ATMS 305 WEEK 5: WORKING WITH TEXT

Lecture 1: Strings, formatting, and reading a text file

USING DICTIONARIES

Last time, we showed how to use dictionaries to organize data that we read in.

```
filename='aravg.ann.land_ocean.90S.90N.v4.0.1.201612.asc'
wxdata={'year':[],'temperature':[]}    #define dictionary
with open(filename, "r") as f:    #read all the lines in the file
    alist = f.read().splitlines()
for line in alist:  #iterate through lines
    wxdata['year'].append(int(line.split()[0]))
    wxdata['temperature'].append(float(line.split()[1]))
```

How can we use the dictionary for some common tasks?

QUERYING DATA

We can use several techniques to query list data.

Lists can be queried using built in commands:

```
wxdata['year'].count(1986) #Try this
```

Function with Description
cmp(list1, list2) Compares elements of both lists.
len(list) Gives the total length of the list.
max(list) Returns item from the list with max value.
min(list) Returns item from the list with min value.
list(seq) Converts a tuple into list.

Methods with Description
list.append(obj) Appends object obj to list
list.count(obj) Returns count of how many times obj occurs in list
list.extend(seq) Appends the contents of seq to list
list.index(obj) Returns the lowest index in list that obj appears
list.insert(index, obj) Inserts object obj into list at offset index
list.pop(obj=list[-1]) Removes and returns last object or obj from list
list.remove(obj) Removes object obj from list
list.reverse() Reverses objects of list in place
list.sort([func]) Sorts objects of list, use compare func if given

LIST COMPREHENSION

Python has a method to construct and query lists called list comprehension.

```
>>> S = [x**2 for x in range(10)]
>>> V = [2**i for i in range(13)]
>>> M = [x for x in S if x % 2 == 0]
>>>
>>> print S; print V; print M
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81][1, 2, 4, 8, 16, 32, 64, 128,
256, 512, 1024, 2048, 4096][0, 4, 16, 36, 64]
```

ANOTHER EXAMPLE

```
>>> words = 'The quick brown fox jumps over the lazy dog'.split()
>>> print words['The', 'quick', 'brown', 'fox', 'jumps', 'over', 'the', 'lazy', 'dog']
>>>
>>> stuff = [[w.upper(), w.lower(), len(w)] for w in words]
>>> for i in stuff:
...     print i
...
['THE', 'the', 3]
['QUICK', 'quick', 5]
['BROWN', 'brown', 5]
['FOX', 'fox', 3]
['JUMPS', 'jumps', 5]
['OVER', 'over', 4]
['THE', 'the', 3]
['LAZY', 'lazy', 4]
['DOG', 'dog', 3]
```

SORTING AND SELECTING TOGETHER USING LIST COMPREHENSION

Sort using a technique called **list comprehension**.

```
highesttemps = [i for i in wxdata['temperature'] if i >= 0.4]
highesttemps.sort()
```

Note that this just sorts the one column.

SORTING LISTS AND DICTIONARIES

Want to sort a list by another, or a dictionary by a column? We have the sort command.

```
X = ["a", "b", "c", "d", "e", "f", "g", "h", "i"]

Y = [ 0, 1, 1, 0, 1, 2, 2, 0, 1]

yx = zip(Y, X)

yx [(0, 'a'), (1, 'b'), (1, 'c'), (0, 'd'), (1, 'e'), (2, 'f'), (2, 'g')
, (0, 'h'), (1, 'i')]

yx.sort()

yx [(0, 'a'), (0, 'd'), (0, 'h'), (1, 'b'), (1, 'c'), (1, 'e'), (1, 'i')
, (2, 'f'), (2, 'g')]

x_sorted = [x for y, x in yx]

x_sorted

['a', 'd', 'h', 'b', 'c', 'e', 'i', 'f', 'g']
```

OR A ONE LINER

```
[x for y, x in sorted(zip(Y, X))]
```

How can we sort dictionaries in the same way? One way is to do the same procedure, reassigning a new dictionary.

Try it!

FORMATTING OUTPUT

We showed that you can “add” strings together to combine them.

```
student = ’Sally’
print(‘Good morning ‘+student+’!’)
```

How do we get more control over this?

Answer: the `format` statement!

FORMAT METHOD

```
template.format(p0, p1, ..., k0=v0, k1=v1, ...)
```

The template (or format string) is a string which contains one or more format codes (fields to be replaced) embedded in constant text. The "fields to be replaced" are surrounded by curly braces {}.

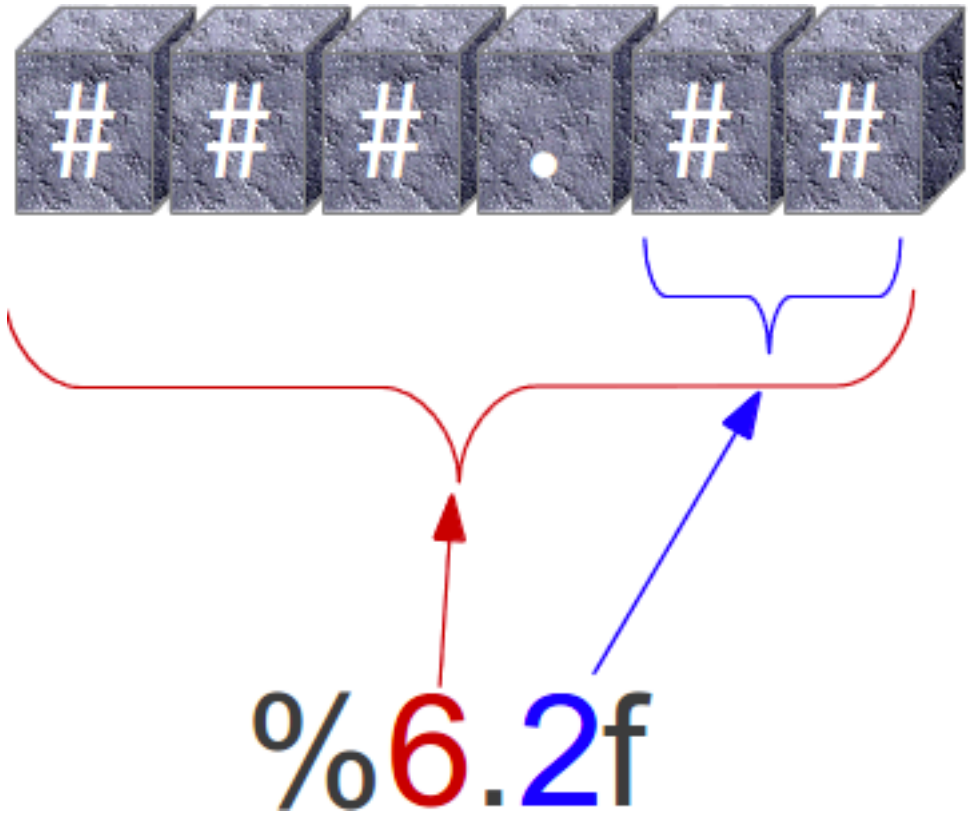
The curly braces and the "code" inside will be substituted with a formatted value from one of the arguments, according to the rules which we will specify soon.

Anything else, which is not contained in curly braces will be literally printed, i.e. without any changes.

FORMAT CODES

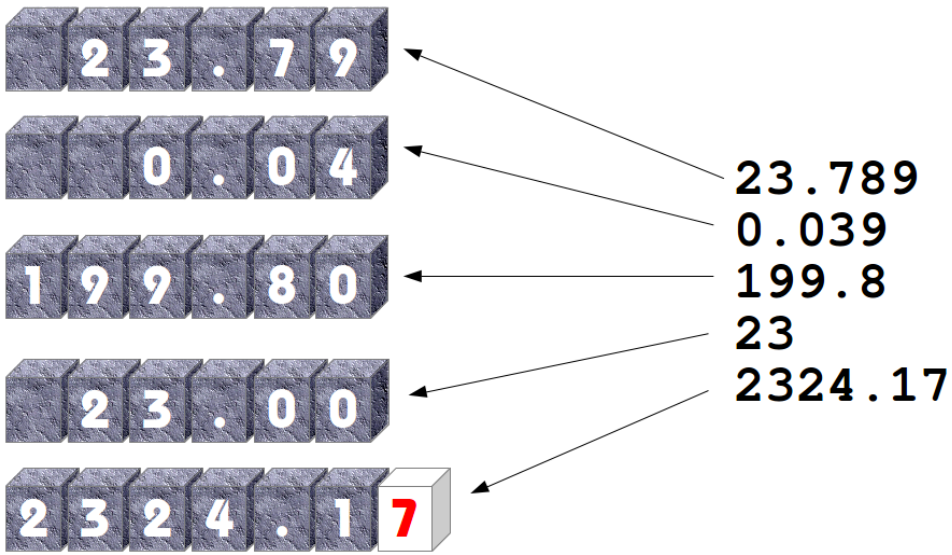
The general syntax for a format placeholder is

`%[flags][width][.precision]type`



Conversion	Meaning
d	Signed integer decimal.
i	Signed integer decimal.
e	Floating point exponential format (lowercase).
E	Floating point exponential format (uppercase).
f	Floating point decimal format.
F	Floating point decimal format.
g	Same as "e" if exponent is greater than -4 or less than precision, "f" otherwise.
G	Same as "E" if exponent is greater than -4 or less than precision, "F" otherwise.
c	Single character (accepts integer or single character string).
r	String (converts any python object using repr()).
s	String (converts any python object using str()).
%	No argument is converted, results in a "%" character in the result.

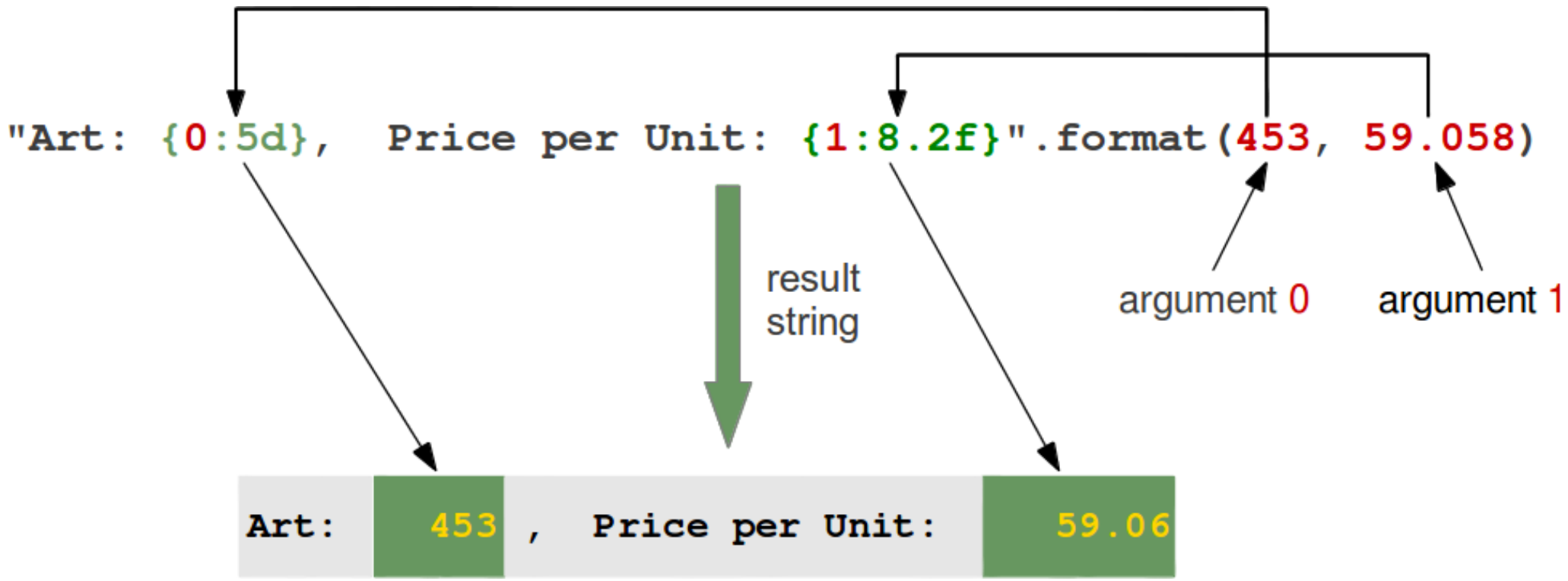
%6.2f with different values:



OTHER FORMATTING CODES

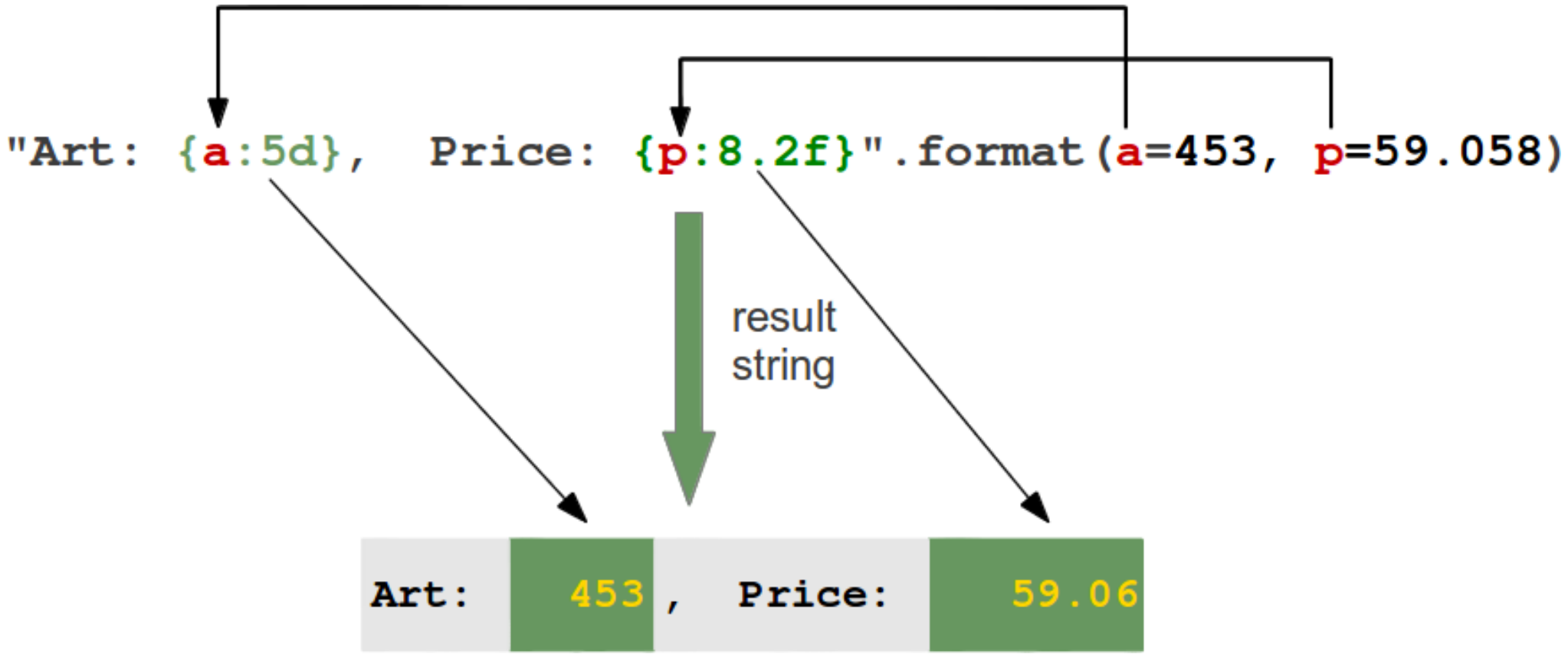
Flag	Meaning
#	Used with o, x or X specifiers the value is preceded with 0, 0o, 0O, 0x or 0X respectively.
0	The conversion result will be zero padded for numeric values.
-	The converted value is left adjusted
	If no sign (minus sign e.g.) is going to be written, a blank space is inserted before the value.
+	A sign character ("+" or "-") will precede the conversion (overrides a "space" flag).

“ILLUSTRATED” FORMAT EXAMPLE



```
>>> "First argument: {0}, second one: {1}".format(47,11) 'First argument: 47, second one: 11'

>>> "Second argument: {1}, first one: {0}".format(47,11) 'Second argument: 11, first one: 47'

>>> "Second argument: {1:3d}, first one: {0:7.2f}".format(47.42,11) 'Second argument: 11, first one: 47.42'

>>> "First argument: {}, second one: {}".format(47,11) 'First argument: 47, second one: 11'

>>> # arguments can be used more than once: ...

>>> "various precisions: {0:6.2f} or {0:6.3f}".format(1.4148) 'various precisions: 1.41 or 1.415'

>>>
```

WE CAN ALSO USE KEYWORD PARAMETERS IN FORMAT STATEMENTS



JUSTIFY

It's possible to left or right justify data with the format method. To this end, we can precede the formatting with a "<" (left justify) or ">" (right justify). We demonstrate this with the following examples:

```
>>> "{0:<20s} {1:6.2f}".format('Spam & Eggs:', 6.99)
'Spam & Eggs:            6.99
'>>> "{0:>20s} {1:6.2f}".format('Spam & Eggs:', 6.99)
'        Spam & Eggs:   6.99'
```

USING DICTIONARIES IN FORMAT

```
capital_country = {"United States" : "Washington",
                    "US" : "Washington",
                    "Canada" : "Ottawa",
                   "Germany": "Berlin",
                   "France" : "Paris",
                   "England" : "London",
                   "UK" : "London",
                   "Switzerland" : "Bern",
                   "Austria" : "Vienna",
                   "Netherlands" : "Amsterdam"}
print("Countries and their capitals:")
for c in capital_country:
    print("{country}: {capital}".format(country=c,
          capital=capital_country[c]))
```

WHAT'D WE LEARN? 🤓

- **sort** can sort lists for you
- **list comprehension** can help you loop, we used it to find elements in a list
- **format** helps you make your output look pretty, with the ultimate in German engineering