Forritunarmálið Python

Day 4

Regex and Exceptions

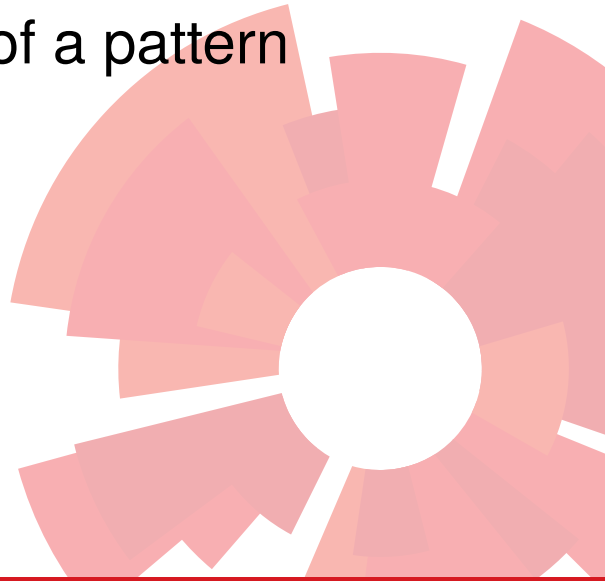
Hjalti Magnússon

30. nóvember 2017

Regular expressions

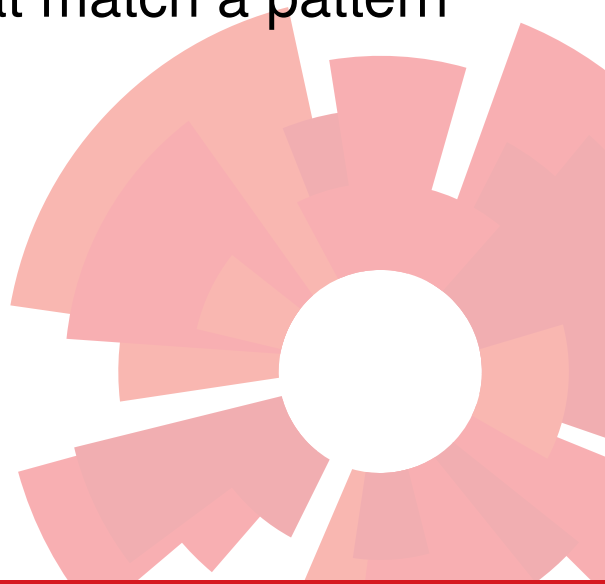
What are they?

- Search for a pattern in a string
- Application range from
  - Finding a pattern
  - Replacing occurrences of patterns
  - Splitting strings on every occurrence of a pattern



Available applications

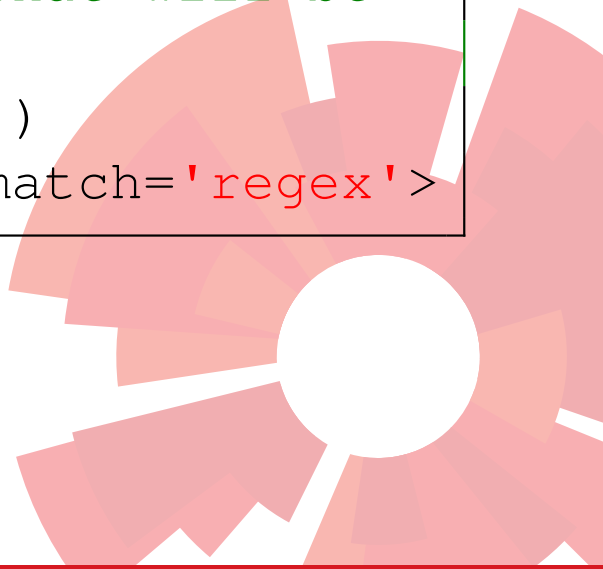
- `search`, `match`, `fullmatch` - Search a string for a pattern
- `findall`, `finditer` - Find all substrings that match a pattern
- `split` - Splits a string on all substrings that match a pattern
- `sub` - Replaces all substrings in a string that match a pattern with a new string



Pattern syntax

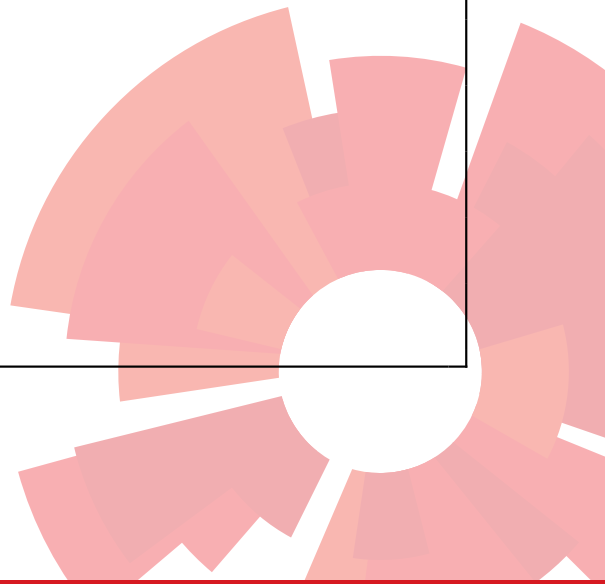
- All characters match themselves, except a few *special* characters
- The Python documentation gives a good overview

```
>>> import re
# The regex is the first parameter
# The second parameter is the string that will be
    searched
>>> re.search('regex', 'regexes rock!')
<_sre.SRE_Match object; span=(0, 5), match='regex'>
```



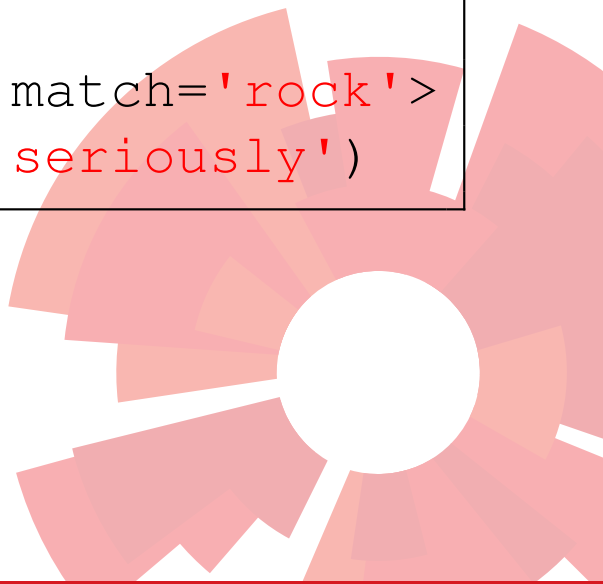
The Match object

```
>>> import re
>>> m = re.search('regex', 'regexes rock!')

# The string that was matched
>>> m.group()
'regex'

# Start/end position of match
>>> m.start()
0
>>> m.end()
5
>>> m.span()
(0, 5)
```



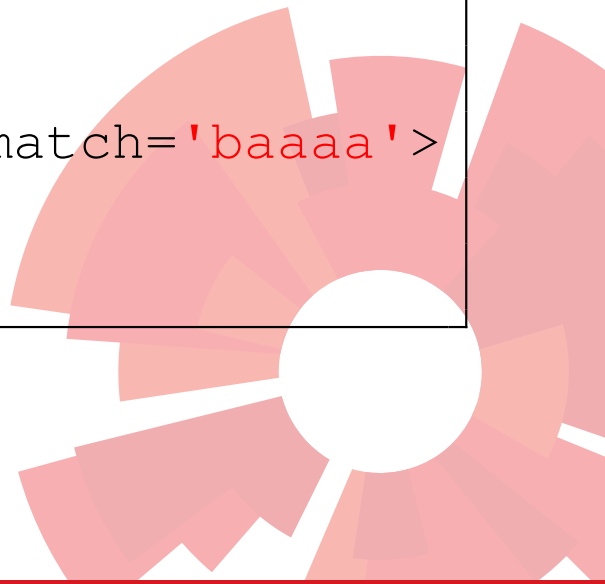
Start and end of strings

```
# ^ matches beginning of string
# $ matches end of string

>>> re.search('^regex', 'regexes rock!')
<_sre.SRE_Match object; span=(0, 5), match='regex'>
>>> re.search('^regex', 'really, regexes rock!')

>>> re.search('rock$', 'regexes rock')
<_sre.SRE_Match object; span=(8, 12), match='rock'>
>>> re.search('rock$', 'regexes rock, seriously')
```



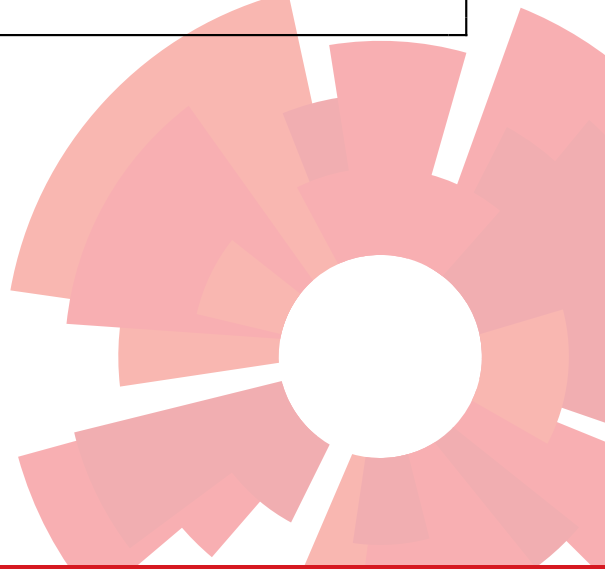
Repetitions

```
# * matches 0 or more repetitions of pattern
# + matches 1 or more repetitions of pattern

>>> re.search('ba*', 'baaaa')
<_sre.SRE_Match object; span=(0, 5), match='baaaa'>
>>> re.search('ba*', 'b')
<_sre.SRE_Match object; span=(0, 1), match='b'>
>>> re.search('ba*', 'c')

>>> re.search('ba+', 'baaaa')
<_sre.SRE_Match object; span=(0, 5), match='baaaa'>
>>> re.search('ba+', 'b')
>>> re.search('ba+', 'c')
```



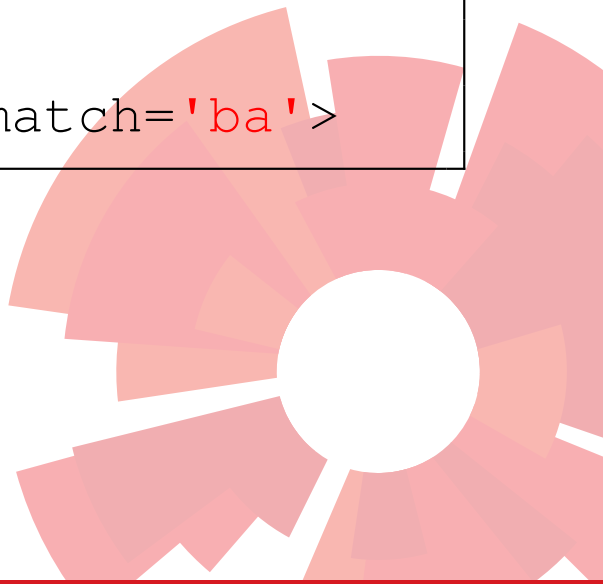
Maybe

```
>>> re.search('ba?', 'b')
<_sre.SRE_Match object; span=(0, 1), match='b'>
>>> re.search('ba?', 'ba')
<_sre.SRE_Match object; span=(0, 2), match='ba'>
>>> re.search('ba?', 'baa')
<_sre.SRE_Match object; span=(0, 2), match='ba'>
```



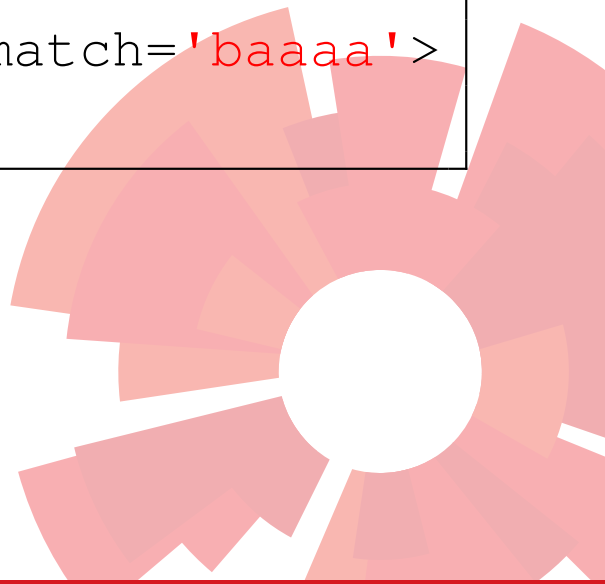
Non-greedy repetitions

```
# *? and +? match as few repetitions as possible

>>> re.search('ba*?', 'baaaa')
<_sre.SRE_Match object; span=(0, 1), match='b'>
>>> re.search('ba*?c', 'baaac')
<_sre.SRE_Match object; span=(0, 5), match='baaac'>

>>> re.search('ba+?', 'baaaa')
<_sre.SRE_Match object; span=(0, 2), match='ba'>
```



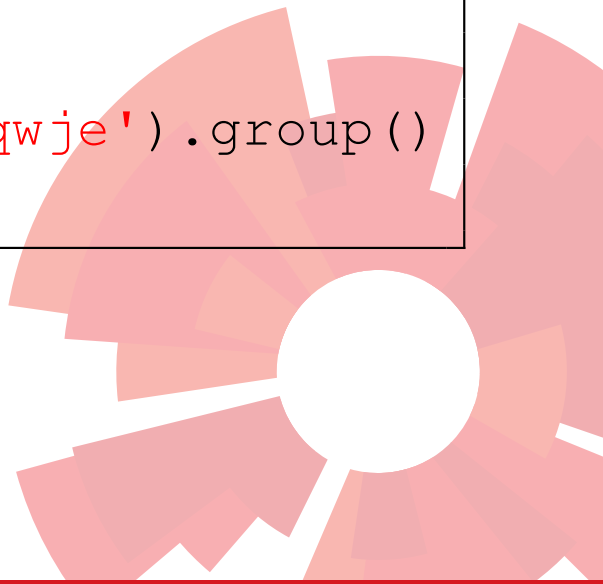
Limited repetitions

```
# {n} matches pattern n times
# {n,m} matches pattern n to m times

>>> re.search('ba{3}', 'baaaa')
<_sre.SRE_Match object; span=(0, 4), match='baaa'>

>>> re.search('ba{3,5}', 'baaaa')
<_sre.SRE_Match object; span=(0, 5), match='baaaa'>
>>> re.search('ba{3,5}', 'baa')
```



Character sets

```
# [] matches sets of characters

>>> re.search('b[ac]', 'ba')
<_sre.SRE_Match object; span=(0, 2), match='ba'>
>>> re.search('b[ac]', 'bc')
<_sre.SRE_Match object; span=(0, 2), match='bc'>
>>> re.search('b[ac]+', 'bccacaa').group()
'bccacaa'

>>> re.search('b[a-z]+', 'babkdjfaewrqwje').group()
'babkdjfaewrqwje'
```

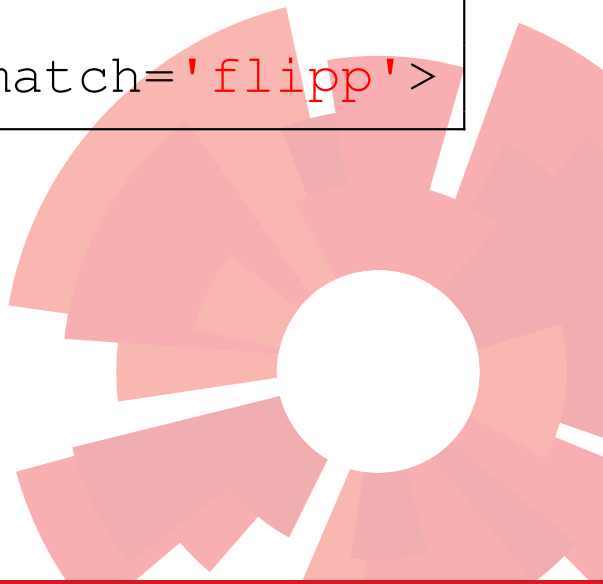


Character sets

```
# [^...] matches the complement of sets of
    characters

>>> re.search('b[^ac]', 'ba')
>>> re.search('b[^ac]', 'bx')
<_sre.SRE_Match object; span=(0, 2), match='bx'>
>>> re.search('b[^a-z]+', 'bAEW3?8xTEST').group()
'bAEW3?8'
>>> re.search('b[^0-9A-Za-z]+', 'b./&%#ab').group()
'b./&%#'
```

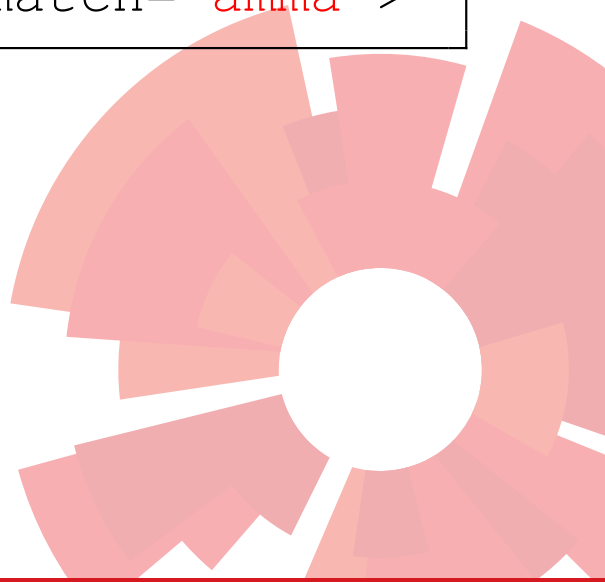
Or

```
# A|B matches either the pattern A or B

>>> re.search('b[^ac]|ba', 'ba')
<_sre.SRE_Match object; span=(0, 2), match='ba'>
>>> re.search('test|flipp', 'test')
<_sre.SRE_Match object; span=(0, 4), match='test'>
>>> re.search('test|flipp', 'flipp')
<_sre.SRE_Match object; span=(0, 5), match='flipp'>
```



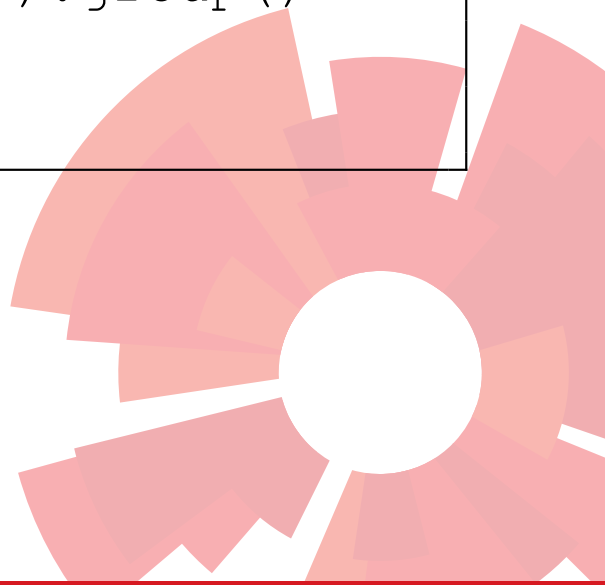
Groups

```
# (...) groups patterns to make a large pattern

>>> re.search('a(mm|nn)a', 'anna')
<_sre.SRE_Match object; span=(0, 4), match='anna'>
>>> re.search('a(mm|nn)a', 'amma')
<_sre.SRE_Match object; span=(0, 4), match='amma'>
```



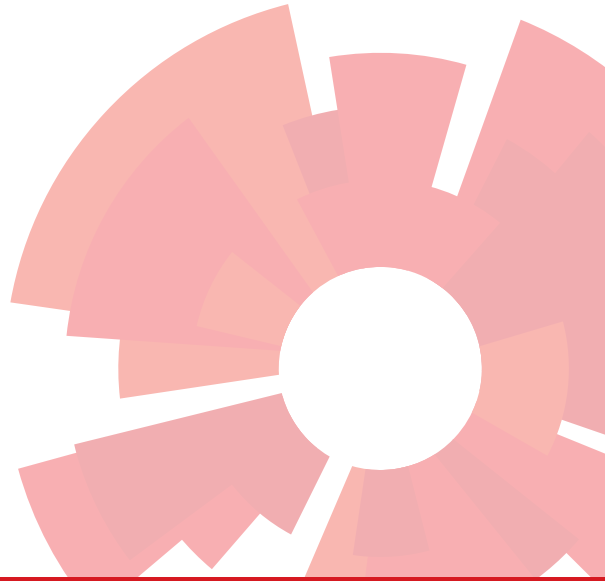
Groups

```
>>> re.search('a(mm|nn)*a', 'aa').group()
'aa'
>>> re.search('a(mm|nn)*a', 'amma').group()
'amma'
>>> re.search('a(mm|nn)*a', 'ammmmmma').group()
'ammmmmma'
>>> re.search('a(mm|nn)*a', 'ammnnmma').group()
'ammnnmma'
>>> re.search('a(mm|nn)*a', 'ammma')
```



Special sets

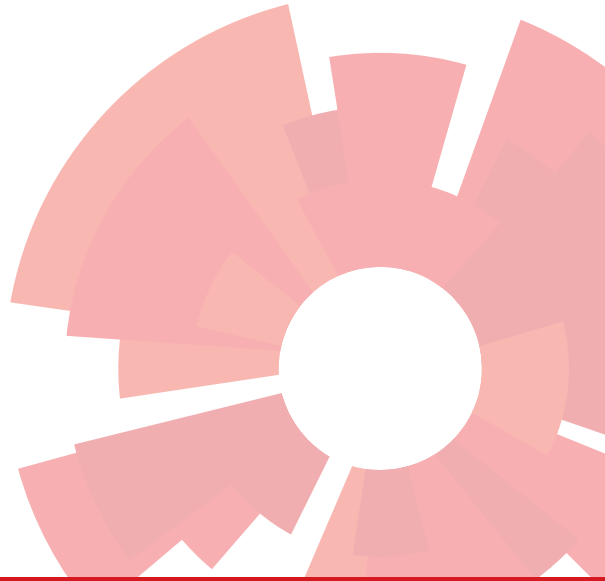
- `\w` - Matches word characters (letters, digits, underscore)
- `\W` - Matches non-word characters
- `\d` - Matches digits
- `\D` - Matches non-digits
- `\s` - Matches whitespace
- `\D` - Matches non-whitespace
- `\b` - Matches word border



Problems

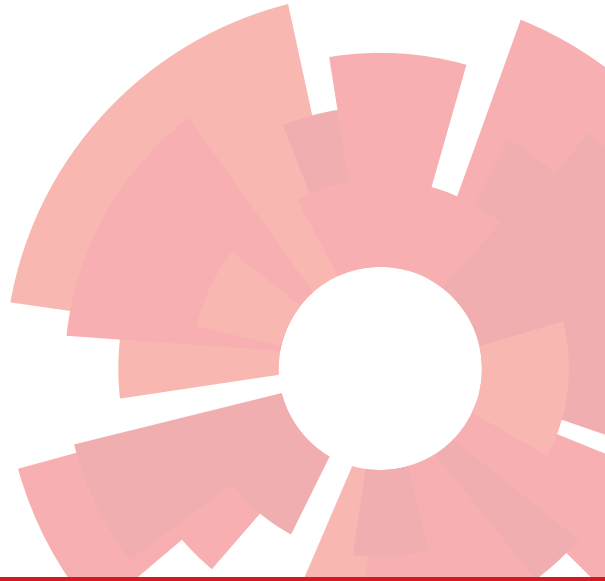
Problem 1

Find all words ending with “ing”



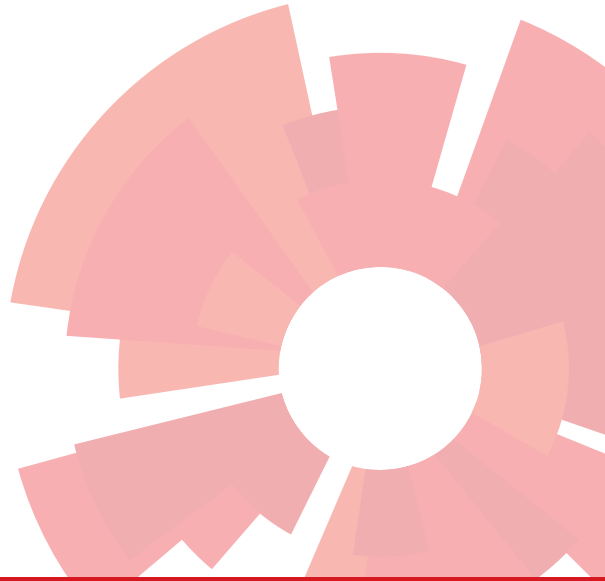
Problem 2

Match all valid Icelandic phone numbers



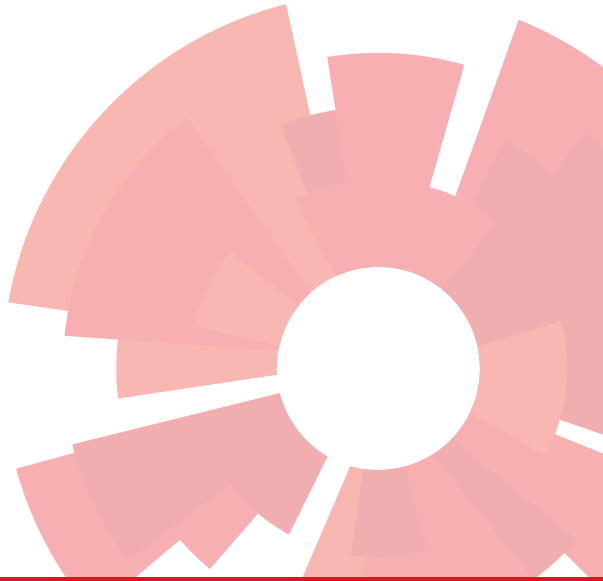
Problem 3

Find a full name in a string, First name, optional middle name, and a Last name



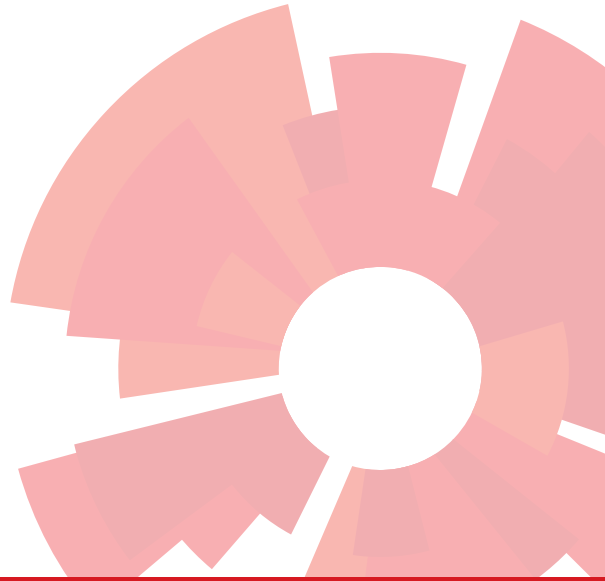
Problem 4

Match a string containing a C++ string



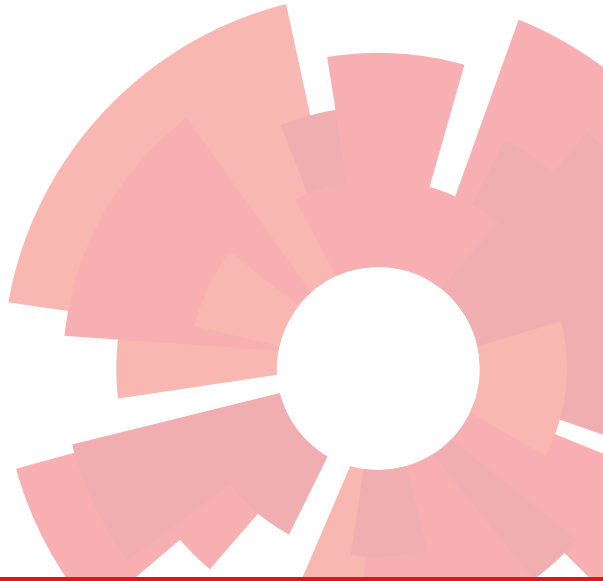
Problem 5

Split a string on two or more spaces



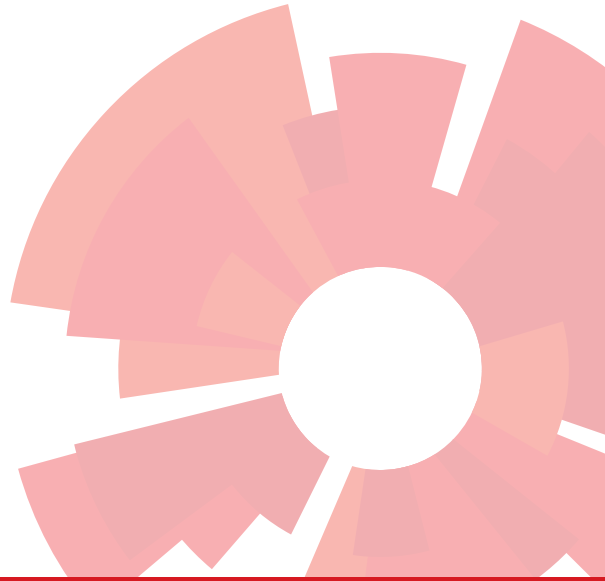
Problem 6

Parse “config” file



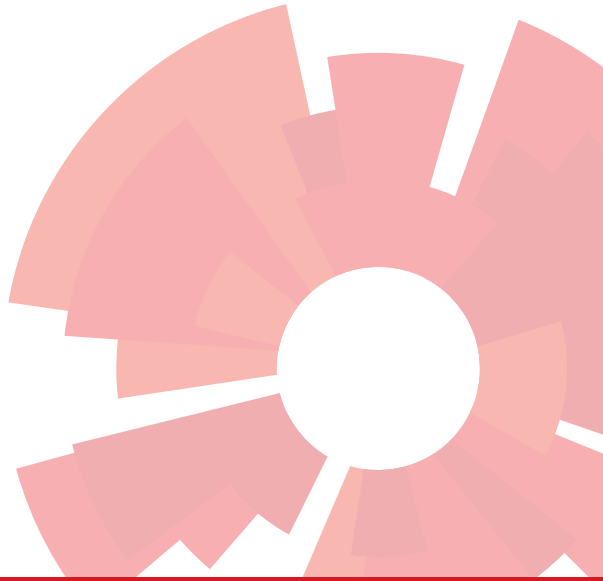
Problem 7

Change all words ending with “ing” to “nodding”



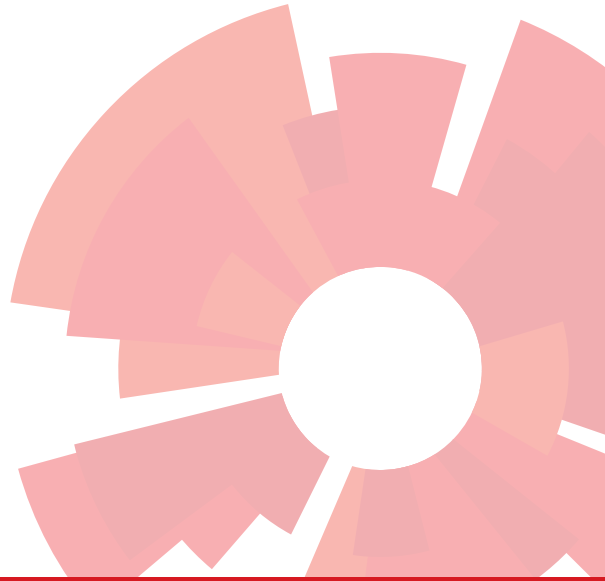
Problem 8

Increase all years by one



Problem 9

Find XML tag with content

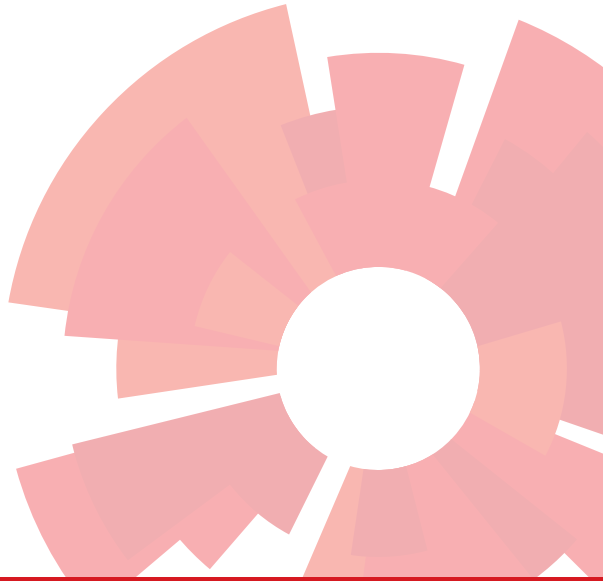


Final words

Solution looking for a problem

Regular expressions are cool, but that does not mean that you should use them to solve all your problems!

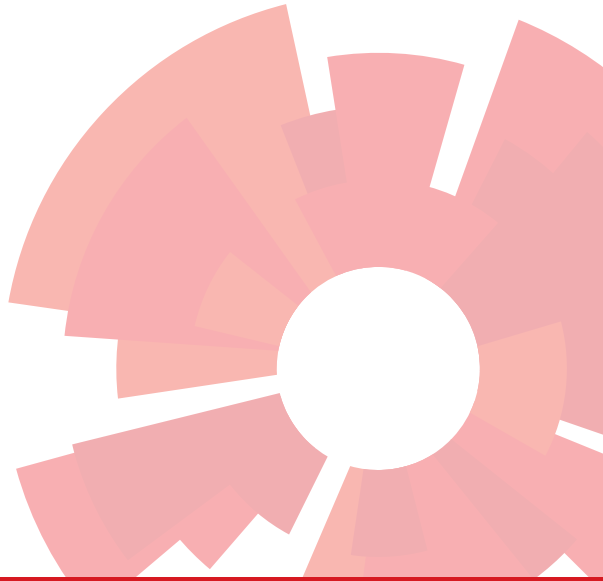
> *Some people, when confronted with a problem, think "I know, I'll use regular expressions." Now they have two problems. —A very wise person*



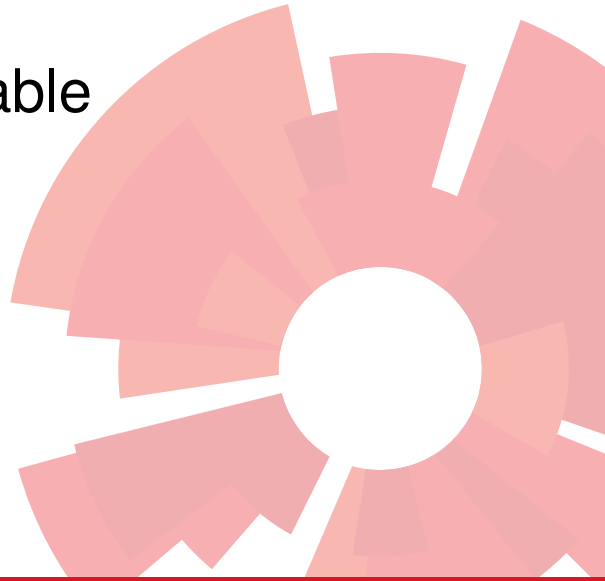
`itertools` - efficient looping

Advanced iteration

- The module provides very nice methods to work with iteratos and iterables
- We will focus on functions that help us, in some sense, generate “all options”



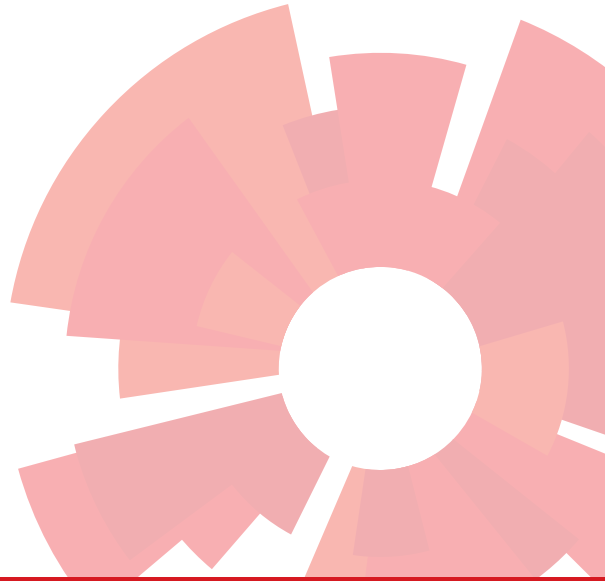
- `product`
  - Generates the product of two or more iterables
- `permutations`
  - Generates all permutations of an iterable
- `combinations`
  - Generates all combinations of an iterable
- `groupby`



Problems

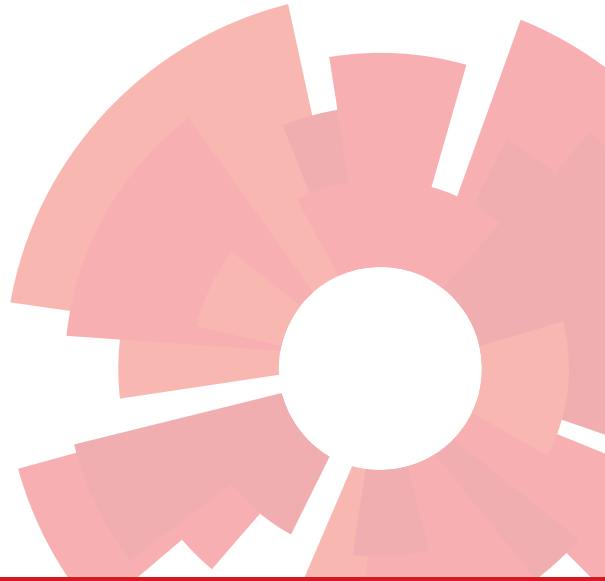
Problem 1

Generate a deck of cards



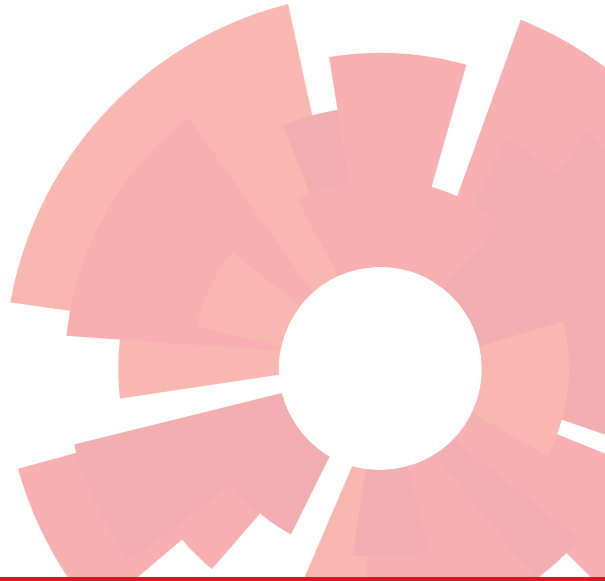
Problem 2

Generate all binary strings of length 10



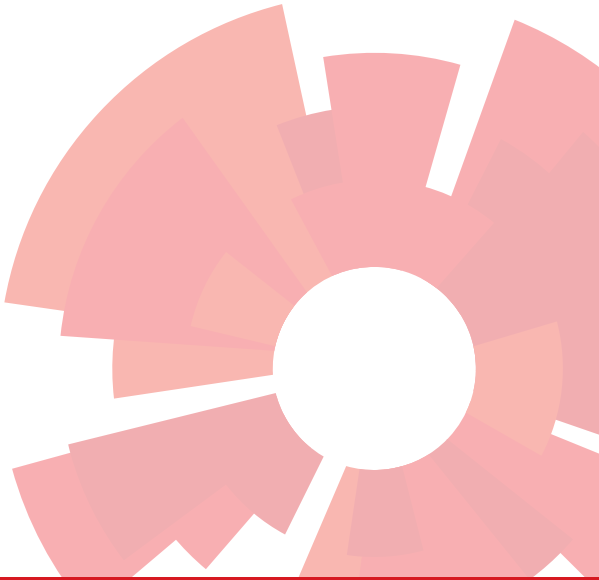
Problem 3

Subset sum



Problem 4

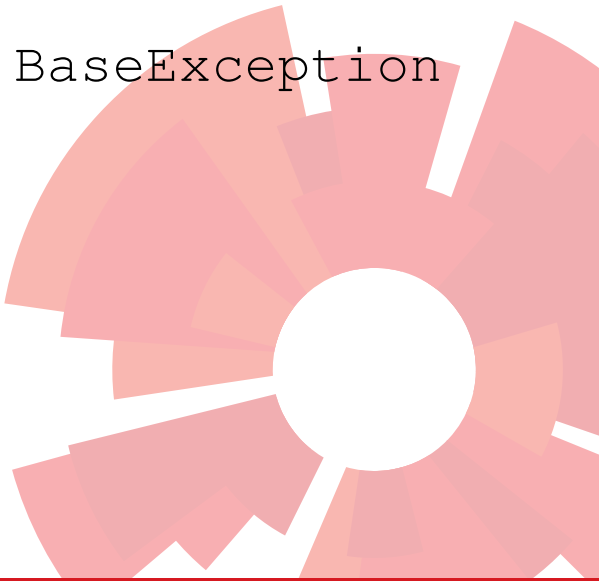
Find all numbers that can be formed using a list of digits



Exceptions

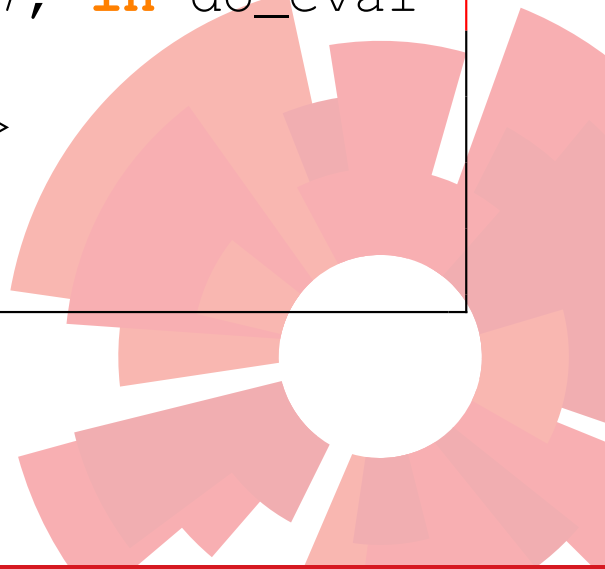
Raising exceptions

- Exceptions denote exceptional behaviour
- Used more in Python than most languages
  - Also used as a part of normal program flow
- The keyword `raise` is used to raise exceptions
  - The exception raised must inherit from `BaseException`



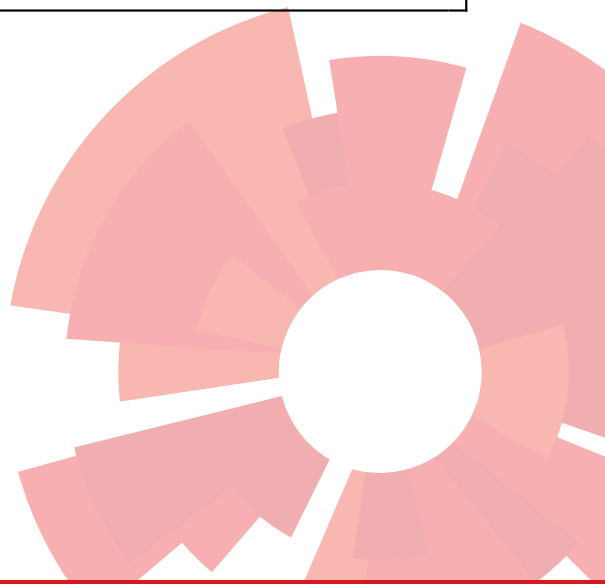
Raising exceptions

```
>>> def div(a, b):
...     if b == 0:
...         raise Exception('Cannot divide by zero')
...     return a / b

>>> div(3, 0)
Traceback (most recent call last):
  File "/home/hjalti/doc/teaching/Python/schedule/.
     lecture/hooks/pyeval.py", line 17, in do_eval
    return eval(s, context)
  File "<string>", line 1, in <module>
  File "<string>", line 3, in div
Exception: Cannot divide by zero
```



Catching exceptions

```
>>> try:
...     int('test')
... except ValueError as e:
...     print(e)
invalid literal for int() with base 10: 'test'
```



More on exception control structures

```
try:
    do_stuff()
except ValueError as e:
    # Handle ValueError
except KeyError:
    # Handle KeyError
except:
    # Handle all other exceptions
else:
    # If no error was raised
finally:
    # Always executed
```