Working with text

- file formats
- CSV, JSON, XML, Excel
- regular expressions
- module re, finditer

Some file formats

File extension	Content
.html	HyperText Markup Language
.mp3	Audio File
.png .jpeg .jpg	Image files
.svg	Scalable Vector Graphics file
.json	JavaScript Object Notation
.csv	Comma separated values
.xml	eXtensible Markup Language
.xlmx	Micosoft Excel 2010/2007 Workbook

File extension	Description
.exe	Windows executable file
.app	Max OS X Application
.py	Python program
.pyc	Python compiled file
.java	Java program
.cpp	C++ program
.c	C program
.txt	Raw text file

PIL – the Python Imaging Library

- pip install Pillow

rotate_image.py

```
from PIL import Image
img = Image.open("Python-Logo.png")
img_out = img.rotate(45, expand=True)
img_out.save("Python-rotated.png")
```

- For many file types there exist Python packages handling such files, e.g. for images Pillow supports 40+ different file formats

Python-Logo.png

Python-rotated.png

python-pillow.org

CSV files - Comma Separated Values

- Simple 2D tables are stored as rows in af file, with values separated by comma
- Strings stored are quoted if necessary
- Values read are strings
- The deliminator (default comma) can be changed by keyword argument `delimiter`. Other typical deliminators are tabs `'\t'`, and semicolon `';'`

docs.python.org/3/library/csv.html

csv-example.py

```
import csv
FILE = 'csv-data.csv'
data = [[1, 2, 3],
        ['a', '"b"'],
        [1.0, ['x',"y"], 'd']]
with open(FILE, 'w', newline="") as outfile:
    csv_out = csv.writer(outfile)
    for row in data:
        csv_out.writerow(row)
with open(FILE, 'r', newline="") as infile:
    for row in csv.reader(infile):
        print(row)
```

Python shell

```
| ['1', '2', '3']
  ['a', '"b"']
  ['1.0', "['x', 'y']", 'd']
```

csv-data.csv

```
1,2,3
a,"""b"""
1.0,"['x', 'y']",d
```

CSV files - Tab Separated Values

csv-tab-separated.py

```python
import csv

FILE = 'tab-separated.csv'

with open(FILE) as infile:
    for row in csv.reader(infile, delimiter='\t'):
        print(row)
```

Python shell

```
['1', '2', '3']
['4', '5', '6']
['7', '8', '9']
```

tab-separated.csv

```
1	2	3
4	5	6
7	8	9
```

CSV files - Quoting

- The amount of quoting is controlled with keyword argument `quoting`
- `csv.QUOTE_MINIMAL` etc. can be used to select the quoting level
- Depending on choice of quoting, numeric values and strings cannot be distinguished in CSV file (`csv.reader` will read all as strings anyway)

csv-quoting.py

```
import csv
import sys

data = [[1, 1.0, '1.0'], ['abc', '"', '\t"']]

quoting_options = [(csv.QUOTE_MINIMAL, "QUOTE_MINIMAL"),
                   (csv.QUOTE_ALL, "QUOTE_ALL"),
                   (csv.QUOTE_NONNUMERIC, "QUOTE_NONNUMERIC"),
                   (csv.QUOTE_NONE, "QUOTE_NONE")]

for quoting, name in quoting_options:
    print(name)
    csv_out = csv.writer(sys.stdout, quoting=quoting, escapechar='\\')
    for row in data:
        csv_out.writerow(row)
```

Python shell

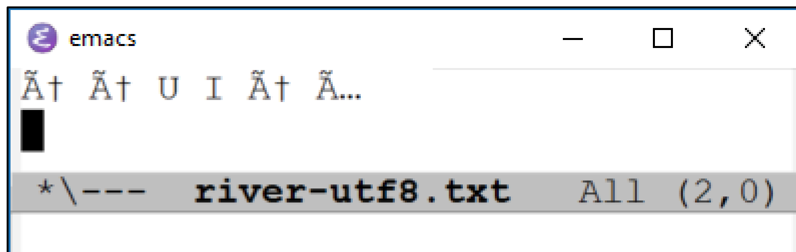
```
| QUOTE_MINIMAL  # cannot distinguish 1.0 and "1.0"
| 1,1.0,1.0
| abc,"""","       """
| QUOTE_ALL  # cannot distinguish 1.0 and "1.0"
| "1","1.0","1.0"
| "abc","""","     """
| QUOTE_NONNUMERIC
| 1,1.0,"1.0"
| "abc","""","     """
| QUOTE_NONE  # cannot distinguish 1.0 and "1.0"
| 1,1.0,1.0
| abc,\",        \"
```

```
river-utf8.py   (size 17 bytes, encoding UTF-8)
Æ Æ U I Æ Å
```

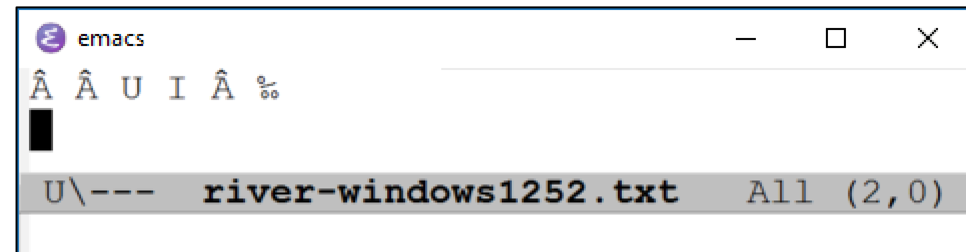
```
river-windows1252.py   (size 13 bytes, encoding Windows-1252)
Æ Æ U I Æ Å
```

File encodings...

- Text files can be *encoded* using many different encodings (UTF-8, UTF-16, UTF-32, Windows-1252, ANSI, ASCII, ISO-8859-1, ...)
- Different encodings can result in different file sizes, in particular when containing non-ASCII symbols
- Programs often try to predict the encoding of text files (often with success, but not always)
- Opening files assuming wrong encoding can give strange results....



Opening UTF-8 encoded file but trying to decode using Windows-1252



Opening Windows-1252 encoded file but trying to decode using UTF-8

en.wikipedia.org/wiki/Character_encoding

encoding.py

```
for filename in ["river-utf8.txt", "river-windows1252.txt"]:
    print(filename)
    f = open(filename, "rb")  # open input in binary mode, default = text mode = 't'
    line = f.readline()  # type(line) = bytes = immutable list of integers in 0..255
    print(line)  # literals for bytes look like strings, except for a prefix 'b'
    print(list(line))  # print bytes as list of integers
    f = open(filename, "r", encoding="utf-8")  # try to open file as UTF-8
    line = f.readline()  # fails if input line is not utf-8
    print(line)
```

river-utf8.py

```
Æ Æ U I Æ Å
```

Python shell

```
| river-utf8.txt
| b'\xc3\x86 \xc3\x86 U I \xc3\x86 \xc3\x85\r\n'  # \x
| [195, 134, 32, 195, 134, 32, 85, 32, 73, 32, 195, 134, 32, 195, 133, 13, 10]
| Æ Æ U I Æ Å

| river-windows1252.txt
| b'\xc6 \xc6 U I \xc6 \xc5\r\n'
| [198, 32, 198, 32, 85, 32, 73, 32, 198, 32, 197, 13, 10]
| UnicodeDecodeError: 'utf-8' codec can't decode byte 0xc6 in position 0: invalid
  continuation byte
```

Reading CSV files with specific encoding

read_shopping.py

```
import csv
with open("shopping.csv", encoding="Windows-1252") as file:
    for article, amount in csv.reader(file):
        print("Buy", amount, article)
```

Python shell

```
| Buy 2 æbler
| Buy 4 pærer
| Buy 3 jordbær
| Buy 10 gulerøder
```

shopping.csv

```
æbler,2
pærer,4
jordbær,3
gulerøder,10
```

CSV file saved with Windows-1252 encoding

JSON

*"**JSON** (**J**ava**S**cript **O**bject **N**otation) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate. It is based on a subset of the JavaScript Programming Language, Standard ECMA-262 3rd Edition - December 1999. JSON is an ideal data-interchange language."*

www.json.org

- Human readable file format
- Easy way to save a Python expression to a file
- Does not support all Python types, .e.g. sets are not supported, and tuples are saved (and later loaded) as lists

JSON example

json-example.py

```python
import json
FILE = 'json-data.json'
data = ((None, True), (42.7, (42,)), [3,2,4], (5,6,7),
        {'b':'banana', 'a':'apple', 'c': 'coconut'})

with open(FILE, 'w') as outfile:
    json.dump(data, outfile, indent=2, sort_keys=True)

with open(FILE) as infile:
    indata = json.load(infile)

print(indata)
```

Python shell

```
| [[None, True], [42.7, [42]], [3, 2, 4], [5, 6, 7], {'a':
  'apple', 'b': 'banana', 'c': 'coconut'}]
```

json-data.json


```json
[
  [
    null,
    true
  ],
  [
    42.7,
    [
      42
    ]
  ],
  [
    3,
    2,
    4
  ],
  [
    5,
    6,
    7
  ],
  {
    "a": "apple",
    "b": "banana",
    "c": "coconut"
  }
]
```

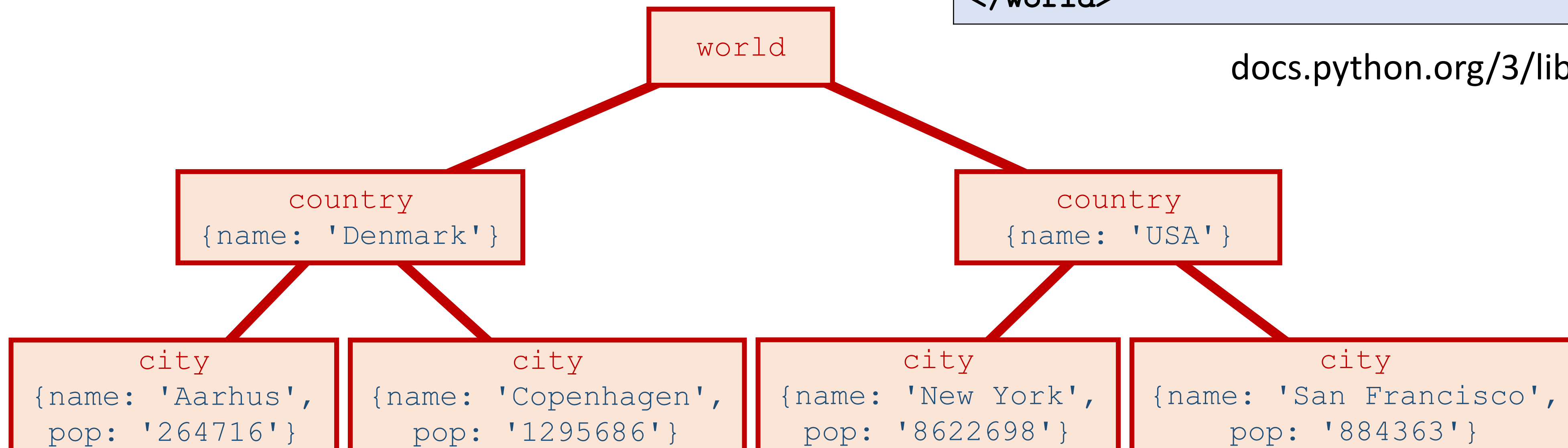
XML - eXtensible Markup Language

- XML is a widespread used data format to store hierarchical data with tags and attributes

cities.xml

```
<?xml version="1.0"?>
<world>
  <country name="Denmark">
    <city name="Aarhus" pop="264716"/>
    <city name="Copenhagen" pop="1295686"/>
  </country>
  <country name="USA">
    <city name="New York" pop="8622698"/>
    <city name="San Francisco" pop="884363"/>
  </country>
</world>
```

docs.python.org/3/library/xml.html



xml-example.py

```python
import xml.etree.ElementTree as ET
FILE = 'cities.xml'
tree = ET.parse(FILE)  # parse XML file to internal representation
root = tree.getroot()  # get root element
for country in root:
    for city in country:
        print(city.attrib['name'],  # get value of attribute for an element
              'in',
              country.attrib['name'],
              'has a population of',
              city.attrib['pop'])
print(root.tag, root[0][1].attrib)  # the tag & indexing the children of an element
print([city.attrib['name'] for city in root.iter('city')])  # .iter finds elements
```

Python shell

```
| Aarhus in Denmark has a population of 264716
  Copenhagen in Denmark has a population of 1295686
  New York in USA has a population of 8622698
  San Francisco in USA has a population of 884363
  world {'name': 'Copenhagen', 'pop': '1295686'}
  ['Aarhus', 'Copenhagen', 'New York', 'San Francisco']
```

XML tags with text

city-descriptions.xml

```
<?xml version="1.0"?>
<world>
  <country name="Denmark">
    <city name="Aarhus" pop="264716">The capital of Jutland</city>
    <city name="Copenhagen" pop="1295686">The capital of Denmark</city>
  </country>
  <country name="USA">
    <city name="New York" pop="8622698">Known as Big Apple</city>
    <city name="San Francisco" pop="884363">Home of the Golden Gate Bridge</city>
  </country>
</world>
```

xml-descriptions.py

```
import xml.etree.ElementTree as ET
FILE = 'city-descriptions.xml'
tree = ET.parse(FILE)
root = tree.getroot()

for city in root.iter('city'):
    print(city.get('name'), "-", city.text)
```

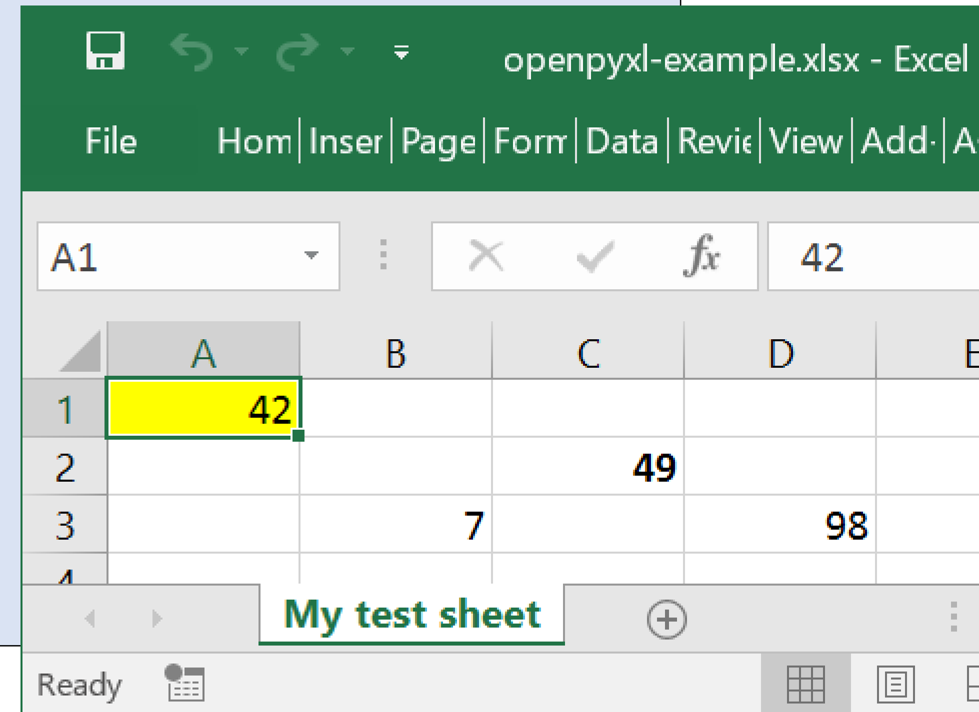
Python shell

```
Aarhus - The capital of Jutland
Copenhagen - The capital of Denmark
New York - Known as Big Apple
San Francisco - Home of the Golden Gate Bridge
```

Openpyxl - Microsoft Excel 2010 manipulation

openpyxl-example.py

```
from openpyxl import Workbook
from openpyxl.styles import Font, PatternFill

wb = Workbook() # create workbook
ws = wb.active  # active worksheet

ws['A1'] = 42
ws['B3'] = 7
ws['C2'] = ws['A1'].value + ws['B3'].value
ws['D3'] = '=A1+B3+C2'

ws.title = 'My test sheet'

ws['A1'].fill = PatternFill('solid', fgColor='ffff00')
ws['C2'].font = Font(bold=True)

wb.save("openpyxl-example.xlsx")
```



String searching using **`find`**

- Search for first occurence of *substring* in *str*[*start*, *end*]

  `str.find(substring[, start[, end]])`

- Returns -1 if no occurence found.
- `.index` similar as `.find`, except raises ValueError exception if substring not found

string-search.py

```
text = 'this is a string - a list of characters'
pattern = 'is'
idx = text.find(pattern)
while idx >= 0:
    print(idx, end=" ")
    idx = text.find(pattern, idx + 1)
```

Python shell

```
| 2 5 22
```

docs.python.org/3/library/stdtypes.html#textseq

Regular expression
– A powerful language to describe sets of strings

- **Examples**
  - `abc` denotes a string of letters
  - `ab*c` any string starting with `a`, followed by an arbitrary number of `b`s and terminated by `c`, i.e. {`ac`, `abc`, `abbc`, `abbbc`, `abbbbc`, ...}
  - `ab+c` as `ab*c`, except that there must be at least one `b`
  - `a\wc` any three letter string, starting with `a` and ending with `c`, where second character is any character in `[a-zA-Z0-9_]`
  - `a[xyz]c` any three letter string, starting with `a` and ending with `c`, where second character is either `x`, `y` or `z`
  - `a[^xyz]c` any three letter string, starting with `a` and ending with `c`, where second character is none of `x`, `y` or `z`
  - `^xyz` match at start of string
  - `xyz$` match at end of string
  - ...
- See docs.python.org/3/library/re.html for more

String searching using regular expressions

- `re.search(`*`pattern`*`, `*`text`*`)`
  - find the first occurence of *pattern* in *text* – returns `None` or a *match object*
- `re.findall(`*`pattern`*`, `*`text`*`)`
  - returns a list of non-overlapping occurence of *pattern* in *text* – returns a list of substrings
- `re.finditer(`*`pattern`*`, `*`text`*`)`
  - iterator returning a match object for each non-overlapping occurence of *pattern* in *text*

Python shell

```
> text = 'this is a string - a list of characters'
> re.findall(r'i\w*', text)  # prefix with 'r' for raw string literal
| ['is', 'is', 'ing', 'ist']
> for m in re.finditer(r'a[^at]*t', text):
      print("text[%s, %s] = %s" % (m.start(), m.end(), m.group()))
| text[8, 12] = a st
  text[19, 25] = a list
  text[33, 36] = act
```

docs.python.org/3/library/re.html

Substitution and splitting using regular expressions

- `re.sub(`*`pattern`*`, `*`replacement`*`, `*`text`*`)`
  - replace any occurence of the *`pattern`* in *`text`* by *`replacement`*
- `re.split(`*`pattern`*`, `*`text`*`)`
  - split *`text`* at all occurences of *`patern`*

Python shell

```
> text = 'this is a string - a list of characters'
> re.sub(r'\w*i\w*', 'X', text)
| 'X X a X - a X of characters'
> re.split(r'[^\w]+a[^\w]+', text)
| ['this is', 'string', 'list of characters']
```

docs.python.org/3/library/re.html

Fun with strings: Lindenmayer systems (L-systems)

Axiom	A
(1)	AB
(2)	ABA
(3)	ABAAB
(4)	ABAABABA

Rules

A → AB

B → A

First four iterations of parallel rewriting

"L-systems were introduced and developed in 1968 by Aristid Lindenmayer, a Hungarian theoretical biologist and botanist at the University of Utrecht. Lindenmayer used L-systems to describe the behaviour of plant cells and to model the growth processes of plant development."

- Wikipedia

L_system.py

```
S = 'A'  # axiom
rules = {'A': 'AB', 'B': 'A'}
for i in range(8):
    S = "".join(rules.get(c, c) for c in S)
    print(S)
```

Python shell

```
| AB
| ABA
| ABAAB
| ABAABABA
| ABAABABAABAAB
| ABAABABAABAABABAABABA
| ABAABABAABAABABAABABAABAABABAABAAB
```

en.wikipedia.org/wiki/L-system

Heighway Dragon

dragon.py

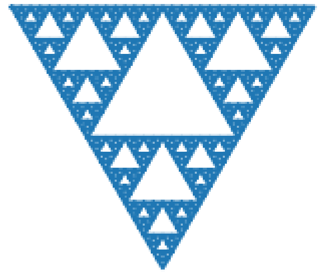
```python
import matplotlib.pyplot as plt
from math import sin, cos, radians

axiom = 'FX'
rules = {'X': 'X+YF+', 'Y': '-FX-Y'}

def apply_rules(axiom, rules, repeat):
    for _ in range(repeat):
        axiom = ''.join(rules.get(symbol, symbol) for symbol in axiom)
    return axiom

def walk(commands, position=(0, 0), angle=0, turn=90):
    path = [position]
    for move in commands:
        if move == 'F':
            position = (position[0] + cos(radians(angle)),
                        position[1] + sin(radians(angle)))
            path.append(position)
        elif move == '-': angle -= turn
        elif move == '+': angle += turn
    return path

path = walk(apply_rules(axiom, rules, 13))
plt.plot(*zip(*path), '-')
plt.title('Heighway dragon')
plt.show()
```

Interprete the symbols of the resulting string as a walk where 'F' = draw line forward, and '+' and '-' are turn left and right 90° (X and Y are skipped)

FX+YF++-FX-YF++-FX+YF+--FX-YF++-FX+YF++-FX-YF+--FX+YF+--FX-YF++-FX+YF++-FX-YF++-FX+YF+--FX-YF+--FX+YF++-FX-YF+--FX+YF+--FX-YF++-FX+YF++-FX-YF++-FX+YF+--FX-YF++-FX+YF++-FX-YF+--FX+Y

More space filling curves...

Sierpinski triangle



Axiom F-G-G
F → F-G+F+G-F
G→ GG

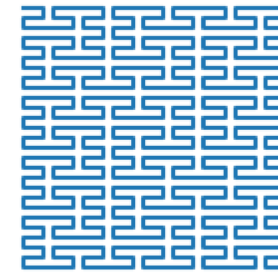
Forward F and G
Turns 120°

Sierpinski arrowhead curve

Axiom A
A → B-A-B
B → A+B+A

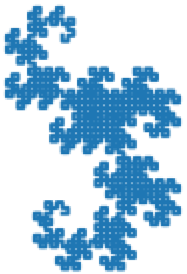
Forward A and B
Turns 60°

Peano curve



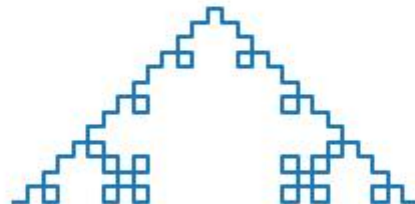
Axiom L
L → LFRFL-F-RFLFR+F+LFRFL
R→ RFLFR+F+LFRFL-F-RFLFR

Heighway dragon



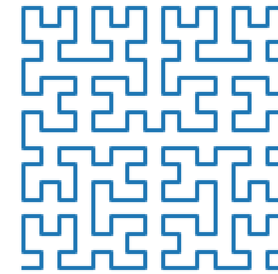
Axiom FX
X → X+YF+
Y → -FX-Y

Koch curve



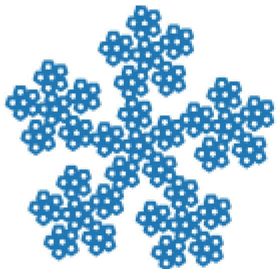
Axiom F
F → F+F-F-F+F

Hilbert curve



Axiom L
L → +RF-LFL-FR+
R → -LF+RFR+FL-

McWorter Pentigree curve



Axiom F-F-F-F-F
F → F-F-F++F+F-F

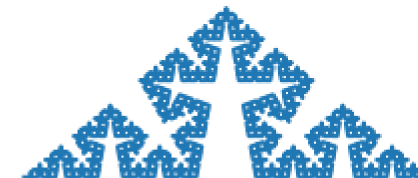
Turns 72°

Tree

Axiom F
F → F[+FF][-FF]F[-F][+F]F

Turns 36°
[and] return to start point when done

Cesero fractal



Axiom F
F → F+F--F+F

Turns 80°

More space filling curves... (source code)

space-filling-L_systems.py

```python
import matplotlib.pyplot as plt
from math import sin, cos, radians

def walk(commands,
         pos=(0, 0),
         forward=frozenset('F'),
         angle=0,
         turn=90):
  paths = [[pos]]
  stack = []
  for move in commands:
    if move in forward:
      pos = (pos[0]+cos(radians(angle)),
             pos[1]+sin(radians(angle)))
            paths[-1].append(pos)
    elif move == '-': angle -= turn
    elif move == '+': angle += turn
    elif move == '[':
      stack.append((pos, angle))
    elif move == ']':
      pos, angle = stack.pop()
      paths.append([pos])
  return paths
```

```python
def apply_rules(axiom, rules, repeat=1):
    for _ in range(repeat):
        axiom = ''.join(rules.get(symbol, symbol) for symbol in axiom)
    return axiom

curves = [  # Lindenmayer systems (L-systems)
  ('Sierpinski triangle', 'F-G-G', {'F': 'F-G+F+G-F', 'G': 'GG'}, 5, {'turn': 120, 'forward': {'F','G'}}),
  ('Sierpinski arrowhead curve', 'A', {'A': 'B-A-B', 'B': 'A+B+A'}, 5, {'turn': 60, 'forward': {'A','B'}}),
  ('Peano curve', 'L', {'L': 'LFRFL-F-RFLFR+F+LFRFL', 'R': 'RFLFR+F+LFRFL-F-RFLFR'}, 3, {}),
  ('Heighway dragon','FX', {'X': 'X+YF+', 'Y': '-FX-Y'}, 10, {}),
  ('Koch curve', 'F', {'F': 'F+F-F-F+F'}, 3, {}),
  ('Hilbert curve', 'L', {'L': '+RF-LFL-FR+', 'R': '-LF+RFR+FL-'}, 4, {}),
  ('McWorter Pentigree curve', 'F-F-F-F-F', {'F': 'F-F-F++F+F-F'}, 3, {'turn': 72}),
  ('Tree', 'F', {'F': 'F[+FF][-FF]F[-F][+F]F'}, 3, {'turn': 36}),
  ('Cesero fractal', 'F', {'F': 'F+F--F+F'}, 5, {'turn': 80})
]

for idx, (title, axiom, rules, repeat, walk_arg) in enumerate(curves, start=1):
    paths = walk(apply_rules(axiom, rules, repeat), **walk_arg)
    ax = plt.subplot(3, 3, idx, aspect='equal')
    ax.set_title(title)
    for points in paths:
        X, Y = zip(*points)
        plt.plot(X, Y, '-')
    plt.axis('off')
plt.show()
```