UNPACKING ARGUMENTS

Python Programming — Auburn University

UNPACKING LIST AS POSITIONAL ARGUMENTS

- Sometimes we end up with the collection of positional arguments in a *list*.
- Python makes it easy to "unpack" that list into a collection of arguments:

```
def f(x, y): return 2*x-y

values = [3, 2]
print(f(*values))     # x = 3, y = 2, prints 4
```

- This is a nice feature but don't put your arguments in a list just to use it :)

UNPACKING CONTINUED

- You can combine multiple unpacked lists and normal positional arguments:

```
def f(a, b, c, d, e, f):
    return a+b+c+d+e+f

val1 = [5, 3, 2]
val2 = [4, 22]
print(f(*val1, 9, *val2))  # prints 45
```

UNPACKING KEYWORD ARGUMENTS

- Sometimes we end up with arguments values in a *dictionary*.
- Python makes it easy to "unpack" a dictionary into keyword arguments:

```
def f(x, y): return 2*x-y

values = {"y": 2, "x": 3}
print(f(**values))    # x = 3, y = 2, prints 4
```

- Note * (list-based positional unpacking) vs ** (dictionary-based keyword unpacking)

UNPACKING KEYWORD ARGS CONTINUED

- Unpacked keyword arguments can be combined with positional arguments
  - but the positional arguments must proceed all unpacked keyword arguments
- Unpackaged keyword arguments can be combined with other unpacked keyword arguments.
- Examples to follow…

COMBINING WITH POSITIONAL ARGUMENTS:

```
def f(a, b, c, d, e, f):
    return a+b+c+d+e+f

val1 = {“c”: 2}
val2 = {“d”: 9, “e”: 4, “f”: 22}
val3 = {“a”: 1, “b”: 2}
print(f(3, 5, **val1, **val2))   # prints 45
print(f(**val3, 15, **val2))  # error, positional argument
                              # after unpacked keyword argument
```