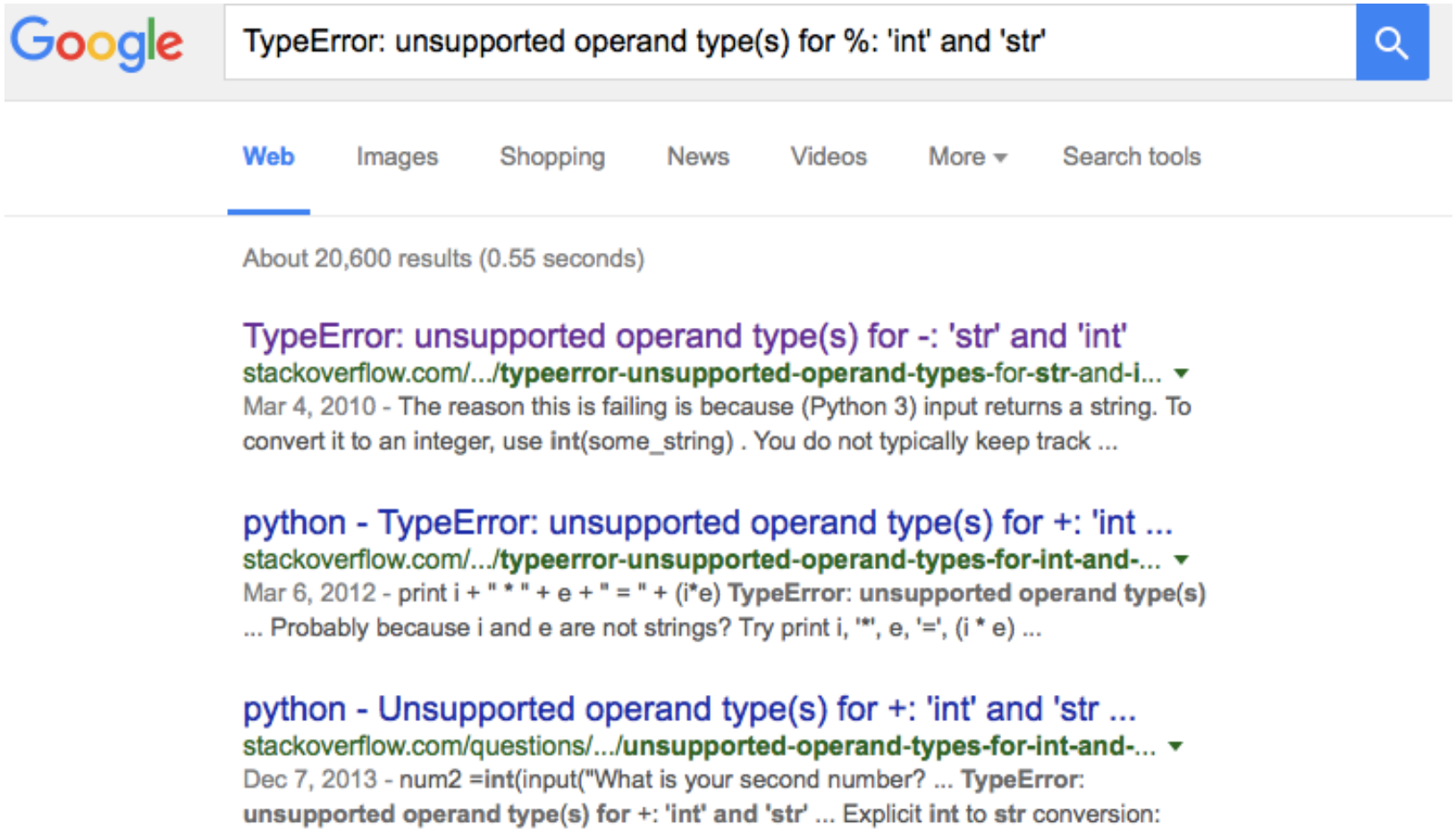
```
AttributeError: 'int' object has no attribute 'append'
```

```
IndexError: list index out of range
```

2. Google the error message

```
TypeError: unsupported operand type(s) for %: 'int' and 'str'
```

Error messages



Don’t just stare at your code!

- Make the ***computer*** work, not you!
- Add print statements (especially in loops!)

```
# Find all words whose first two letters are
# the same
for line in open("words.txt"):
        line=line.strip()
        if len(line)>=2:
                a,b=line[1:3]
                if a==b:
                        print line
```

```
ValueError: need more than 1 value
to unpack
```

“My code doesn’t work.”

```
# Find all words whose first two letters are
# the same
for line in open("words.txt"):
        line=line.strip()
        if len(line)>=2:
                print line
                a,b=line[1:3]
                print a,b
                if a==b:
                        print line
```

```
# Find all words whose first two letters are
# the same
for line in open("words.txt"):
        line=line.strip()
        if len(line)>=2:
                a,b=line[0:2]
                if a==b:
                        print line
```

Ninja coding tricks

1. Break the problem down and write functions
2. Test every function (even every ***line***) you write to make sure it works (write then execute, write then execute)
3. Comment out lines to narrow down your search for the problem

Ultimate Ninja Coding Secret

- If you're confused, frustrated, and can't make progress…

REWRITE YOUR CODE

STACKS

Stack

- A stack is an ***abstract data type***
- ***Not*** a type in Python
- We will use lists to make stacks

List methods for stack

- append()
- pop()