streprogen Documentation

Release 1.0.0

Tommy O.

Sep 27, 2017

Contents

1 Project summary **3**

2 Installation **5**

3 Sample code **7**

4 Contents **9**

4.1 Getting started 9

4.2 Advanced usage 12

4.3 API: Classes 16

4.4 API: Functions 23

5 Indices and tables **27**

Python Module Index **29**

Welcome to the documentation for streprogen, the Python strength program generator.

CHAPTER 1

Project summary

Streprogen (short for **Stre**ngth **Pro**gram **Gen**erator) is a Python package which allows the user to easily create dynamic, flexible strength training programs. The main features are:

- **Sensible defaults**: The software comes with sensible default values for all input parameters, giving the novice strength athlete some guidance on parameter selection. The software will raise warnings if the input parameters are unreasonable, but will still run.
- **High level of customization**: Every important parameter can be changed by the user. It is possible to create long-term training programs with several layers of periodization if the user wishes to do so.
- **Simple object oriented interface**: The four main classes are `StaticExercise`, `DynamicExercise`, `Day` and `Program`.
- **Pretty output**: The training programs are easily saved as `.txt`, `.html` or `.tex` files.

CHAPTER 2

Installation

1. The Anaconda distribution of Python 3.x from the Anaconda Website.
2. Run `pip install streprogen` in the terminal (cmd.exe on Windows) to install streprogen from PyPi.
3. Open a Python Editor (such as Spyder, which comes with Anaconda).
4. Look at the tutorial.

CHAPTER 3

Sample code

```
from streprogen import Program, Day, DynamicExercise, StaticExercise

# Create a 4-week program
program = Program('My first program!', duration = 4)

# Create some dynamic and static exercises
bench = DynamicExercise('Bench press', 60, 80)
squats = DynamicExercise('Squats', 80, 95)
curls = StaticExercise('Curls', '3 x 12')
day = Day(exercises = [bench, squats, curls])

# Add day(s) to program and render it
program.add_days(day)
program.render()
print(program)
```

```
------------------------------------------------------------------
Program: My first program!

Program parameters
  duration: 4
  reps_per_exercise: 25
  avg_intensity: 75
  reps_scalers: 1.2, 1, 0.8, 1
  intensity_scalers: 0.9, 1, 1, 1
  units: kg
------------------------------------------------------------------
Exercise information
  Day 1
   Bench press    60kg -> 80kg     reps: [3, 8]   weekly inc.: 7.5%
   Squats         80kg -> 95kg     reps: [3, 8]   weekly inc.: 4.4%
   Curls          3 x 12
------------------------------------------------------------------
Program
 Week 1
```

```
 Day 1
  Bench press   6 x 45kg    7 x 42.5kg  7 x 42.5kg  7 x 42.5kg
  Squats        6 x 60kg    7 x 57.5kg  8 x 52.5kg  8 x 52.5kg
  Curls         3 x 12

Week 2
 Day 1
  Bench press   4 x 55kg    5 x 52.5kg  5 x 52.5kg  5 x 52.5kg  6 x 50kg
  Squats        4 x 70kg    4 x 70kg    5 x 65kg    6 x 62.5kg  7 x 57.5kg
  Curls         3 x 12

Week 3
 Day 1
  Bench press   6 x 55kg    6 x 55kg    6 x 55kg
  Squats        6 x 67.5kg  6 x 67.5kg  6 x 67.5kg
  Curls         3 x 12

Week 4
 Day 1
  Bench press   5 x 62.5kg  6 x 60kg    6 x 60kg    7 x 57.5kg
  Squats        5 x 75kg    6 x 70kg    6 x 70kg    7 x 67.5kg
  Curls         3 x 12

------------------------------------------------------------
```

CHAPTER 4

Contents

Getting started

This tutorial was written using a Jupyter Notebook.

Minimal working example

This example shows to to use the *Program*, *StaticExercise* and *Day* classes to create a simple static strength training program. The example may not look impressive, but it shows how to create a working strength training program. The *Program.render()* method is very important, because it populates all the days and weeks in the program with calculations.

```
In [1]: from streprogen import StaticExercise, Day, Program

        # Create a  3 week training program
        program = Program('Minimal program', duration = 3)

        # Create a static exercise to a day
        squats = StaticExercise('Squats', '5 x 5 @ 80kg')
        day = Day(exercises = [squats])

        # Add the day to the program and render it
        program.add_days(day)
        program.render()
        print(program)
```

```
------------------------------------------------------------------
Program: Minimal program

Program parameters
  duration: 3
  reps_per_exercise: 25
  avg_intensity: 75
  reps_scalers: 1, 0.8, 0.8
  intensity_scalers: 1, 1, 0.9
```

```
  units: kg
------------------------------------------------------------
Exercise information 6
  Day 1
   Squats   5 x 5 @ 80kg
------------------------------------------------------------
Program
 Week 1
  Day 1
   Squats   5 x 5 @ 80kg

 Week 2
  Day 1
   Squats   5 x 5 @ 80kg

 Week 3
  Day 1
   Squats   5 x 5 @ 80kg

------------------------------------------------------------
```

Using dynamic exercises

This example introduces the *DynamicExercise* class, and also shows how to save a program as a `.html` file. Three output methods are supported:

- To `txt` with the *to_txt()* method.
- To `html` with the *to_html()* method.
- To `tex` with the *to_tex()* method.

```
In [2]: from streprogen import StaticExercise, DynamicExercise, Day, Program

        # Create a 8 week training program
        program = Program('Program with dynamic exercise', duration = 8)

        # Create a dynamic exercise, with start weight 100, end weight 110
        # and repetitions between 4 and 8 (inclusive)
        squats = DynamicExercise('Squats', 100, 110, min_reps = 4, max_reps = 8)
        biceps = StaticExercise('Biceps', '3 x 12')
        day = Day(exercises = [squats, biceps])

        # Add the day to the program and render it
        program.add_days(day)
        program.render()

        # Save the program as a HTML file
        with open('program_with_dynamic_ex.html', 'w', encoding = 'utf-8') as file:
            # The table width can be controlled by passing the 'table_width' argument
            file.write(program.to_html(table_width = 8))
```

The output file generated by the code above is:

- program_with_dynamic_ex.html

Several days

This example introduces several new features:

- Controlling repetitions per exercise using `reps_per_exercise`.
- Controlling the average intensity (% of maximum weight) using `avg_intensity`.
- Controlling the rounding globally with `round_to`.

```
In [3]:
        from streprogen import StaticExercise, DynamicExercise, Day, Program
        # Create a 6 week training program with 20 reps per exercise
        program = Program('Program with dynamic exercise', duration = 8, reps_per_exerci


        # Create the first day
        squats = DynamicExercise('Squats', 100, 120, min_reps = 4, max_reps = 8)
        bench = DynamicExercise('Bench press', 80, 95, min_reps = 4, max_reps = 8)
        dayA = Day('Day A', exercises = [squats, bench])

        # Create the second day
        squats = DynamicExercise('Squats', 100, 110, min_reps = 4, max_reps = 8)
        deadlifts = DynamicExercise('Deadlifts', 120, 135, min_reps = 4, max_reps = 8)
        dayB = Day('Day B', exercises = [squats, bench])

        # Add the day to the program and render it
        program.add_days(dayA, dayB)
        program.render()

        # Save a .html file
        with open('program__with_several_days.html', 'w', encoding = 'utf-8') as file:
            # The table width can be controlled by passing the 'table_width' argument
            file.write(program.to_html(table_width = 8))
```

The output file generated by the code above is:

- program__with_several_days.html

A realistic program

Here is a realistic program that was used in real life. It's a three-week, full body program. A function (named `f` in the code below) was used to set the *end_weight* parameter. The *StaticExercise* class can also take a function (of one parameter, the current week) as input.

```
In [4]: from streprogen import StaticExercise, DynamicExercise, Day, Program
        import subprocess # Used to run pdflatex

        # Create a function to map from start weights to end weights
        def f(initial):
            # Function to return final weight,
            # increasing the weights by 2% per day
            return int(initial*1.02**duration)

        # Create a function for the static exercise
        def dips_scheme(week):
            if week <= 4:
                return '4 x 10 @ bodyweight'
            else:
```

```
        return '4 x 12 @ bodyweight + 10kg'

# Create the program
duration = 8
program = Program('A realistic program', units='', round_to=2.5)

# The first day
day1 = Day('Monday')
squats = DynamicExercise('Squats',         95,  f(95))
chins  = DynamicExercise('Chins (light)', 100, f(100))
press  = DynamicExercise('Military press',50,  f(50))
day1.add_exercises(squats, chins, press)

# The second day
day2 = Day('Wednesday')
deadlifts    = DynamicExercise('Deadlifts', 120, f(120))
bench_press  = DynamicExercise('Bench',      70,  f(70))
chin_ups     = DynamicExercise('Chin ups',   100, f(100))
dips = StaticExercise('Dips', dips_scheme) # Notice that a function is used here
day2.add_exercises(deadlifts, bench_press, chin_ups, dips)

# The third day
day3 = Day('Friday')
squats = DynamicExercise('Squats',         85, f(85))
bench  = DynamicExercise('Bench (light)', 85, f(85))
rows   = DynamicExercise('Rows',          65, f(85))
day3.add_exercises(squats, chins, press)

# Add the days and render the program
program.add_days(day1, day2, day3)
program.render()

# Save a .html file
with open('realistic_program.html', 'w', encoding = 'utf-8') as file:
    file.write(program.to_html(table_width = 6))

# Save a .tex file
with open('realistic_program.tex', 'w', encoding = 'utf-8') as file:
    file.write(program.to_tex(table_width = 8))

# Use pdflatex to create a .pdf from the .tex file
ret = subprocess.call(['pdflatex', 'realistic_program.tex'], shell=False)
```

The output file generated by the code above is:

- realistic_program.html
- realistic_program.tex
- realistic_program.pdf

Advanced usage

This tutorial was written using a Jupyter Notebook.

Start by importing some stuff used by the Jupyter Notebook.

Examine the available rep to intensity mappings

The *Program* class has an input parameter called `reps_to_intensity_func`. This can be set to whatever the user wishes (but warnings and errors might pop up if it is not a sufficiently 'nice' function).

Let us look at the mapping between repetitions and intensity. Three mappings are available:

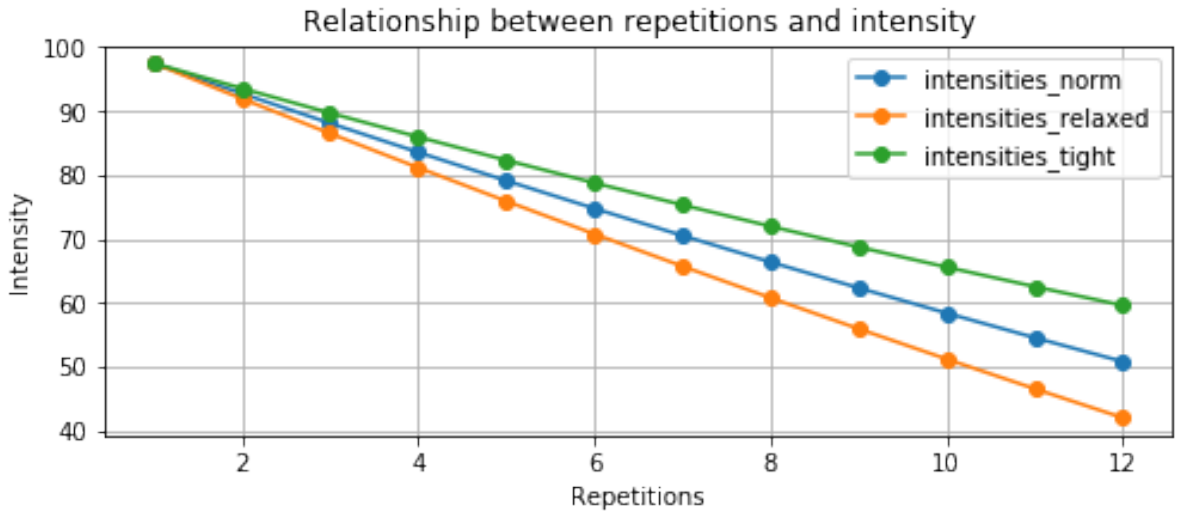
- *reps_to_intensity* - The default map.
- `reps_to_intensity_relaxed` - A more 'relaxed' mapping - many reps is not as heavy any more.
- `reps_to_intensity_tight` - A more 'tight' mapping - many reps is heavier.

```
In [1]: import matplotlib.pyplot as plt
        from streprogen import reps_to_intensity, reps_to_intensity_relaxed, reps_to_int

        # Set up repetitions and apply all three mappings
        reps = list(range(1, 12 + 1))
        intensities_norm = list(map(reps_to_intensity, reps))
        intensities_relaxed = list(map(reps_to_intensity_relaxed, reps))
        intensities_tight = list(map(reps_to_intensity_tight, reps))
```

Plotting the rep to intensity mappings

```
In [2]: plt.figure(figsize = (8, 3))
        plt.title('Relationship between repetitions and intensity')
        plt.plot(reps, intensities_norm, '-o', label = 'intensities_norm')
        plt.plot(reps, intensities_relaxed, '-o', label = 'intensities_relaxed')
        plt.plot(reps, intensities_tight, '-o', label = 'intensities_tight')
        plt.ylabel('Intensity')
        plt.xlabel('Repetitions')
        plt.legend(loc = 'best')
        plt.grid(True)
        plt.show()
```



Plotting the rep to intensity mappings

```
In [3]: table_width = 6
        print('reps'.ljust(8),*[str(i).ljust(table_width) for i in reps])
        print('-'*90)
        print('norm'.ljust(8), *[str(round(i)).ljust(table_width) for i in intensities_r
        print('relaxed'.ljust(8),*[str(round(i)).ljust(table_width) for i in intensities
        print('tight'.ljust(8),*[str(round(i)).ljust(table_width) for i in intensities_t
```

```
reps     1      2      3      4      5      6      7      8      9      10     11     12
------------------------------------------------------------------------------------------
norm     98     93     88     84     79     75     70     66     62     58     54     51
relaxed  98     92     86     81     76     71     66     61     56     51     46     42
tight    98     94     90     86     82     79     75     72     69     66     62     60
```

Creating a new rep to intensity mapping

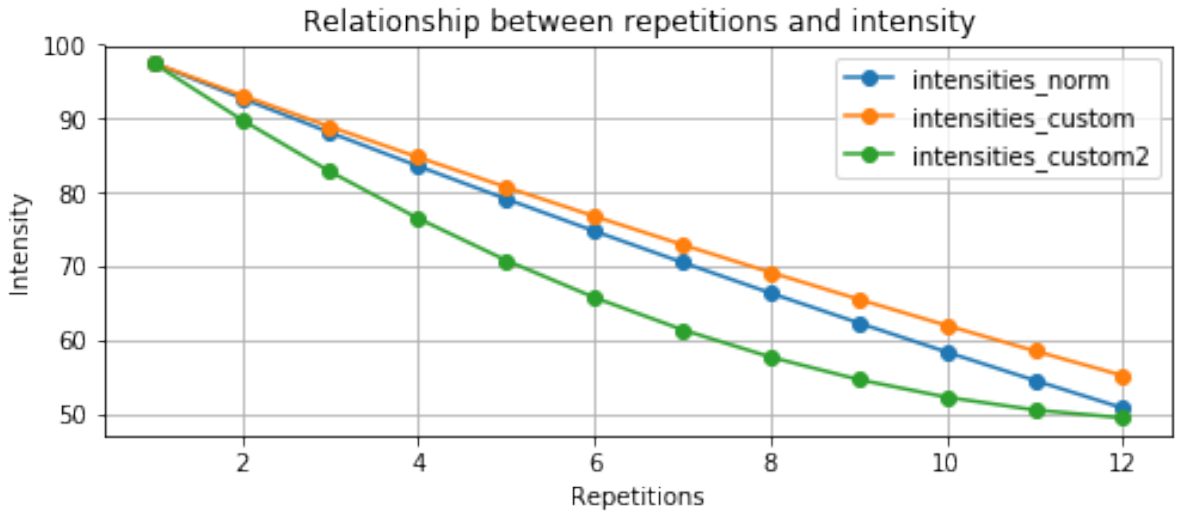
```
In [4]: from functools import partial

        # Method 1: Using a partial function
        custom_set_intensity = partial(reps_to_intensity, slope=-4.4, constant=97.5)
        intensities_custom = list(map(custom_set_intensity, reps))

        # Method 2: Custom function from scratch
        def custom_set_intensity(reps):
            return 97.5 - 8 *(reps - 1) + 0.33*(reps - 1)**2

        intensities_custom2 = list(map(custom_set_intensity, reps))
```

```
In [5]: plt.figure(figsize = (8, 3))
        plt.title('Relationship between repetitions and intensity')
        plt.plot(reps, intensities_norm, '-o', label = 'intensities_norm')
        plt.plot(reps, intensities_custom, '-o', label = 'intensities_custom')
        plt.plot(reps, intensities_custom2, '-o', label = 'intensities_custom2')
        plt.ylabel('Intensity')
        plt.xlabel('Repetitions')
        plt.legend(loc = 'best')
        plt.grid(True)
        plt.show()
```



Examine the available progression models

The *Program* class has an input parameter called `progress_func`. It defaults to *progression_sinusoidal()*, but *progression_linear()* is also available. Partial functions based off *progression_sinusoidal()* can be used, or the user can define their own function, but it must have a signature like `progression_custom(week, start_weight, end_weight, start_week, end_week)`.

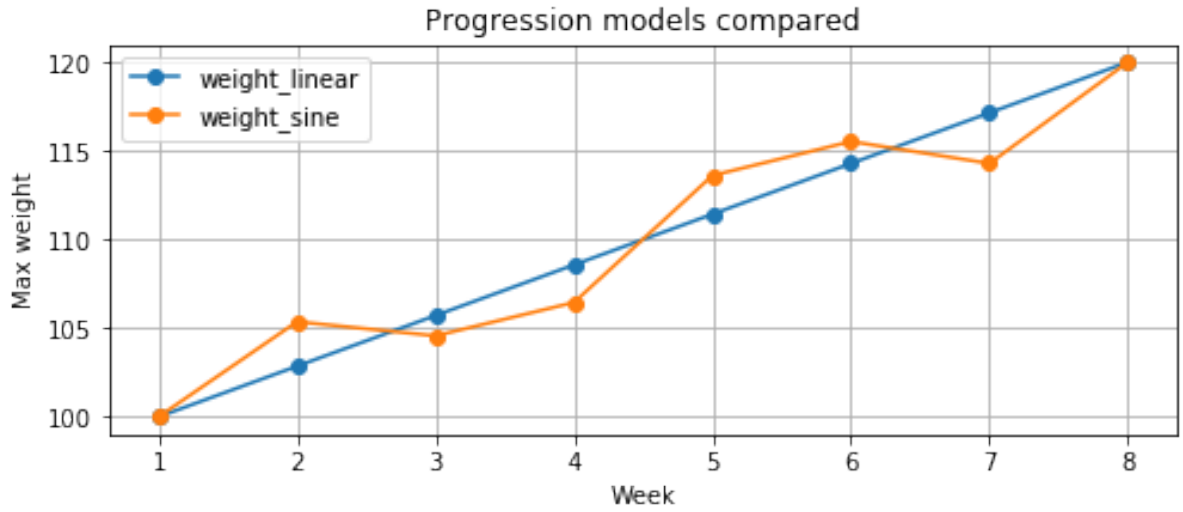
```
In [6]: from streprogen import progression_linear, progression_sinusoidal

        # Set up some constants
        duration = 8
        start, end = 100, 120

        # Create lists
        weeks = list(range(1, duration + 1))
        weight_linear = [progression_linear(week, start, end, 1, duration) for week in w
        weight_sine = [progression_sinusoidal(week, start, end, 1, duration) for week in
```

A plot of the available progression models

```
In [7]: plt.figure(figsize = (8, 3))
        plt.title('Progression models compared')
        plt.plot(weeks, weight_linear, '-o', label = 'weight_linear')
        plt.plot(weeks, weight_sine, '-o', label = 'weight_sine')
        plt.ylabel('Max weight')
        plt.xlabel('Week')
        plt.legend(loc = 'best')
        plt.grid(True)
        plt.show()
```



Scale reps and intensities

The *Program* class has input parameters `rep_scalers` and `intensity_scalers`. By default a *RepellentGenerator* is created and the *RepellentGenerator().yield_from_domain()* method is used to generate a list of factors to scale repetitions

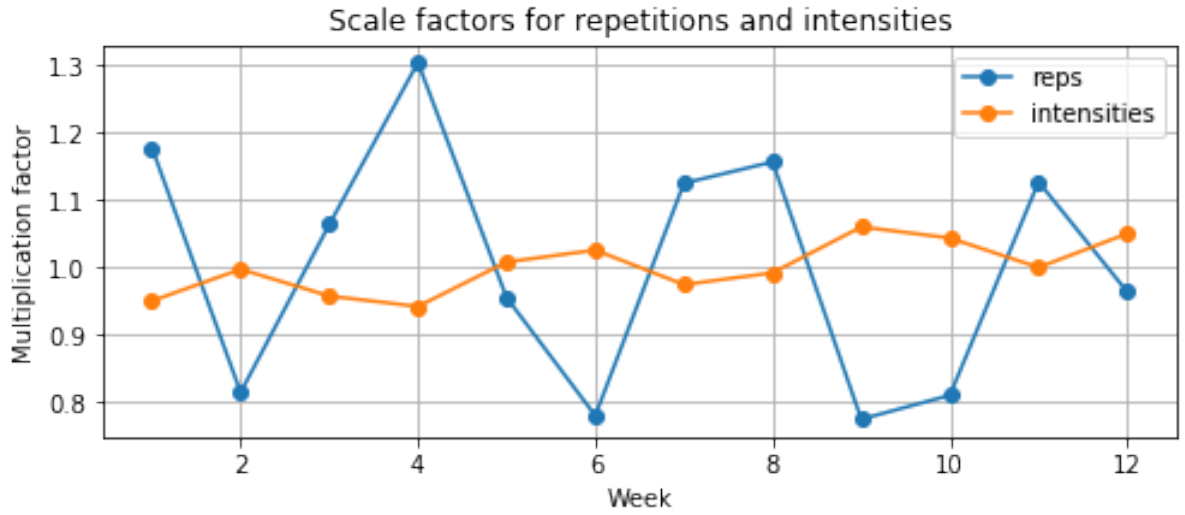
and itensities. The user can define their own list of factors too, as shown below using the *progression_sinusoidal()* function.

```
In [8]: duration = 12

        # A function to create scalers for the repetitions and itensity
        reps      = partial(progression_sinusoidal, start_weight = 1.1,  end_weight = 0.
        intensity = partial(progression_sinusoidal, start_weight = 0.95, end_weight = 1.

        # Create lists
        weeks = list(range(1, duration + 1))
        intensities = list(map(intensity, weeks))
        reps = list(map(reps, weeks))

In [9]: plt.figure(figsize = (8, 3))
        plt.title('Scale factors for repetitions and intensities')
        plt.plot(weeks, reps, 'o-', label = 'reps')
        plt.plot(weeks, intensities, 'o-', label = 'intensities')
        plt.ylabel('Multiplication factor')
        plt.xlabel('Week')
        plt.legend(loc = 'best')
        plt.grid(True)
        plt.show()
```



API: Classes

Brief introduction to classes

There are four classes available:

- `StaticExercise`: For exercises schemes such as “3 x 12”, “5 x 5 @ 80kg” or “stretch for 5 mins”. In other words, this class is merely a container for an exercise name and a string.
- `DynamicExercise`: For exercises where you wish to render a dynamic set/rep scheme. The `DynamicExercise` class is part of what makes streprogen dynamic.
- `Day`: A `Day` class is a container for exercises associated with the same day.

- `Program`: This is where the magic happens. The `Program` class is a container for `Day``s (and therefore also instances of ``StaticExercise` and `DynamicExercise`). The algorithms used to render the program is also contained in the `Program` class. The most important method is the `Program.render()` method, which renders the dynamic exercises.

The DynamicExercise class

class `streprogen.`**`DynamicExercise`**(*name*, *start_weight*, *end_weight*, *min_reps=3*, *max_reps=8*, *reps=None*, *avg_intensity=None*, *round_to=None*)

Class for dynamic exercises.

`__init__`(*name*, *start_weight*, *end_weight*, *min_reps=3*, *max_reps=8*, *reps=None*, *avg_intensity=None*, *round_to=None*)

Initialize a new dynamic exercise. A dynamic exercise is rendered by the program, and the set/rep scheme will vary from week to week.

Parameters

- **`name`** – The name of the exercise, e.g. 'Squats'.
- **`start_weight`** – Maximum weight you can lift at the start of the program, e.g. 80.
- **`end_weight`** – The goal weight to work towards during the program. This should be set in relation to the duration of the training program, e.g. 90.
- **`min_reps`** – The minimum number of repetitions for this exercise, e.g. 3.
- **`max_reps`** – The maximum number of repetitions for this exercise, e.g. 8.
- **`reps`** – The number of baseline repetitions for this exercise. If this parameter is set, it will override the global 'reps_per_exercise' parameter for the training program. The repetitions will still be scaled by the 'reps_scalers' parameter in the training program.
- **`avg_intensity`** – The average intensity for this exercise. If set, this will override the 'avg_intensity' parameter in the training program. The intensity will still be scaled by the 'intensity_scalers' parameter.
- **`round_to`** – Round the output to the closest multiple of this number, e.g. 2.5.

Returns A DynamicExercise object.

Return type *DynamicExercise*

Examples

```
>>> bench = DynamicExercise('Bench press', 100, 120, 3, 8)
```

`weekly_growth`(*weeks*)

Calculate the weekly growth in percentage, and rounds to one digit.

Parameters **`weeks`** – Number of weeks to calculate growth over.

Returns A real number such that start * growth_factor** weeks = end.

Return type growth_factor

Examples

```
>>> bench = DynamicExercise('Bench press', 100, 120, 3, 8)
>>> bench.weekly_growth(8)
2.3
>>> bench.weekly_growth(4)
4.7
```

The StaticExercise class

class streprogen.**StaticExercise**(*name*, *sets_reps='4 x 10'*)

Class for static exercises.

__init__(*name*, *sets_reps='4 x 10'*)

Initialize a new static exercise. A static exercise is simply a placeholder for some text.

Parameters

- **name** – The name of the exercise, e.g. 'Curls'.
- **sets_reps** – A static set/rep scheme, e.g. '4 x 10', or '10 minutes'. This paramter can also be a function of one parameter, the current week. The function must return a string for that specific week.

Returns A StaticExercise object.

Return type *StaticExercise*

Examples

```
>>> curls = StaticExercise('Curls', '4 x 10')
>>> stretching = StaticExercise('Stretching', '10 minutes')
```

The Day class

class streprogen.**Day**(*name=None*, *exercises=None*)

A day object is a container for exercises associated with the specific day.

__init__(*name=None*, *exercises=None*)

Initialize a new day object.

Parameters

- **name** – The name of the day, e.g. 'Day A'. If no name is given then the day will automatically be given a numeric name such as 'Day 1', 'Day 2', etc.
- **exercises** – A list of exercises. Exercises can also be associated with a day using the 'add_exercises' method later on.

Returns A day object.

Return type *Day*

Examples

```
>>> monday = Day(name = 'Monday')
>>> curls = StaticExercise('Curls', '3 x 12')
>>> monday.add_exercises(curls)
>>> curls in monday.static_exercises
True
```

add_exercises(**exercises*)

Add the exercises to the day. The method will automatically infer whether a static or dynamic exercise is passed to it.

Parameters ***exercises** – An unpacked tuple of exercises.

Examples

```
>>> monday = Day(name = 'Monday')
>>> curls = StaticExercise('Curls', '3 x 12')
>>> pulldowns = StaticExercise('Pulldowns', '4 x 10')
>>> monday.add_exercises(curls, pulldowns)
>>> curls in monday.static_exercises
True
>>> pulldowns in monday.static_exercises
True
```

The Program class

class streprogen.**Program**(*name='Untitled'*, *duration=8*, *reps_per_exercise=25*, *rep_scalers=None*, *intensity=75*, *intensity_scalers=None*, *units='kg'*, *round_to=2.5*, *progress_func=None*, *reps_to_intensity_func=None*, *min_reps_consistency=None*, *minimum_percentile=0.2*, *go_to_min=False*, *verbose=False*)

The program class is a container for days and exercises, along with the methods and functions used to create training programs.

__init__(*name='Untitled'*, *duration=8*, *reps_per_exercise=25*, *rep_scalers=None*, *intensity=75*, *intensity_scalers=None*, *units='kg'*, *round_to=2.5*, *progress_func=None*, *reps_to_intensity_func=None*, *min_reps_consistency=None*, *minimum_percentile=0.2*, *go_to_min=False*, *verbose=False*)

Initialize a new program.

Parameters

- **name** – The name of the training program, e.g. 'Tommy_August_2017'.
- **duration** – The duration of the training program in weeks, e.g. 8.

- **reps_per_exercise** – The baseline number of repetitions per dynamic exercise. Typically a value in the range [20, ..., 35].
- **rep_scalers** – A list of factors of length 'duration', e.g. [1, 0.9, 1.1, ...]. For each week, the baseline number of repetitions is multiplied by the corresponding factor, adding variation to the training program. Each factor is typically in the range [0.7, ..., 1.3]. If None, a list of random factors is generated.
- **intensity** – The baseline intensity for each dynamic exercise. The intensity of an exercise for a given week is how heavy the average repetition is compared to the expected 1RM (max weight one can lift) for that given week. Typically a value around 75.
- **intensity_scalers** – A list of factors of length 'duration', e.g. [1, 0.95, 1.05, ...]. For each week, the baseline intensity is multiplied by the corresponding factor, adding variation to the training program. Each factor is typically in the range [0.95, ..., 1.05]. If None, a list of random factors is generated.
- **units** – The units used for exporting and printing the program, e.g. 'kg'.
- **round_to** – Round the dynamic exercise to the nearest multiple of this parameter. Typically 2.5, 5 or 10.
- **progress_func** – The function used to model overall 1RM progression in the training program. If None, the program uses *streprogen.progression_sinusoidal()*. Custom functions may be used, but they must implement arguments like the *streprogen.progression_sinusoidal()* and *streprogen.progression_linear()* functions.
- **reps_to_intensity_func** – The function used to model the relationship between repetitions and intensity. If None, the program uses *streprogen.reps_to_intensity()*. Custom functions may be used, and the functions `streprogen.reps_to_intensity_tight()` and `streprogen.reps_to_intensity_relaxed()` are available.
- **min_reps_consistency** – This is an advanced feature. By default, the program will examine the dynamic exercises and try to set a minimum repetition consistency mode. If all dynamic exercises in the program use the same repetition range, it will be set to 'weekly'. If all dynamic exercises in each day use the same repetition range, it will be set to 'daily'. If neither, it will be set to 'exercise'.

  The minimum reps consistency mode tells the program how often it should draw a new random value for the minimum repetition to work up to. If 'min_reps_consistency' is 'weekly' and the 'go_to_min' parameter is set to True, you can expect that every exercise will work up to the same minimum number of repetitions.

  The 'min_reps_consistency' argument will override the program default. If, for example, every exercise is set to the repetition range 3-8 but you wish to work up to different minimum values, set 'min_reps_consistency' to 'daily' or 'exercise'.

- **minimum_percentile** – This is an advanced feature. To protect the athlete against often working up to heavy weights, the repetition range is "clipped" randomly. A repetition range 1-8 might be clipped to, say, 3-8, 2-8 or 1-8. If clipped to 3-8, the repetitions are drawn from [3, ..., 8] instead of [1, ..., 8].

  The 'minimum_percentile' determines the percentile of the repetition range to clip away. If 0, no clipping occurs. If 0.5, half the repetition range could potentially be clipped away. How often the range is clipped and a new minimum repetition value is computed is determined by the minimum repetition consistency mode, which may be controlled by the 'minimum_percentile' argument.

- **go_to_min** – This is an advanced feature. Whether or not to force the program to work up to the minimum repetition possible for a given dynamic exercise. Consider a program where 'minimum_percentile' is 0.2, and a dynamic exercise has a repetition range 1-8. The program will drawn repetitions in ranges 1-8, 2-8 or 3-8. If 'go_to_min' is True, the program will be forced to work up to 1, 2 or 3 repetitions respectively. If 'go_to_min' is False, the same range will be used, but the program need not go to the minimum number of repeitions.

- **verbose** – If True, information will be outputted as the program is created.

Returns A Program instance.

Return type *Program*

Examples

```
>>> program = Program('My training program')
>>> program._rendered
False
```

add_days(**days*)

Add one or several days to the program.

Parameters ***days** – Unpacked tuple containing *streprogen.Day* instances.

Examples

```
>>> program = Program('My training program')
>>> day1, day2 = Day(), Day()
>>> program.add_days(day1, day2)
```

render(*validate=True*)

Render the training program to perform the calculations. The program can be rendered several times to produce new information given the same input parameters.

Parameters **validate** – Boolean that indicates whether or not to run a validation heurestic on the program before rendering. The validation will warn the user if inputs seem unreasonable.

static **repstring_penalty**(*reps*, *intensities*, *desired_reps*, *desired_intensity*, *minimum_rep*)

Penalty function which calculates how "bad" a set of reps and intensities is, compared to the desired repetitions, the desired intensity level and the minimum repetitions. Advanced users may substitute this function for their own version.

Parameters

- **reps** – A list of repetitions (sorted), e.g. [8, 6, 5, 2].
- **intensities** – A list of intensities corresponding to the repetitions, e.g. [64.7, 72.3, 76.25, 88.7].
- **desired_reps** – Desired number of repetitions in total, e.g. 25.
- **desired_intensity** – The desired average intensity, e.g. 75.
- **minimum_rep** – The minimum repetition which is allowed, e.g. 2.

Returns A penalty, a positive real number.

Return type float

Examples

```
>>> desired_reps = 25
>>> desired_intensity = 75
>>> minimum_rep = 1
>>> high = Program().repstring_penalty([8, 8, 8], [60, 60, 60],
...                                    desired_reps, desired_intensity,
...                                    minimum_rep)
>>> low = Program().repstring_penalty([8, 6, 5, 4, 2], [64, 72,␣
↪75, 80, 88],
...                                    desired_reps, desired_intensity,
...                                    minimum_rep)
>>> high > low
True
```

to_html(*table_width=5*)

Write the program information to HTML code, which can be saved, printed and brought to the gym.

Parameters **table_width** – The table with of the HTML code.

Returns HTML code.

Return type string

to_tex(*text_size='large'*, *table_width=5*)

Write the program information to a .tex file, which can be rendered to .pdf running pdflatex. The program can then be printed and brought to the gym.

Parameters

- **text_size** – The tex text size, e.g. 'small', 'normalsize', 'large', 'Large' or 'LARGE'.
- **table_width** – The table with of the .tex code.

Returns Program as tex.

Return type string

to_txt(*verbose=False*)
Write the program information to text, which can be printed in a terminal.

Parameters **verbose** – If True, more information is shown.

Returns Program as text.

Return type string

API: Functions

Functions documented here.

Functions modeling reps/intensity mapping

reps_to_intensity

streprogen.**reps_to_intensity**(*reps*, *slope=-4.8*, *constant=97.5*, *quadratic=True*)
A function mapping from repetitions in the range 1 to 12 to intensities in the range 0 to 100.

Parameters

- **reps** – The number of repetitions to map to the intensity range.
- **slope** – Slope for the linear function.
- **constant** – Constant for the linear function
- **quadratic** – If 'True', add a slight quadratic offset.

Returns An intensity value in the range from 0 to 100.

Return type intensity

Examples

```
>>> reps_to_intensity(5, slope = -5, constant = 100, quadratic =␣
↪False)
80
```

```
>>> reps_to_intensity(8, slope = -5, constant = 100, quadratic = True)
67.45
```

```
>>> reps_to_intensity(8, slope = -5, constant = 100, quadratic =␣
↪False)
65
```

Functions modeling progression

progression_linear

streprogen.**progression_linear**(*week*, *start_weight*, *end_weight*, *start_week*, *end_week*)

A linear progression function going through the points ('start_week', 'start_weight') and ('end_week', 'end_weight'), evaluated in 'week'.

Parameters

- **week** – The week to evaluate the linear function at.
- **start_weight** – The weight at 'start_week'.
- **end_weight** – The weight at 'end_week'.
- **start_week** – The number of the first week, typically 1.
- **end_week** – The number of the final week, e.g. 8.

Returns The weight at 'week'.

Return type weight

Examples

```
>>> progression_linear(week = 2, start_weight = 100, end_weight = 120,
...                    start_week = 1, end_week = 3)
110.0
```

```
>>> progression_linear(3, 100, 140, 1, 5)
120.0
```

progression_sinusoidal

streprogen.**progression_sinusoidal**(*week*, *start_weight*, *end_weight*, *start_week*, *end_week*, *periods=2*, *scale=0.025*, *offset=0*)

A sinusoidal progression function going through the points ('start_week', 'start_weight') and ('end_week', 'end_weight'), evaluated in 'week'. This function calls a linear progression function and multiplies it by a sinusoid.

Parameters

- **week** – The week to evaluate the linear function at.
- **start_weight** – The weight at 'start_week'.
- **end_weight** – The weight at 'end_week'.
- **start_week** – The number of the first week, typically 1.
- **end_week** – The number of the final week, e.g. 8.
- **periods** – Number of sinusoidal periods in the time range.
- **scale** – The scale (amplitude) of the sinusoidal term.

- **offset** – The offset (shift) of the sinusoid.

Returns The weight at 'week'.

Return type weight

Examples

```
>>> progression_sinusoidal(1, 100, 120, 1, 8)
100.0
>>> progression_sinusoidal(8, 100, 120, 1, 8)
120.0
>>> progression_sinusoidal(4, 100, 120, 1, 8)
106.44931454758678
```

The RepellentGenerator

RepellentGenerator

class streprogen.**RepellentGenerator**(*domain*, *probability_func=None*, *generated=None*)

Generates objects from a domain, each time an object is drawn, the probability of it being drawn again is determined by the probability function.

__init__(*domain*, *probability_func=None*, *generated=None*)

Initialize a RepellentGenerator, which is a generator where when an object is generated, the probability of it begin generated changes.

Parameters

- **domain** – A list of objects to generate from, e.g. [1, 2, 3].
- **probability_func** – A decreasing probability function, e.g. lambda x: 1 / 2**x.
- **generated** – A user specified dictionary of the form {element1: num1, element2: num2, ...} where num1, num2, ... are the initial states descriping how many times the elements element1, element2, ... have been generated. This argument changes the initial probability distribution.

Returns A RepellentGenerator object.

Return type *RepellentGenerator*

Examples

```
>>> domain = [1, 2, 3]
>>> generator = RepellentGenerator(domain)
>>> generator.generate_one() in domain
True
```

generate_one()

Generate a single element.

Returns An element from the domain.

Return type element

Examples

```
>>> generator = RepellentGenerator(['a', 'b'])
>>> gen_item = generator.generate_one()
>>> gen_item in ['a', 'b']
True
```

yield_from_domain(*num=1*)

Yield 'num' elements from the domain.

Yields *A sequence of elements from the domain.*

Examples

```
>>> domain = ['a', 1]
>>> generator = RepellentGenerator(domain)
>>> for element in generator.yield_from_domain(3):
...     print(element in domain)
True
True
True
```

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Python Module Index

S

streprogen, 23

Index

Symbols

__init__() (streprogen.Day method), 18
__init__() (streprogen.DynamicExercise method), 17
__init__() (streprogen.Program method), 19
__init__() (streprogen.RepellentGenerator method), 25
__init__() (streprogen.StaticExercise method), 18

A

add_days() (streprogen.Program method), 21
add_exercises() (streprogen.Day method), 19

D

Day (class in streprogen), 18
DynamicExercise (class in streprogen), 17

G

generate_one() (streprogen.RepellentGenerator method), 25

P

Program (class in streprogen), 19
progression_linear() (in module streprogen), 24
progression_sinusoidal() (in module streprogen), 24

R

render() (streprogen.Program method), 21
RepellentGenerator (class in streprogen), 25
reps_to_intensity() (in module streprogen), 23
repstring_penalty() (streprogen.Program static method), 21

S

StaticExercise (class in streprogen), 18
streprogen (module), 17, 23

T

to_html() (streprogen.Program method), 22
to_tex() (streprogen.Program method), 22
to_txt() (streprogen.Program method), 23

W

weekly_growth() (streprogen.DynamicExercise method), 17

Y

yield_from_domain() (streprogen.RepellentGenerator method), 26