Effective Programming Practices for Economists

8. File input / output, strings, and data structures

Hans-Martin von Gaudecker

Department of Economics, Universität Bonn

universitätbonn

Review and completions

Built-in numeric types

- `int()`
  - Long integer, arbitrary precision
- `float()`
- `complex()`
  - Self-explanatory, otherwise similar to `float()`

Licensed under the Creative Commons Attribution License

Review and completions

Control flow

- `if condition:  / elif condition:  / else:`
- `for running_variable in iterable:`
- `while condition:`
  - Same syntax for `condition` as in `if`-statements
  - Stop when `condition` evaluates to `False` at the beginning of the loop.
  - Seldomly needed, `for`-loops usually cleaner

Licensed under the Creative Commons Attribution License

Review and completions

Modules and functions

- Namespaces, name resolution from inside out
- Do not see variables from other module unless imported

Licensed under the Creative Commons Attribution License

Plan for this lecture

- File input / output
- Strings
- Container types
  - Lists and tuples
  - Sets, frozensets
  - Dictionaries
- Important concepts introduced along the way
  - Special characters and escape sequences
  - Indexing and slicing
  - Methods
  - Aliasing

Licensed under the Creative Commons Attribution License

Working with files

- Usually, data is not stored in Python scripts . . .
- Use the built-in function `open` to open a file

```
input_file = open('exercise_1.tex', 'r')
```

- First argument is path, second is “`r`” (read) or “`w`” (write)
- Returns a `file` **object** with **methods** for input or output

Licensed under the Creative Commons Attribution License

Objects and Methods

- Anything defined in a namespace is an object
  - Numbers, functions, strings, . . .
- A method is a function that is tied to a particular object
- Almost everything in Python has methods
  - Numbers are the only quasi-exception
- To call a method `meth` of object `obj`, type `obj.meth()`

```
>>> input_file.readline()
'\\documentclass[11pt,a4paper,leqno]{article}\n'
```

- In contrast to functions, methods do not see objects in the enclosing scope, but only objects that are passed in

Licensed under the Creative Commons Attribution License

File methods

Method	Purpose	Example
read	Read N bytes from the file, returning the empty string if the file is empty. If N is not given, read the rest of the file.	`next_block = input_file.read(1024)`
readline	Read the next line of text from the file, returning the empty string if the file is empty.	`line = input_file.readline()`
readlines	Return the remaining lines in the file as a list, or an empty list at the end of the file.	`rest = input_file.readlines()`
write	Write a string to a file (without appending a newline).	`output_file.write( "gamma = {}".format(u_curv))`
writelines	Write each string in a list to a file (without appending newlines).	`output_file.writelines( ["gamma", " = ", str(u_curv)])`
close	Close the file; no more reading or writing is allowed.	`input_file.close()`

Licensed under the Creative Commons Attribution License

Class exercise: Reading in a file

Using the methods on the previous screen, write a Python script that prints the entire content of a text file of your choice to the screen. Make sure the file handle is closed at the end.

Licensed under the Creative Commons Attribution License

The `with`-Statement

- The `with` statement executes a block of code and then cleanly closes the file.

```
with open('exercise_1.tex', 'r') as input_file:
    print(input_file.read())
```

```
input_file = open('exercise_1.tex', 'r')
print(input_file.read())
input_file.close()
```

Licensed under the Creative Commons Attribution License

How to read the contents of a file?

- Choice of `read`, `readlines`, a while-loop combined with `readline`, **or** using a for-loop over the file object:

```
with open('exercise_1.tex', 'r') as input_file:
    for line in input_file:
        print(line, end=' ')
```

- Whatever is easiest to read in a particular context!

- Looping over the file handle useful for large files, because it only stores one line at a time in memory, as opposed to the entire content

Licensed under the Creative Commons Attribution License

A closer look at strings

- Strings are a sequence type, i.e. an **ordered** collection
  - A collection of Unicode characters, to be precise
- Can be indexed (append index in square brackets)
- Indices start at 0

```
>>> 'crra_parameter'[0]
'c'
```

- Negative indices start counting from the end of the string

```
>>> 'crra_parameter'[-1]
'r'
```

Licensed under the Creative Commons Attribution License

A closer look at strings

- Strings are **immutable**
- Cannot be modified once created

```
>>> gamma = 'crra_parameter'
>>> gamma[1] = 'a'
Traceback (most recent call last):
  File "<console>", line 1, in <module>
TypeError: 'str' object does not support item assignment
```

- But variables can be modified, of course

```
>>> gamma = 'cara_parameter'
>>> print(gamma[1])
a
```

Licensed under the Creative Commons Attribution License

Extracting substrings

- `text[start:end]` takes a slice out of text
  - Creates a new string containing the characters of `text` from `start` up to (but not including) `end`

```
>>> print('crra_parameter'[0:4])
```

```
crra
```

- Not specifying a number before (after) the colon in a slice takes everything to (from) the beginning (end) of the string
  - `text[:]` returns a copy of `text`

```
>>> print('crra_parameter'[-9:])
```

```
parameter
```

Licensed under the Creative Commons Attribution License

Bounds checking

- Trying to index a string element outside the range raises an `IndexError`

```
>>> 'crra_parameter'[22]
Traceback (most recent call last):
  File "<console>", line 1, in <module>
IndexError: string index out of range
```

- But slicing just takes everything to the end (from the beginning) of the string

```
>>> 'crra_parameter'[-9:22]
'parameter'
```

- A foolish consistency . . .

Licensed under the Creative Commons Attribution License

Consequences

- `text[1:2]` is either:
  - The second character in `text` ...
  - ... or the empty string
- So `text[2:1]` is always the empty string
- So is `text[1:1]`
  - From index 1, up to but not including index 1
- `text[1:-1]` is everything except the first and last characters
  - Which may again be the empty string

Licensed under the Creative Commons Attribution License

Special characters

- End-of-line:

  `\n` Unix, MacOS X

  `\r\n` Windows

  `\r` MacOS 9

- Tabs:

  `\t`

- Single and double quotes

Licensed under the Creative Commons Attribution License

Special characters

- So you need a way to print these characters
- Prepend another (a) backslash, e.g., `\\n` or `\'`
  - “Escape sequences”
- Quotes: Just use triple quotes or the respective other kind (single, double) to delimit the string
- Other: Prepend ‘`r`’ (for ‘raw’) to the string – special meaning of backslashes is ignored

Licensed under the Creative Commons Attribution License

Special characters

- Thus . . .

```
>>> print('crra_parameter\ncara_parameter')
crra_parameter
cara_parameter

>>> print('crra_parameter\\ncara_parameter')
crra_parameter\ncara_parameter

>>> print(r'crra_parameter\ncara_parameter')
crra_parameter\ncara_parameter
```

Licensed under the Creative Commons Attribution License

Comparison and search methods

- `str.startswith(prefix[, start[, end]])`
  `str.endswith(suffix[, start[, end]])`
  - Return `True` if the string starts (ends) with the specified prefix (suffix), otherwise return `False`. [...] With optional `start`, test beginning at that position. With optional `end`, stop comparing at that position

```
>>> gamma = 'crra_parameter'
>>> gamma.startswith('crra')
True
>>> gamma[:4] == 'crra'
True
>>> gamma.startswith('par', 5)
True
>>> gamma.endswith('ter', 0, 4)
False
```

Licensed under the Creative Commons Attribution License

Comparison and search methods

- `str.find(sub[, start[, end]])`
  - Return the lowest index in the string where substring `sub` is found, such that sub is contained in the slice `s[start:end]`. Optional arguments `start` and `end` are interpreted as in slice notation. Return `-1` if `sub` is not found
- `str.count(sub[, start[, end]])`
  - Return the number of non-overlapping occurrences of substring `sub` in the range `[start, end]`. Optional arguments `start` and `end` are interpreted as in slice notation

Licensed under the Creative Commons Attribution License

Methods for string manipulations

- `str.replace(old, new[, count])`
  - Return a copy of the string with all occurrences of substring old replaced by new. If the optional argument `count` is given, only the first `count` occurrences are replaced

```
>>> gamma = 'crra_parameter'
>>> print(gamma.replace('r', 'a', 1))
cara_parameter
>>> print(gamma)
crra_parameter
```

Licensed under the Creative Commons Attribution License

Methods for string manipulations

- `str.upper()` / `str.lower()`
  - Return a copy of the string converted to uppercase (lowercase)

```
>>> 'crra_parameter'.upper()
'CRRA_PARAMETER'
```

Licensed under the Creative Commons Attribution License

Methods for string manipulations

- `str.strip([chars])`
  - Return a copy of the string with the leading and trailing characters removed. The `chars` argument is a string specifying the set of characters to be removed. If omitted or `None`, the `chars` argument defaults to removing whitespace. The `chars` argument is not a prefix or suffix; rather, all combinations of its values are stripped

```
>>> '   spacious   '.strip()
'spacious'
>>> 'www.example.com'.strip('cmowz.')
'example'
```

Licensed under the Creative Commons Attribution License

Methods for string manipulations

- `str.lstrip([chars])` / `str.rstrip([chars])`
  - Same as `str.strip([chars])`, but only affects leading (trailing) characters

```
>>> '   spacious   '.lstrip()
'spacious   '
>>> '   spacious   '.rstrip()
'   spacious'
```

Licensed under the Creative Commons Attribution License

Justifying text in a string of given length

- `str.ljust(width[, fillchar])`
  `str.rjust(width[, fillchar])`
  `str.center(width[, fillchar])`
  - Return the string left justified (right justified, centered) in a string of length `width`. Padding is done using the specified `fillchar` (default is a space). The original string is returned if `width` is less than len(s)

```
>>> for parameter in '\( \gamma \)', '\( \lambda \)':
...     print(parameter.ljust(14) + '&' + '1.0'.rjust(5) + r' \\')
...
\( \gamma \)   &  1.0 \\
\( \lambda \)  &  1.0 \\
```

Licensed under the Creative Commons Attribution License

The format method: Substitutions

- `str.format(*args, *kwargs)`
  - Perform a string formatting operation. The string on which this method is called can contain literal text or replacement fields delimited by braces {}. Each replacement field contains either the numeric index of a positional argument (could be left out), or the name of a keyword argument. Returns a copy of the string where each replacement field is replaced with the string value of the corresponding argument

Licensed under the Creative Commons Attribution License

The format method: Presentation types

- Format specifiers come after positional/keyword argument
- E.g. field width:

```
>>> "The sum of {} + {} is {:4}".format(4, 5, 4+5)
'The sum of 4 + 5 is    9'
>>> "The sum of {0} + {1} is {2:4}".format(4, 5, 4+5)
'The sum of 4 + 5 is    9'
>>> "The sum of {n} + {n} is {sum:2}".format(n=1, sum=1+1)
'The sum of 1 + 1 is  2'
```

- Or numeric formatters:

```
>>> for parameter in 'gamma', 'lambda':
...     print(r"\( \{:7} \) & {:3.1f} \\".format(parameter, 1.0))
...
\( \gamma   \) & 1.0 \\
\( \lambda  \) & 1.0 \\
```

Licensed under the Creative Commons Attribution License

The format method: Presentation types

- Python will guess the format you want to write numbers in
- But you can take pretty tight control over it if you want
  - Very useful for tables
  - Similar stuff in all languages
- Some important presentation types:

Specifier	Variable Type	Description
s	str	Default, can be left out.
g	numeric	General number – let Python handle part of the formatting.
d	int	Base-10 decimal.
f	float	Fixed-point. Specify precision by, e.g. `.2f`

Licensed under the Creative Commons Attribution License

String (and file) encodings

- 20^{th} century:
  - Little storage space & few characters needed in AE
  - $\Rightarrow$ ASCII codec defines only 128 characters
- Country-specific encodings do what they want with characters 129-255
- Clash with WWW . . . Unicode (UTF-8) emerges as the standard
- Fairly advanced topic, read
  `http://www.joelonsoftware.com/articles/Unicode.html`
  beforehand if you're interested in it or need it.

Licensed under the Creative Commons Attribution License

Maurice Allais, Econometrica 1953

1. LA PRÉSENTE étude est essentiellement destinée à un exposé critique des postulats et axiomes des théories du risque de l'école américaine.

Pour procéder à cet exposé critique, le mieux nous paraît de diviser notre exposé en deux parties; dans la première nous essaierons de faire comprendre quelle est notre propre conception, dans la seconde nous procèderons, compte tenu des indications données dans la première partie, à une analyse critique du principe de Bernoulli et tout particulièrement des différents axiomes de l'école américaine.

D'une manière générale nous essaierons de faire appel à l'intuition et d'éviter dans toute la mesure du possible un formalisme mathématique trop abstrait, qui en réalité n'a que trop souvent pour effet de détourner l'attention des véritables difficultés et de masquer des aspects essentiels. Les mathématiques ne sont qu'un moyen de transformation; seule compte en fait la discussion des prémisses et des résultats.[3]

Licensed under the Creative Commons Attribution License

String (and file) encodings

```
>>> with open('allais_1953_section_1.txt') as a:
...     print(a.read())
...


Traceback (most recent call last):
  File "<stdin>", line 5, in <module>
  File "/usr/local/Cellar/python3/3.2.3/Frameworks/Python.framework/Versions
    (result, consumed) = self._buffer_decode(data, self.errors, final)
UnicodeDecodeError: 'utf-8' codec can't decode byte 0xc9 in position 5: inva
```

Licensed under the Creative Commons Attribution License

String (and file) encodings

```
>>> with open('allais_1953_section_1.txt', encoding='windows-1252') as a:
...     print(a.read())
...
```

```
LA PRÉSENTE etude est essentiellement destinée à un exposé critique
des postulats et axiomes des théories du risque de l'école américaine.
    Pour procéder à cet exposé critique, le mieux nous paraît de diviser
notre exposé en deux parties; dans la première nous essaierons de faire
comprendre quelle est notre propre conception, dans la seconde nous
procèderons, compte tenu des indications données dans la première
partie, à une analyse critique du principe de Bernoulli et tout particu-
lièrement des differents axiomes de l'école américaine.
```

Licensed under the Creative Commons Attribution License

More on sequences

- Sequences are ordered collections of items
  - Ordered bags, in mathematical terms
  - In case of a string, items consist of characters
- Iterating over the items of a sequence does so in the order in which they appear

```
>>> for x in 'ab':
>>>     print(x)
...
a
b
```

Licensed under the Creative Commons Attribution License

Introducing lists

- The workhorse among sequences
- Definition: **Mutable** sequence **of objects**.
  - $\Rightarrow$ Items can consist of anything
- To define it, enclose items in square brackets and separate them with commas

```
>>> example = ['text', 1, ['nested', 'items']]
>>> for item in example:
...     print(item)
...
text
1
['nested', 'items']
```

Licensed under the Creative Commons Attribution License

Defining lists and changing items

- Indexing and slicing work just as with strings
- Mutable means that the list can be changed in-place

```
>>> crra_parameters = [-0.5, 0.0, 0.8]
>>> crra_parameters[2] = 0.5
>>> print(crra_parameters)
[-0.5, 0.0, 0.5]
```

- The index must exist for this to work

```
>>> crra_parameters[5] = 0.99
Traceback (most recent call last):
  File "<console>", line 1, in <module>
IndexError: list assignment index out of range
```

Licensed under the Creative Commons Attribution License

Defining lists and changing items

- You can also assign values to slices

```
>>> crra_parameters[1:] = [-0.4, -0.3]
>>> print(crra_parameters)
[-0.5, -0.4, -0.3]
```

- If slice is out of bounds, the items are appended to the list

```
>>> crra_parameters[8:10] = [-0.2, -0.1]
>>> print(crra_parameters)
[-0.5, -0.4, -0.3, -0.2, -0.1]
```

- But this is not good style
  - Explicit is better than implicit

Licensed under the Creative Commons Attribution License

Adding and deleting list elements

- Use the `extend()` method to add any iterable . . .

```
>>> example = ['a']
>>> example.extend('bc')
>>> print(example)
['a', 'b', 'c']
```

- . . . or join (concatenate) two lists with + operator

```
>>> new_example = example + ['d']
>>> print(new_example)
['a', 'b', 'c', 'd']
```

- The following lines have the same effect:

```
>>> example = example + new_example
>>> example += new_example
>>> example.extend(new_example)
```

Licensed under the Creative Commons Attribution License

Adding and deleting list elements

- Use `append()` method to add an element . . .

```
>>> crra_parameters.append(0.0)
>>> print(crra_parameters)
[-0.5, -0.4, -0.3, -0.2, -0.1, 0.0]
```

- . . . and `remove(x)` method to delete first occurrence of `x`
- Finally, `del example[i]` removes the `i`'th element
- This also works with slices

```
>>> del example[1:3]
>>> print(example)
['a']
```

Licensed under the Creative Commons Attribution License

More list methods

Operation	Result
`s.count(x)`	Return number of `i`'s for which `s[i] == x`
`s.index(x)`	Return smallest k such that `s[k] == x`
`s.insert(i, x)`	Insert `x` into `s` before index `i`
`s.pop(i)`	Return element `i` and delete it from `s`
`s.reverse()`	Reverse the items of `s` in place
`s.sort()`	Sort the items of `s` in place

Licensed under the Creative Commons Attribution License

Notes on list methods

- Some have optional arguments
- Where relevant, `s` is modified **in place**
- Very common mistake:

```
>>> example = ['c', 'a', 'b']

>>> ouch = example.sort()

>>> print(example)
['a', 'b', 'c']

>>> print(ouch)
None
```

Licensed under the Creative Commons Attribution License

String vs. list methods

- The big difference to keep in mind when using string or list methods is mutability
- String methods like `replace()` return a **copy** of the original string
- Lists are modified in place, wherever possible
- Different implementations serve different purposes . . .

Licensed under the Creative Commons Attribution License

Aliasing

a·li·as

adverb

> *Used to indicate that a named person is also known or more familiar under another specified name: Eric Blair, alias George Orwell.*

Here, aliasing refers to the situation where the same memory location can be accessed using different names

Licensed under the Creative Commons Attribution License

Aliasing

- In Python, the value of a name is a reference to an object in memory
  - Changes to that object affect all names referencing it
  - Not creating copies with each assignment leads to dramatic decreases in memory footprint and increases in efficiency
- But you need to be aware of the consequences

```
>>> utility_curvature = [0.0, 0.5]
>>> loss_aversion = utility_curvature
>>> loss_aversion[0:2] = [1.5, 2.0]
>>> print(utility_curvature)
[1.5, 2.0]
```

Licensed under the Creative Commons Attribution License

Aliasing
Example: Creating a 2×2 grid of ones

```
N = 2

grid = []

for x in range(N):
    grid.append([])
    for y in range(N):
        grid[-1].append(1)

print(grid)
```

Licensed under the Creative Commons Attribution License

Aliasing
Example: Creating a 2×2 grid of ones

```
N = 2
EMPTY = []
grid = []

for x in range(N):
    grid.append(EMPTY)
    for y in range(N):
        grid[-1].append(1)

print(grid)
```

Licensed under the Creative Commons Attribution License

Aliasing
Example: Creating a 2×2 grid of ones

```
N = 2
EMPTY = []
grid = []

for x in range(N):
    grid.append(EMPTY)
    for y in range(N):
        EMPTY.append(1)

print(grid)
```

Licensed under the Creative Commons Attribution License

Aliasing
Example: Passing objects to functions

```
def bad_idea(in_object):
    out_object = in_object
    for i, item in enumerate(in_object):
        out_object[i] = item / 2.0
    return out_object

x = [3, 4]
y = bad_idea(x)
print(x, y)
```

Licensed under the Creative Commons Attribution License

Aliasing
Example: Passing objects to functions

```
def good_idea(in_object):
    out_object = []
    for item in in_object:
        out_object.append(item / 2.0)
    return out_object

x = [3, 4]
y = good_idea(x)
print(x, y)
```

Licensed under the Creative Commons Attribution License

Aliasing
Example: Passing objects to functions

- Message: **Never modify the inputs to a function!**
- And if you have to, beware of subtleties
- The function above is said to have **side effects**
  - Introduces state dependence – the value of `x` **implicitly** depends on the number of times `bad_idea(x)` has run
  - Aside: Same issue with modifiying global variables

Licensed under the Creative Commons Attribution License

List indexing vs. list slicing

- Indexing and slicing return different both return references to the list element(s)
- Changes to a slice also affect the original list

Licensed under the Creative Commons Attribution License

Methods for string $\leftrightarrow$ list conversion

- `new_list = original_string.split([split_characters])`
  - `new_list` is a list of the words in `original_characters`, using `split_characters` as the delimiter string. If `split_string` is not specified, it defaults to **consecutive** whitespace
- `new_string = [join_characters].join(iterable)`
  - `new_string` is the concatenation of the strings in `iterable`, separated by `join_characters` (could be the empty string)
- Very common operations

Licensed under the Creative Commons Attribution License

Methods for string $\leftrightarrow$ list conversion

```
>>> a = 'too                far\t\tapart.'
>>> b = a.split()
>>> c = 'not ' + ' '.join(b)
>>> print('a: {}\nb: {}\nc: {}'.format(a, b, c))

a: too                far                apart.
b: ['too', 'far', 'apart.']
c: not too far apart
```

Licensed under the Creative Commons Attribution License

Tuples

- Same as a list, but immutable
- Define using parentheses instead of square brackets: `(1, 2, 3)` instead of `[1, 2, 3]`
- Define an empty tuple: `empty_tuple = ()`
- Define a one-element tuple including a comma: `(55,)`
  - Because `(55)` just has to be the integer 55
- If you put a list in a tuple, it remains mutable . . .

```
>>> test = (1, [2, 3])
>>> test[1][1] = 4
>>> print(test)
(1, [2, 4])
```

Licensed under the Creative Commons Attribution License

Sets and frozensets

- **Def:** Unordered collections of distinct hashable objects
- Difference set / frozenset: Mutability
- Python will take care that any value appears at most once
- Hashable $\approx$ immutable
  - Consequently, you can't put a list in a set
- Example: Track variety of products bought by a household

Licensed under the Creative Commons Attribution License

Sets and frozensets

- Construct using:

```
{[comma-separated elements]}

set(iterable)

frozenset(iterable)
```

```
>>> a = {'Beer', 'Frozen Pizza', 'Beer'}
>>> b = set(('Beer', 'Frozen Pizza', 'Beer'))
>>> print(a, '\n', b)
set(['Beer', 'Frozen Pizza'])

set(['Beer', 'Frozen Pizza'])
```

- Careful: `{}` initialises an empty dictionary, not a set
  - Use `set()` for that purpose

Licensed under the Creative Commons Attribution License

Sets and frozensets

- Python sets follow the concept as a set in mathematics
- Reflected in the methods of sets and frozensets

```
>>> a = {'Beer', 'Milk'}
>>> b = {'Milk'}
>>> b.add('Frozen Pizza')
>>> c = a.intersection(b)
>>> d = b.union(a)
>>> print(c, '\n', d)

set(['Milk'])

set(['Beer', 'Milk', 'Frozen Pizza'])
```

Licensed under the Creative Commons Attribution License

Sets and frozensets

- For a complete list of methods, see `http://docs.python.org/3/library/stdtypes.html#set-types-set-frozenset`
- Remember a set can only store immutable objects
  - Use tuples for storing multi-part, ordered values
  - Use frozensets for storing multi-part, unordered values

Licensed under the Creative Commons Attribution License

Lookup speed

- Problem: Track variety of products a household bought over a period of several weeks

- List implementation:

```
all_products_bought = []
for product in products_bought_week_1:
    if product not in all_products_bought:
        all_products_bought.append(product)
```

- Requires $\frac{N(N+1)}{4}$ comparisons, which is $O(N^2)$

Licensed under the Creative Commons Attribution License

Lookup speed

- Set implementation:

```
all_products_bought = set()
for product in products_bought_week_1:
    all_products_bought.add(product)
```

- $O(N)$: Dramatic difference!

- Need to look up elements by value rather than location

  $\Rightarrow$ Elements must be immutable

- Note: In case of tuples, the indices are immutable, so in contrast to sets it can hold mutable objects

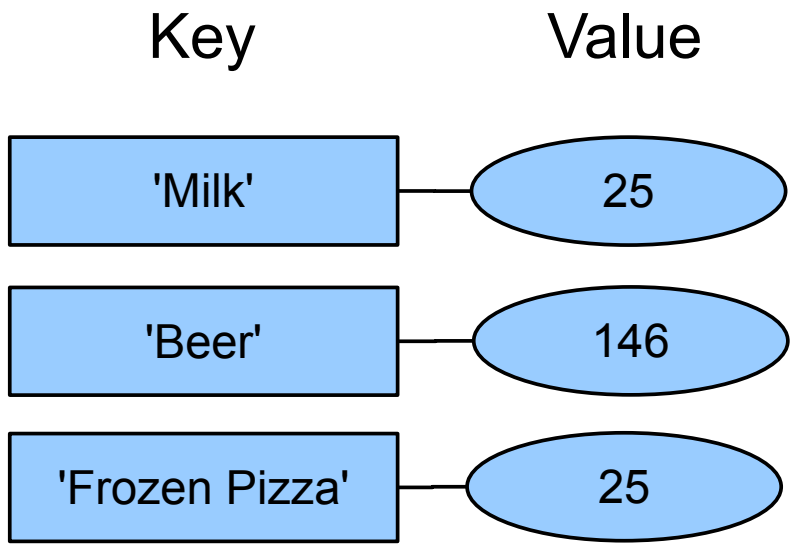
Licensed under the Creative Commons Attribution License

Dictionaries

- Keep track of the amounts people bought of each product
- Back to lists of lists?

```
>>> [["Beer", 146], ["Frozen Pizza", 25] ["Milk", 25]]
```

- Better solution: With each element of a set (‘key’), store extra data (‘value’)



Licensed under the Creative Commons Attribution License

Dictionaries
Creating

- Put comma-separated `key: value` pairs inside `{}` :

```
>>> all_products_bought = {'Milk': 25, 'Beer': 146}
>>> empty_dict = {}
```

- Use the `dict()` constructor – various syntax possibilities

```
>>> all_products_bought = dict(Milk=25, Beer=146)

>>> all_products_bought = dict(zip(
...     ['Milk', 'Beer'],
...     [25, 146]
... ))
>>> empty_dict = dict()
```

- Full list here: `http://docs.python.org/3/library/stdtypes.html#mapping-types-dict`

Licensed under the Creative Commons Attribution License

Dictionaries
Indexing

- Look up the value associated with a key using indexing

```
>>> print(all_products_bought['Beer'])
146
```

- Can only access keys that are present

```
>>> print(all_products_bought['Frozen Pizza'])
Traceback (most recent call last):
  File "<console>", line 1, in <module>
KeyError: 'Frozen Pizza'
```

Licensed under the Creative Commons Attribution License

Dictionaries
Updating

- Assigning to a dictionary key . . .

. . . creates a new entry if the key is not in the dictionary

```
>>> all_products_bought['Frozen Pizza'] = 25
>>> print(all_products_bought)

{'Beer': 25, 'Frozen Pizza': 25, 'Milk': 25}
```

. . . overwrites the previous value if the key is already present

```
>>> all_products_bought['Beer'] = 1
>>> print(all_products_bought)

{'Beer': 1, 'Frozen Pizza': 25, 'Milk': 25}
```

Licensed under the Creative Commons Attribution License

Dictionaries
Selected methods and operations

Method / Operation	Result
`len(d)`	Return the number of items in the dictionary `d`.
`del d[key]`	Remove `d[key]` from `d`. Raises a `KeyError` if `key` is not in the map (set of keys).
`d.clear()`	Remove all items from the dictionary `d`.
`d.get(key[, default])`	Return the value for `key` if `key` is in the dictionary, else `default`. If `default` is not given, it defaults to `None`.
`d.keys()`	Return a view of `d`'s keys.
`d.values()`	Return a view of `d`'s values
`d.items()`	Return a view of `d`'s `(key, value)` tuples.

Licensed under the Creative Commons Attribution License

Dictionaries in action

- Extremely useful invention
  - You'll quickly miss them in Stata
  - Matlab: Similar functionality via `containers.Map` class
- Back to our utility functions
- How can we sensibly employ what we have seen?

Licensed under the Creative Commons Attribution License

Dictionaries in action

- Sensibly?
  - Make code more readable
  - Make code run faster
  - Make code more general (but only if there are real benefits)
- Gambles are natural candidates for a dictionary

```
{
    outcome_1: probability_1,
    outcome_2: probability_2
}
```

  - If outcomes are not unique for some reason, just sum up the probabilities
  - Allowing for more than two lottery outcomes is a free lunch

Licensed under the Creative Commons Attribution License

Dictionaries in action

```
"""Compute the utility value of a lottery under prospect theory
(neglecting probability weighting).

"""

from utility_functions import v_prospect_theory


# Define the characteristics of the individual.
power_coefficient = 0.5
loss_aversion_coefficient = 1.0

# Define the characteristics of the lottery.
lottery = {-500.0: 0.4, 550.0: 0.6}

# Compute the expected utility value.
utility_evaluation = v_prospect_theory(
    lottery,
    gamma=power_coefficient,
    lambda_=loss_aversion_coefficient
)

print('Utility evaluation:', utility_evaluation)
```

Licensed under the Creative Commons Attribution License

```
"""Provide a collection of utility functions."""


def u_prospect_theory(z, gamma, lambda_):
    """Return the utility evaluation of a monetary quantity z under power
    utility with utility curvature parameter gamma and loss aversion
    parameter lambda_.

    """

    if z >= 0:
        u = z ** (1 - gamma)
    elif z < 0:
        if gamma <= 0:
            u = -lambda_ * (-z) ** (1 + gamma)
        elif gamma > 0:
            u = -lambda_ * (-z) ** (1 - gamma)
    return u


def v_prospect_theory(lotteries, gamma, lambda_=1):
    """Return the expected utility value of a gambles described by
    {outcome: probability}-pairs *lotteries* under power utility with utility
    curvature parameter *gamma* and loss aversion parameter *lambda_*.

    """

    v = 0.0
    for outcome, probability in lotteries.items():
        # Note: += does in-place addition, i.e. v += ...  <=>  v = v + ...
        v += probability * u_prospect_theory(outcome, gamma, lambda_)

    return v
```

Functions are objects

- Create an alias for a function by simple assignment

```
>>> def f(x):
...     return x ** 2

>>> g = f

>>> print(f(2), g(3))

4 9
```

Licensed under the Creative Commons Attribution License

User-defined objects

- Think of functions in another module as methods of an object (which happens to be a module)
  - Will meet `numpy.diag([1, 3])`
- Users can define their own custom objects
  - Create a class – similar meaning as in mathematics
  - And then objects as instances of that class
  - Can have all kinds of attributes, methods, . . .
  - Another level of abstraction

Licensed under the Creative Commons Attribution License

User-defined objects

- For now, you need to understand the concept because all extensions we will meet provide custom objects
  - E.g. the equivalent to a Matlab array
  - Resist the temptation to use a list as an array!

Licensed under the Creative Commons Attribution License

Sorting

- Use the `sort()` method of lists only for specific cases
- Else there is the `sorted()` function
- Returns a sorted list of the items in its argument
  - Obvious if the iterable is a string, list, set, etc.
  - A sorted list of the keys if the iterable is a dictionary
- Optional arguments allow for descending sorts or customised sorting of complex objects

Licensed under the Creative Commons Attribution License

Review of concepts

- Files and encodings
- Working with strings, methods
- Another important abstraction: Container types
  - Lists, tuples
  - Sets, frozensets
  - Mapping types: Dictionaries
- Aliasing, mutability

Licensed under the Creative Commons Attribution License

Acknowledgements

- This course is designed after and borrows a lot from the `Software Carpentry` course designed by `Greg Wilson` for scientists and engineers.

- The Software Carpentry course material is made available under a `Creative Commons Attribution License`, as is this course's material.

Licensed under the Creative Commons Attribution License

License for the course material

[Links to the full `legal` text and the source text for `this` page.] You are free:

- **to Share** to copy, distribute and transmit the work
- **to Remix** to adapt the work

Under the following conditions:

- **Attribution** You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).

With the understanding that:

- **Waiver** Any of the above conditions can be waived if you get permission from the copyright holder.
- **Public Domain** Where the work or any of its elements is in the public domain under applicable law, that status is in no way affected by the license.
- **Other Rights** In no way are any of the following rights affected by the license:
  - Your fair dealing or fair use rights, or other applicable copyright exceptions and limitations;
  - The author's moral rights;
  - Rights other persons may have either in the work itself or in how the work is used, such as publicity or privacy rights.

Notice For any reuse or distribution, you must make clear to others the license terms of this work. The best way to do this is with a link to `this` web page.