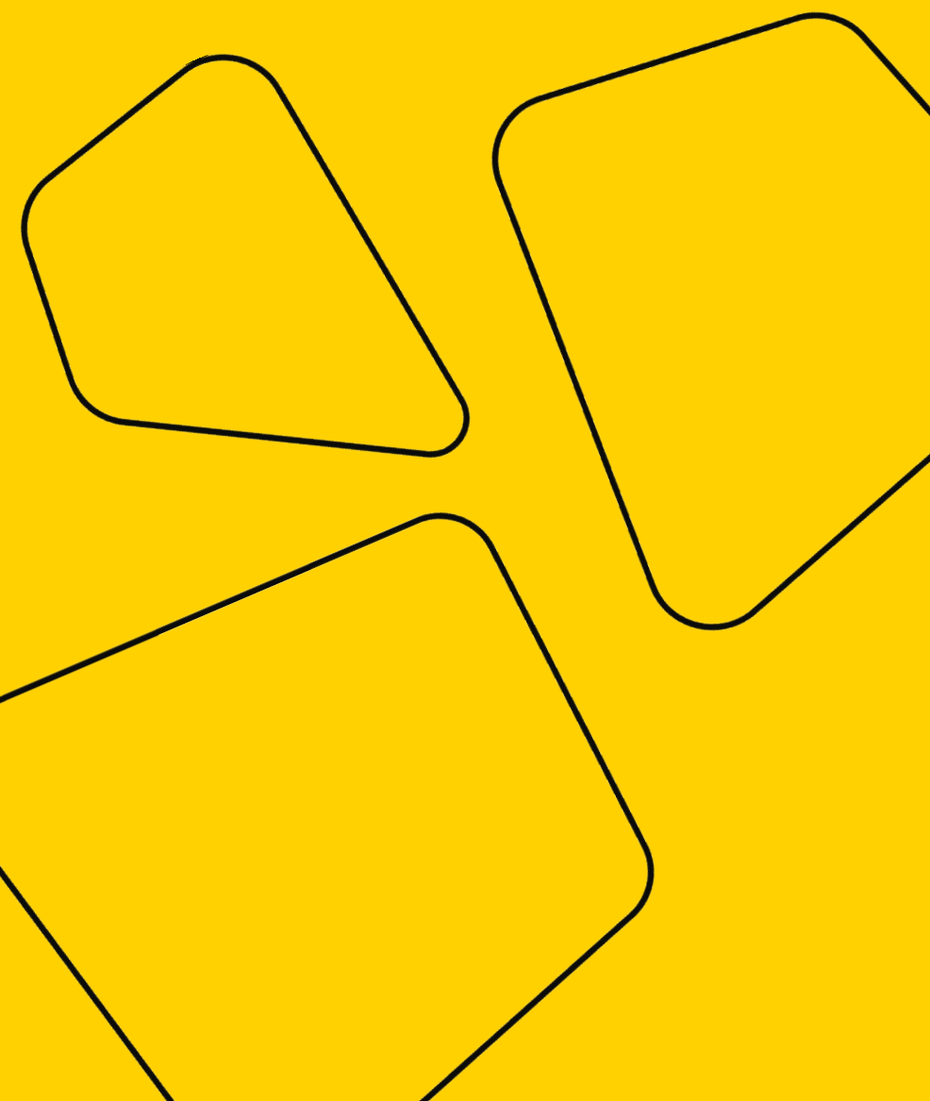
Functions

Software Development & Python

Nick Levashov, 2021

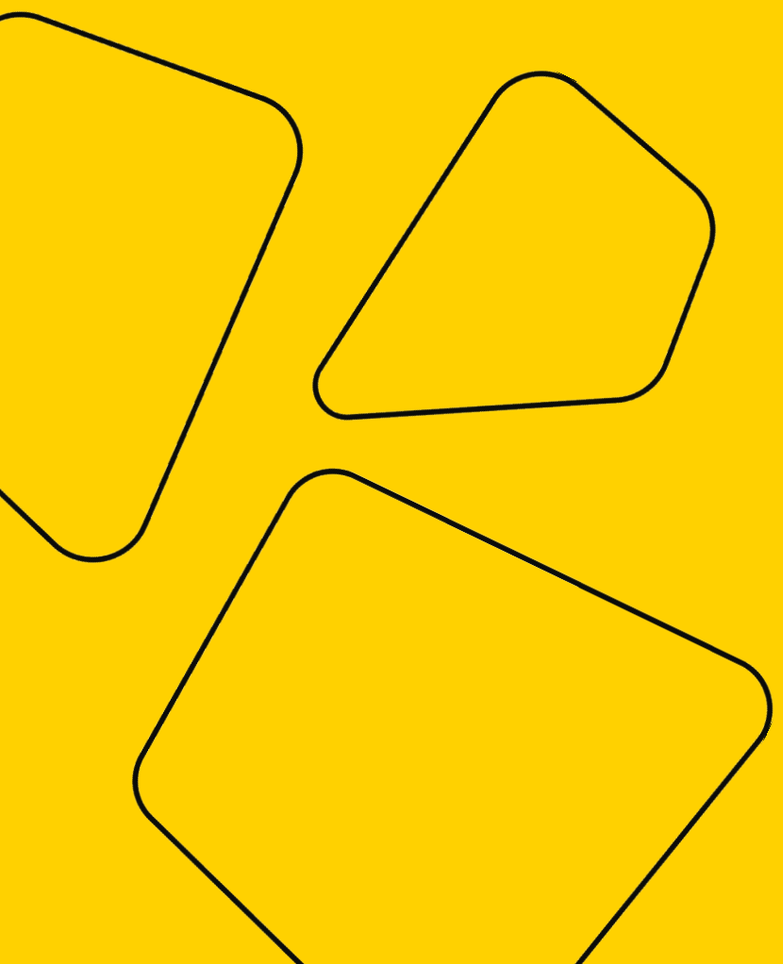


girafe
ai

Functions

```
template = 'Hello, {name}!'

def say_hi(name, *other_names):
    print(template.format(name=name))
    if other_names:
        print('and others!')
```

Decorators

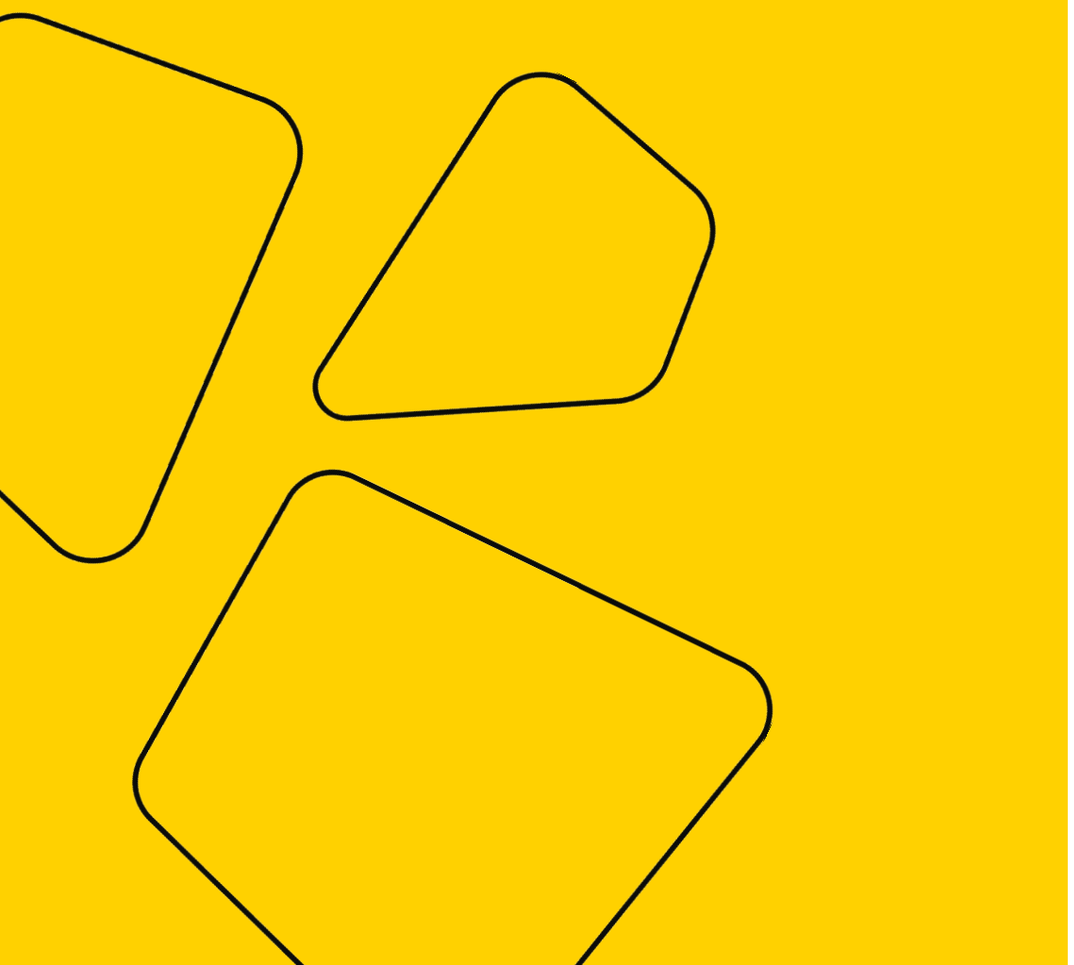


```
def add_bun(func):
    def wrapper():
        print('bun top')
        func()
        print('bun bottom')
    return wrapper


@add_bun
def patty():
    print('beef')
```

Packing & Unpacking

```
a, *b, c = 1, 2, 3, 4, 5

a = {'a': 1}
b = {**a, 'b': 2}
```

Functional programming



```
>>> l1 = [0, 1, 2]
>>> l2 = [3, 4, 5]
>>>
>>> sum(map(operator.mul, l1, l2))
14
```

jupyter Welcome to P

File Edit View Insert Cell

jupyter

Welcome to the

This Notebook Server wa

WARNING

Don't rely on this serv

Your server is hosted thar

Run some Python

To run the code below:

1. Click on the cell to se
2. Press SHIFT+ENTER

A full tutorial for using the

```
In [ ]: %matplotlib inline

        import pandas as pd
        import numpy as np
        import matplotlib
```

jupyter Lorenz Differential Equations (autosaved)

File Edit View Insert Cell Kernel Help Python 3

Code Cell Toolbar: None

Exploring the Lorenz System

In this Notebook we explore the Lorenz system of differential equations:

$$\dot{x} = \sigma(y - x)$$
$$\dot{y} = \rho x - y - xz$$
$$\dot{z} = -\beta z + xy$$

This is one of the classic systems in non-linear differential equations. It exhibits a range of complex behaviors as the parameters (σ, β, ρ) are varied, including what are known as *chaotic solutions*. The system was originally developed as a simplified mathematical model for atmospheric convection in 1963.

```
In [7]: interact(Lorenz, N=fixed(10), angle=(0.,360.),
                σ=(0.0,50.0),β=(0.,5), ρ=(0.0,50.0))
```

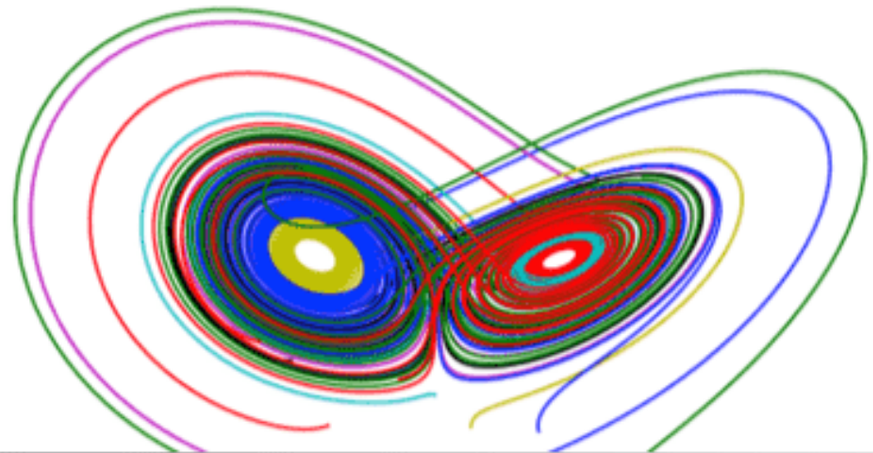
angle 308.2

max_time 12

σ 10

β 2.6

ρ 28



girafe
ai

Decorator updates (3.9)

```
# old syntax
decorator: '@' dotted_name [ '(' [arglist] ')' ] NEWLINE
```

```
# new syntax
decorator: '@' namedexpr_test NEWLINE
```

Decorator updates (3.9)

```
buttons = [QPushButton(f'Button {i}') for i in range(10)]
```

Decorator updates (3.9)

```
buttons = [QPushButton(f'Button {i}') for i in range(10)]

@buttons[0].clicked.connect
def spam():
    pass

@buttons[1].clicked.connect
def eggs():
    pass
```

Decorator updates (3.9)

```
buttons = [QPushButton(f'Button {i}') for i in range(10)]

@buttons[0].clicked.connect
def spam():
    pass

@buttons[1].clicked.connect
def eggs():
    pass

# before python3.9:

button_0 = buttons[0]

@button_0.clicked.connect
def spam():
    pass
```

Packing and Unpacking

girafe ai

```
>>> def pack_args(*args, **kwargs):
...     return args, kwargs
...
>>> args, kwargs = pack_args(1, 2, c=3, d=4)
>>> args
(1, 2)
>>> kwargs
{'c': 3, 'd': 4}
```

```
>>> a, b, c = iterable
```

```
>>> a, b, c = 1, 2, 3
```

```
>>> a, b, c = 1, 2, 3

>>> t = 1, 2, 3
>>> type(t)
<class 'tuple'>
>>> a, b, c = t
```

```
>>> a, b, c = 1, 2, 3

>>> t = 1, 2, 3
>>> type(t)
<class 'tuple'>
>>> a, b, c = t
```

```
>>> a, b, c = [1, 2, 3]
>>> a, b, c
(1, 2, 3)
>>> a, b, c = 'abc'
>>> a, b, c
('a', 'b', 'c')
```

```
>>> a, b, c = {1, 2, 3}
>>> a, b, c
(1, 2, 3)

>>> a, b, c = {3, 2, 1}
>>> a, b, c
(1, 2, 3)

>>> a, b, c = {'a': 1, 'b': 2, 'c': 3}
>>> a, b, c
('a', 'b', 'c')
```

```
>>> a, *b, c = 1, 2, 3, 4, 5
>>> a
1
>>> b
[2, 3, 4]
>>> c
5
```

```
>>> a, *b, c = 1, 2, 3, 4, 5
>>> a
1
>>> b
[2, 3, 4]
>>> c
5
```

```
>>> def pack_args(a, *b):
...     print(b)
...
>>> pack_args(1, 2, 3, 4)
(2, 3, 4)
```

```
>>> a = [1, 2, 3]
>>> b = [4, 5, 6]
>>> c = [*a, *b, 7, 8, 9]
>>> c
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
>>> a = [1, 2, 3]
>>> b = [4, 5, 6]
>>> c = [*a, *b, 7, 8, 9]
>>> c
[1, 2, 3, 4, 5, 6, 7, 8, 9]

>>> t = *a, *b, 7, 8, 9
>>> s = {*a, *b, 7, 8, 9}
```

```
>>> a = [1, 2, 3]
>>> b = [4, 5, 6]
>>> c = [*a, *b, 7, 8, 9]
>>> c
[1, 2, 3, 4, 5, 6, 7, 8, 9]

>>> t = *a, *b, 7, 8, 9
>>> s = {*a, *b, 7, 8, 9}

>>> def pack_args(*args):
...         return args
...
>>> pack_args(*a, *b, 7, 8, 9)
(1, 2, 3, 4, 5, 6, 7, 8, 9)
```

```
>>> def get_3d_point(*args, **kwargs):
...     return 0, 1, 0.9
...
>>> x, y, z = get_3d_point()
>>> point = get_3d_point()
```

```
>>> points = [(0, 1), (2.5, 3.5), (1, 1)]
>>> for x, y in points:
...     pass
...
```

```
>>> points = [(0, 1), (2.5, 3.5), (1, 1)]
>>> for x, y in points:
...     pass
...
>>> for key, value in {}.items():
...     pass
...
```

```
>>> points = [(0, 1), (2.5, 3.5), (1, 1)]
>>> for i, point in enumerate(points):
...     print(i, point)
0 (0, 1)
1 (2.5, 3.5)
2 (1, 1)
```

```
>>> points = [(0, 1), (2.5, 3.5), (1, 1)]
>>> for i, point in enumerate(points):
...     print(i, point)
0 (0, 1)
1 (2.5, 3.5)
2 (1, 1)
```

```
>>> for i, (x, y) in enumerate(points):
...     print(i, x, y)
...
0 0 1
1 2.5 3.5
2 1 1
```

```
>>> points = [(0, 1), (2.5, 3.5), (1, 1)]

>>> xs = [0, 2.5, 1]
>>> ys = [1, 3.5, 1]
>>> for x, y in zip(xs, ys):
...     print(x, y)
...
0 1
2.5 3.5
1 1
```

```
>>> def pack_args(*args, **kwargs):
...     return args, kwargs
...
>>> args, kwargs = pack_args(1, 2, c=3, d=4)
>>> kwargs
{'c': 3, 'd': 4}
```

```
>>> def pack_args(*args, **kwargs):
...     return args, kwargs
...
>>> args, kwargs = pack_args(1, 2, c=3, d=4)
>>> kwargs
{'c': 3, 'd': 4}
```

```
>>> d1 = {'a': 1, 'b': 2}
>>> d2 = {'b': 3, 'c': 4}
>>> {**d1, **d2}
{'a': 1, 'b': 3, 'c': 4}
```

```
>>> def pack_args(*args, **kwargs):
...     return args, kwargs
...
>>> args, kwargs = pack_args(1, 2, c=3, d=4)
>>> kwargs
{'c': 3, 'd': 4}
```

```
>>> d1 = {'a': 1, 'b': 2}
>>> d2 = {'b': 3, 'c': 4}
>>> {**d1, **d2}
{'a': 1, 'b': 3, 'c': 4}
>>> {**d1, **d2, 'b': 5}
{'a': 1, 'b': 5, 'c': 4}
>>> {'b': 5, **d2, **d1}
{'b': 2, 'c': 4, 'a': 1}
```

```
>>> def pack_args(*args, **kwargs):
...     return args, kwargs
...
>>> d1 = {'a': 1, 'b': 2}
>>> d2 = {'b': 3, 'c': 4}
>>> pack_args(**d1, c=3, d=4)
((), {'a': 1, 'b': 2, 'c': 3, 'd': 4})
```

```
>>> def pack_args(*args, **kwargs):
...     return args, kwargs
...
>>> d1 = {'a': 1, 'b': 2}
>>> d2 = {'b': 3, 'c': 4}
>>> pack_args(**d1, c=3, d=4)
((), {'a': 1, 'b': 2, 'c': 3, 'd': 4})

>>> pack_args(**d1, **d2, b=5)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: __main__.pack_args() got multiple
values for keyword argument 'b'
```

```
>>> def pack_args(*args, **kwargs):
...     return args, kwargs
...
>>> d1 = {'a': 1, 'b': 2}
>>> d2 = {'b': 3, 'c': 4}
>>> pack_args(**d1, c=3, d=4)
((), {'a': 1, 'b': 2, 'c': 3, 'd': 4})

>>> pack_args(**d1, **d2, b=5)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: __main__.pack_args() got multiple
values for keyword argument 'b'

>>> pack_args(**{**d1, **d2, 'b': 5})
((), {'a': 1, 'b': 5, 'c': 4})
```

```
>>> def pack_args(*args, **kwargs):
...     return args, kwargs
...
>>> d1 = {'a': 1, 'b': 2}
>>> d2 = {'b': 3, 'c': 4}
>>> pack_args(**d1, c=3, d=4)
((), {'a': 1, 'b': 2, 'c': 3, 'd': 4})

>>> pack_args(**d1, **d2, b=5)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: __main__.pack_args() got multiple
values for keyword argument 'b'

>>> pack_args(**{**d1, **d2, 'b': 5})
((), {'a': 1, 'b': 5, 'c': 4})

>>> pack_args(**(d1 | d2 | {'b': 5}))
((), {'a': 1, 'b': 5, 'c': 4})
```

girafe
ai

girafe ai

Functional programming elements

Programming paradigms

Programming paradigms

- **Imperative**

Programming paradigms

- **Imperative**

- **Declarative**

girafe ai

Programming paradigms

- **Imperative**
    - Procedural programming
    - Object oriented programming (OOP)
- **Declarative**

Programming paradigms

- **Imperative**
    - Procedural programming
    - Object oriented programming (OOP)
- **Declarative**
    - Functional programming

```python
# calculate sum of squares of even integers from 0 to 10

s = 0
for i in range(10):
    if i % 2 == 0:
        s += i ** 2
```

```python
# calculate sum of squares of even integers from 0 to 10

s = 0
for i in range(10):
    if i % 2 == 0:
        s += i ** 2

s = sum(map(lambda x: x ** 2, filter(lambda x: x % 2 == 0, range(10))))
```

```python
# calculate sum of squares of even integers from 0 to 10

s = 0
for i in range(10):
    if i % 2 == 0:
        s += i ** 2

s = sum(map(lambda x: x ** 2, filter(lambda x: x % 2 == 0, range(10))))

s = sum(i ** 2 for i in range(10) if i % 2 == 0)
```

Functional programming

Functional programming

- Pure functions

Functional programming

- Pure functions
- Higher order functions

Functional programming

- Pure functions
- Higher order functions
- Variables are immutable

Functional programming

- Pure functions
- Higher order functions
- Variables are immutable
- Iterables

Functional programming elements

Functional programming elements

- Iterables

Functional programming elements

- Iterables
- *list*, *set* and *dict* comprehension

Functional programming elements

- Iterables
- *list*, *set* and *dict* comprehension
- Generator expression

Functional programming elements

- Iterables
- *list*, *set* and *dict* comprehension
- Generator expression
- Lambda expression

Functional programming elements

- Iterables
- *list*, *set* and *dict* comprehension
- Generator expression
- Lambda expression
- Built-in functions *map*, *filter*, *enumerate*, *zip*, *sorted*, *any*, *all*

Functional programming elements

- Iterables
- *list*, *set* and *dict* comprehension
- Generator expression
- Lambda expression
- Built-in functions *map*, *filter*, *enumerate*, *zip*, *sorted*, *any*, *all*
- Decorators

Functional Programming Modules

The modules described in this chapter provide functions and classes that support a functional programming style, and general operations on callables.

The following modules are documented in this chapter:

https://docs.python.org/3/library/functional.html

```
>>> a = [1, 2, 3, 4, 5]
```

```
>>> a = [1, 2, 3, 4, 5]

>>> list(map(lambda x: -x, a))
[-1, -2, -3, -4, -5]
```

```
>>> a = [1, 2, 3, 4, 5]

>>> list(map(lambda x: -x, a))
[-1, -2, -3, -4, -5]

>>> import operator
>>> list(map(operator.neg, a))
[-1, -2, -3, -4, -5]
```

```
>>> a = [1, 2, 3, 4, 5]

>>> list(map(lambda x: -x, a))
[-1, -2, -3, -4, -5]

>>> import operator
>>> list(map(operator.neg, a))
[-1, -2, -3, -4, -5]

>>> from operator import neg
>>> list(map(neg, a))
[-1, -2, -3, -4, -5]
```

```
>>> a = [
...       {'name': 'A', 'count': 20},
...       {'name': 'B', 'count': 30},
...       {'name': 'C', 'count': 10},
... ]
```

```
>>> a = [
...     {'name': 'A', 'count': 20},
...     {'name': 'B', 'count': 30},
...     {'name': 'C', 'count': 10},
... ]

>>> sorted(a, key=lambda elem: elem['count'])
[{'name': 'C', 'count': 10}, {'name': 'A',
'count': 20}, {'name': 'B', 'count': 30}]
```

```
>>> a = [
...     {'name': 'A', 'count': 20},
...     {'name': 'B', 'count': 30},
...     {'name': 'C', 'count': 10},
... ]

>>> sorted(a, key=lambda elem: elem['count'])
[{'name': 'C', 'count': 10}, {'name': 'A',
'count': 20}, {'name': 'B', 'count': 30}]

>>> from operator import itemgetter
>>> sorted(a, key=itemgetter('count'))
[{'name': 'C', 'count': 10}, {'name': 'A',
'count': 20}, {'name': 'B', 'count': 30}]
```

```
>>> import operator
>>> l1 = [0, 1, 2]
>>> l2 = [3, 4, 5]
>>> sum(map(operator.mul, l1, l2))
14
```

Operation	Syntax	Function
Addition	`a + b`	`add(a, b)`
Concatenation	`seq1 + seq2`	`concat(seq1, seq2)`
Containment Test	`obj in seq`	`contains(seq, obj)`
Division	`a / b`	`truediv(a, b)`
Division	`a // b`	`floordiv(a, b)`
Bitwise And	`a & b`	`and_(a, b)`
Bitwise Exclusive Or	`a ^ b`	`xor(a, b)`
Bitwise Inversion	`~ a`	`invert(a)`
Bitwise Or	`a | b`	`or_(a, b)`
Exponentiation	`a ** b`	`pow(a, b)`
Identity	`a is b`	`is_(a, b)`
Identity	`a is not b`	`is_not(a, b)`
Indexed Assignment	`obj[k] = v`	`setitem(obj, k, v)`
Indexed Deletion	`del obj[k]`	`delitem(obj, k)`
Indexing	`obj[k]`	`getitem(obj, k)`
Left Shift	`a << b`	`lshift(a, b)`
Modulo	`a % b`	`mod(a, b)`
Multiplication	`a * b`	`mul(a, b)`
Matrix Multiplication	`a @ b`	`matmul(a, b)`

Negation (Arithmetic)	`- a`	`neg(a)`
Negation (Logical)	`not a`	`not_(a)`
Positive	`+ a`	`pos(a)`
Right Shift	`a >> b`	`rshift(a, b)`
Slice Assignment	`seq[i:j] = values`	`setitem(seq, slice(i, j), values)`
Slice Deletion	`del seq[i:j]`	`delitem(seq, slice(i, j))`
Slicing	`seq[i:j]`	`getitem(seq, slice(i, j))`
String Formatting	`s % obj`	`mod(s, obj)`
Subtraction	`a - b`	`sub(a, b)`
Truth Test	`obj`	`truth(obj)`
Ordering	`a < b`	`lt(a, b)`
Ordering	`a <= b`	`le(a, b)`
Equality	`a == b`	`eq(a, b)`
Difference	`a != b`	`ne(a, b)`
Ordering	`a >= b`	`ge(a, b)`
Ordering	`a > b`	`gt(a, b)`

```
a = iadd(a, b)        <=>  a += b.
a = iand(a, b)        <=>  a &= b.
a = iconcat(a, b)     <=>  a += b  # a and b - sequences
a = ifloordiv(a, b)   <=>  a //= b.
a = ilshift(a, b)     <=>  a <<= b.
a = imod(a, b)        <=>  a %= b.
a = imul(a, b)        <=>  a *= b.
a = imatmul(a, b)     <=>  a @= b.
a = ior(a, b)         <=>  a |= b.
a = ipow(a, b)        <=>  a **= b.
a = irshift(a, b)     <=>  a >>= b.
a = isub(a, b)        <=>  a -= b.
a = itruediv(a, b)    <=>  a /= b.
a = ixor(a, b)        <=>  a ^= b.
```

```
operator.attrgetter(attr)
operator.attrgetter(*attrs)

operator.itemgetter(item)
operator.itemgetter(*items)

operator.methodcaller(name, /, *args, **kwargs)
```

```
getattr(object, name[, default])
hasattr(object, name)
setattr(object, name, value)
delattr(object, name)
```

```
getattr(object, name[, default])
hasattr(object, name)
setattr(object, name, value)
delattr(object, name)

>>> getattr(x, name)
>>> x.name
```

```
getattr(object, name[, default])
hasattr(object, name)
setattr(object, name, value)
delattr(object, name)

>>> getattr(x, name)
>>> x.name

>>> setattr(x, name, value)
>>> x.name = value
```

```
getattr(object, name[, default])
hasattr(object, name)
setattr(object, name, value)
delattr(object, name)

>>> getattr(x, name)
>>> x.name

>>> setattr(x, name, value)
>>> x.name = value

>>> delattr(x, name)
>>> del x.name
```

```
>>> a = list(range(100))

>>> a[1:10:2]
[1, 3, 5, 7, 9]
```

```
>>> a = list(range(100))

>>> a[1:10:2]
[1, 3, 5, 7, 9]

>>> a[slice(1, 10, 2)]
[1, 3, 5, 7, 9]
```

functools

functools.wraps

```
>>> from functools import wraps
>>> def my_decorator(f):
...     @wraps(f)
...     def wrapper(*args, **kwargs):
...         print('Calling decorated function')
...         return f(*args, **kwargs)
...     return wrapper
...
>>> @my_decorator
... def example():
...     """Docstring"""
...     print('Called example function')
...
>>> example()
Calling decorated function
Called example function
>>> example.__name__
'example'
>>> example.__doc__
'Docstring'
```

functools.cache

```
from functools import cache

@cache
def fib(n):
    if n < 2:
        return n
    return fib(n-1) + fib(n-2)

>>> [fib(n) for n in range(16)]
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610]
```

functools.lru_cache

```
from functools import lru_cache

@lru_cache(maxsize=10)
def fib(n):
    if n < 2:
        return n
    return fib(n-1) + fib(n-2)

>>> [fib(n) for n in range(16)]
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610]
```

functools.lru_cache

```
from functools import lru_cache

@lru_cache  # @lru_cache(maxsize=128)
def fib(n):
    if n < 2:
        return n
    return fib(n-1) + fib(n-2)

>>> [fib(n) for n in range(16)]
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610]
```

functools.lru_cache

```
from functools import lru_cache

@lru_cache(maxsize=None)  # @cache
def fib(n):
    if n < 2:
        return n
    return fib(n-1) + fib(n-2)

>>> [fib(n) for n in range(16)]
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610]
```

functools.reduce

```
functools.reduce(function, iterable[, initializer])
```

```
>>> from functools import reduce
```

```
>>> reduce(lambda x, y: x+y, [1, 2, 3, 4, 5])
```

```
>>> ((((1+2)+3)+4)+5)
```

functools.partial

```
functools.partial(func, /, *args, **keywords)

>>> from functools import partial

>>> def pack_args(*args, **kwargs):
...     return args, kwargs
...
>>> pack_extra_args = partial(pack_args, 1, 2, a='a', b='b')

>>> pack_extra_args(3, 4, c='c', d='d')
((1, 2, 3, 4), {'a': 'a', 'b': 'b', 'c': 'c', 'd': 'd'})
```

functools

- update_wrapper
- wraps
- total_ordering
- cache
- cmp_to_key
- lru_cache
- reduce
- partial
- partialmethod
- singledispatch
- singledispatchmethod
- cached_property

itertools

itertools

Infinite iterators:

Iterator	Arguments	Results	Example
`count()`	start, [step]	start, start+step, start+2*step, …	`count(10) --> 10 11 12 13 14 ...`
`cycle()`	p	p0, p1, … plast, p0, p1, …	`cycle('ABCD') --> A B C D A B C D ...`
`repeat()`	elem [,n]	elem, elem, elem, … endlessly or up to n times	`repeat(10, 3) --> 10 10 10`

Iterators terminating on the shortest input sequence:

Iterator	Argu-ments	Results	Example
`accumulate()`	p [,func]	p0, p0+p1, p0+p1+p2, …	`accumulate([1,2,3,4,5]) --> 1 3 6 10 15`
`chain()`	p, q, …	p0, p1, … plast, q0, q1, …	`chain('ABC', 'DEF') --> A B C D E F`
`chain.from_iterable()`	iterable	p0, p1, … plast, q0, q1, …	`chain.from_iterable(['ABC', 'DEF']) --> A B C D E F`
`compress()`	data, selectors	(d[0] if s[0]), (d[1] if s[1]), …	`compress('ABCDEF', [1,0,1,0,1,1]) --> A C E F`
`dropwhile()`	pred, seq	seq[n], seq[n+1], start-ing when pred fails	`dropwhile(lambda x: x<5, [1,4,6,4,1]) --> 6 4 1`
`filterfalse()`	pred, seq	elements of seq where pred(elem) is false	`filterfalse(lambda x: x%2, range(10)) --> 0 2 4 6 8`
`groupby()`	iterable[, key]	sub-iterators grouped by value of key(v)	
`islice()`	seq, [start,] stop [, step]	elements from seq[start:stop:step]	`islice('ABCDEFG', 2, None) --> C D E F G`
`pairwise()`	iterable	(p[0], p[1]), (p[1], p[2])	`pairwise('ABCDEFG') --> AB BC CD DE EF FG`
`starmap()`	func, seq	func(*seq[0]), func(*seq[1]), …	`starmap(pow, [(2,5), (3,2), (10,3)]) --> 32 9 1000`
`takewhile()`	pred, seq	seq[0], seq[1], until pred fails	`takewhile(lambda x: x<5, [1,4,6,4,1]) --> 1 4`
`tee()`	it, n	it1, it2, … itn splits one iterator into n	
`zip_longest()`	p, q, …	(p[0], q[0]), (p[1], q[1]), …	`zip_longest('ABCD', 'xy', fillvalue='-') --> Ax By C- D-`

Combinatoric iterators:

Iterator	Arguments	Results
`product()`	p, q, … [repeat=1]	cartesian product, equivalent to a nested for-loop
`permutations()`	p[, r]	r-length tuples, all possible orderings, no repeated elements
`combinations()`	p, r	r-length tuples, in sorted order, no repeated elements
`combinations_with_replacement()`	p, r	r-length tuples, in sorted order, with repeated elements

Examples	Results
`product('ABCD', repeat=2)`	`AA AB AC AD BA BB BC BD CA CB CC CD DA DB DC DD`
`permutations('ABCD', 2)`	`AB AC AD BA BC BD CA CB CD DA DB DC`
`combinations('ABCD', 2)`	`AB AC AD BC BD CD`
`combinations_with_replacement('ABCD', 2)`	`AA AB AC AD BB BC BD CC CD DD`

itertools

```
def take(n, iterable):
    "Return first n items of the iterable as a list"
    return list(islice(iterable, n))
```

itertools

```
def take(n, iterable):
    "Return first n items of the iterable as a list"
    return list(islice(iterable, n))


def prepend(value, iterator):
    "Prepend a single value in front of an iterator"
    # prepend(1, [2, 3, 4]) -> 1 2 3 4
    return chain([value], iterator)
```

itertools

```
def take(n, iterable):
    "Return first n items of the iterable as a list"
    return list(islice(iterable, n))


def prepend(value, iterator):
    "Prepend a single value in front of an iterator"
    # prepend(1, [2, 3, 4]) -> 1 2 3 4
    return chain([value], iterator)
```

```
def ncycles(iterable, n):
    "Returns the sequence elements n times"
    return chain.from_iterable(repeat(tuple(iterable), n))
```

itertools

```python
def take(n, iterable):
    "Return first n items of the iterable as a list"
    return list(islice(iterable, n))


def prepend(value, iterator):
    "Prepend a single value in front of an iterator"
    # prepend(1, [2, 3, 4]) -> 1 2 3 4
    return chain([value], iterator)
```

```python
def ncycles(iterable, n):
    "Returns the sequence elements n times"
    return chain.from_iterable(repeat(tuple(iterable), n))


def partition(pred, iterable):
    "Use a predicate to partition entries into false entries and true entries"
    # partition(is_odd, range(10)) --> 0 2 4 6 8   and  1 3 5 7 9
    t1, t2 = tee(iterable)
    return filterfalse(pred, t1), filter(pred, t2)
```

itertools

```python
def take(n, iterable):
    "Return first n items of the iterable as a list"
    return list(islice(iterable, n))


def prepend(value, iterator):
    "Prepend a single value in front of an iterator"
    # prepend(1, [2, 3, 4]) -> 1 2 3 4
    return chain([value], iterator)
```

```python
def ncycles(iterable, n):
    "Returns the sequence elements n times"
    return chain.from_iterable(repeat(tuple(iterable), n))


def partition(pred, iterable):
    "Use a predicate to partition entries into false entries and true entries"
    # partition(is_odd, range(10)) --> 0 2 4 6 8   and  1 3 5 7 9
    t1, t2 = tee(iterable)
    return filterfalse(pred, t1), filter(pred, t2)
```

https://docs.python.org/3/library/itertools.html#itertools-recipes

What to read about functional programming in python?

What to read about functional programming in python?

Functional Programming HOWTO:

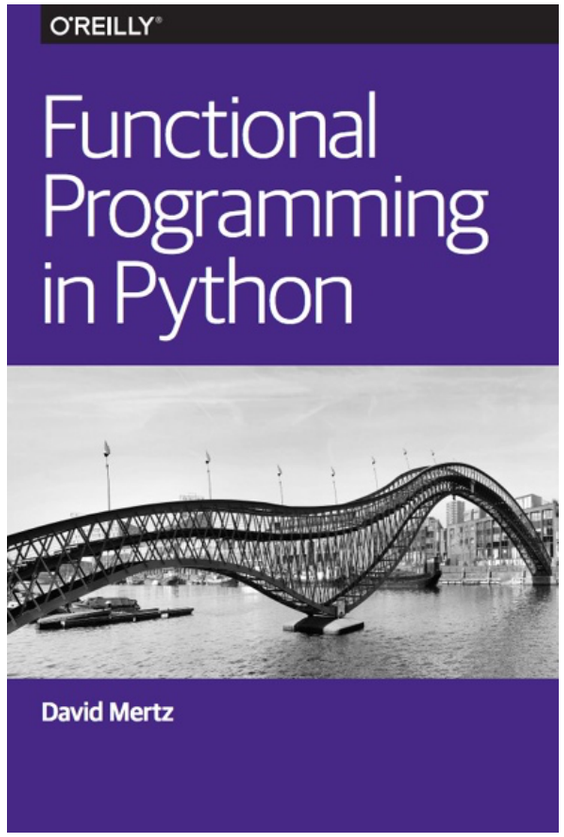
https://docs.python.org/3/howto/functional.html

What to read about functional programming in python?

Functional Programming HOWTO:

https://docs.python.org/3/howto/functional.html

Functional Programming in Python by David Mertz:



girafe
ai

Jupyter notebooks

girafe ai

jupyter Welcome to P

File Edit View Insert Cell

jupyter

Welcome to the

This Notebook Server wa

WARNING

Don't rely on this serv

Your server is hosted thar

Run some Python

To run the code below:

1. Click on the cell to se
2. Press SHIFT+ENTER

A full tutorial for using the

```
In [ ]: %matplotlib inline

        import pandas as pd
        import numpy as np
        import matplotlib
```

jupyter Lorenz Differential Equations (autosaved)

File Edit View Insert Cell Kernel Help Python 3

Code Cell Toolbar: None

Exploring the Lorenz System

In this Notebook we explore the Lorenz system of differential equations:

$$\dot{x} = \sigma(y - x)$$
$$\dot{y} = \rho x - y - xz$$
$$\dot{z} = -\beta z + xy$$

This is one of the classic systems in non-linear differential equations. It exhibits a range of complex behaviors as the parameters (σ, β, ρ) are varied, including what are known as *chaotic solutions*. The system was originally developed as a simplified mathematical model for atmospheric convection in 1963.

```
In [7]: interact(Lorenz, N=fixed(10), angle=(0.,360.),
                σ=(0.0,50.0),β=(0.,5), ρ=(0.0,50.0))
```

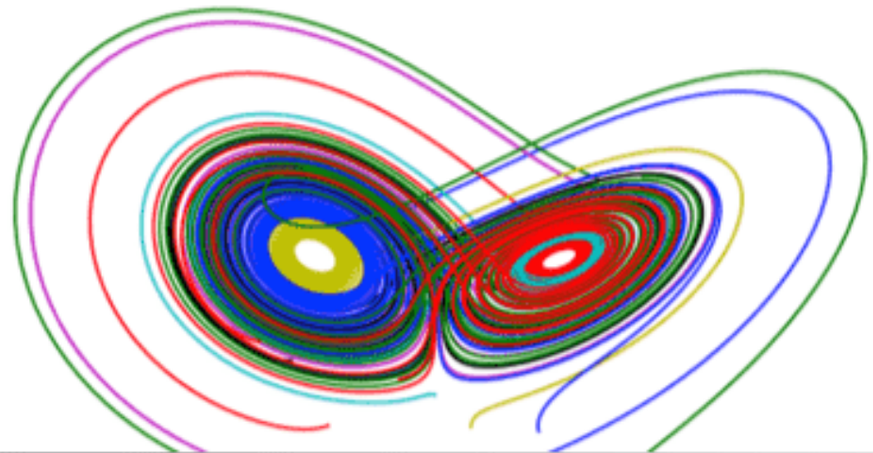
angle 308.2

max_time 12

σ 10

β 2.6

ρ 28



Links

- https://jupyter.org

Links

- https://jupyter.org
- https://nbviewer.org/

Links

- https://jupyter.org
- https://nbviewer.org/
- https://www.anaconda.com/products/individual

Individual Edition

Your data science toolkit

With over 25 million users worldwide, the open-source Individual Edition (Distribution) is the easiest way to perform Python/R data science and machine learning on a single machine. Developed for solo practitioners, it is the toolkit that equips you to work with thousands of open-source packages and libraries.

Anaconda Individual Edition

Download

For MacOS

Python 3.8 • 64-Bit Graphical Installer • 440 MB

Get Additional Installers

Links

- https://jupyter.org
- https://nbviewer.org/
- https://www.anaconda.com/products/individual

Links

- https://jupyter.org
- https://nbviewer.org/
- https://www.anaconda.com/products/individual
- https://docs.conda.io/en/latest/miniconda.html

Links

- https://jupyter.org
- https://nbviewer.org/
- https://www.anaconda.com/products/individual
- https://docs.conda.io/en/latest/miniconda.html
- https://colab.research.google.com

Welcome To Colaboratory

File Edit View Insert Runtime Tools Help

Table of contents

Getting started

Data science

Machine learning

More Resources

Machine Learning Examples

Section

+ Code + Text Copy to Drive

What is Colaboratory?

Colaboratory, or "Colab" for short, allows you to write and execute Python in your browser, with

- Zero configuration required
- Free access to GPUs
- Easy sharing

Whether you're a **student**, a **data scientist** or an **AI researcher**, Colab can make your work easier. Watch Introduction to Colab to learn more, or just get started below!

Getting started

The document you are reading is not a static web page, but an interactive environment called a **Colab notebook** that lets you write and execute code.

For example, here is a **code cell** with a short Python script that computes a value, stores it in a variable, and prints the result:

```
[ ]  seconds_in_a_day = 24 * 60 * 60
     seconds_in_a_day
```

```
86400
```

To execute the code in the above cell, select it with a click and then either press the play button to the left of the code, or use the keyboard shortcut "Command/Ctrl+Enter". To edit the code, just click the cell and start editing.

Variables that you define in one cell can later be used in other cells:

```
[ ]  seconds_in_a_week = 7 * seconds_in_a_day
     seconds_in_a_week
```

```
604800
```

Links

- https://jupyter.org
- https://nbviewer.org/
- https://www.anaconda.com/products/individual
- https://docs.conda.io/en/latest/miniconda.html
- https://colab.research.google.com

girafe
ai