

---

# **worldometer**

***Release 2.0.0***

**Matheus Felipe**

**Jan 16, 2024**



# CONTENTS

<b>1</b>	<b>About</b>	<b>3</b>
<b>2</b>	<b>Install</b>	<b>5</b>
<b>3</b>	<b>API Reference</b>	<b>7</b>
3.1	worldometer package . . . . .	7
3.2	world package . . . . .	8
3.3	population package . . . . .	16
3.4	geography package . . . . .	38
<b>4</b>	<b>Demo</b>	<b>47</b>
<b>5</b>	<b>worldometers.info</b>	<b>49</b>
<b>6</b>	<b>Data Sources</b>	<b>51</b>
<b>7</b>	<b>Contributions</b>	<b>53</b>
<b>8</b>	<b>License</b>	<b>55</b>
<b>9</b>	<b>Indices and tables</b>	<b>57</b>
	<b>Python Module Index</b>	<b>59</b>
	<b>Index</b>	<b>61</b>





*Get live, population, geography, projected, and historical data from around the world.*

---



## ABOUT

The [worldometer](#) package accesses various counters and live data available throughout the [worldometers.info](#) website and provides them through simple and self-describing classes, methods and attributes.

Access data on:

- The world
- Population
- Geography
- Projections
- Historical





## INSTALL

Use `pip` to install the `worldometer` package:

```
$ pip install worldometer
```



## API REFERENCE

The *worldometer* package is divided into several specific sub-packages for each dataset. So if you are looking for information on a specific package, class, or method, this part of the documentation is for you.

### 3.1 worldometer package

The *worldometer* package accesses various counters and live data available throughout the <https://www.worldometers.info> website and provides them through self-describing classes, methods and attributes.

#### Examples

Get the data from the live counters available on the homepage:

```
>>> from worldometer.world import WorldCounters

>>> wc = WorldCounters()

>>> wc.world_population.current_population
8065299074

>>> wc.government_and_economics.computers_produced_this_year
180248430

>>> wc.society_and_media.internet_users_in_the_world_today
5895566559
```

Reload data to get the latest:

```
>>> wc.reload_data()
>>> wc.world_population.current_population
8065300592
```

Get help and view information about mapped sections:

```
>>> help(wc)
```

## Notes

Check <https://www.worldometers.info/about> for more information about the data source, how live counters work, and more related information.

## 3.2 world package

### Index

- *world package*
  - *Live Counters*
  - *Country Codes*

*world* is the main API of the *worldometer* package, where you can access various live counters and data available on the <https://www.worldometers.info> website through self-descriptive classes, methods, and attributes.

### 3.2.1 Live Counters

Get the data provided by the live counters.

Each section is represented by an attribute in the *WorldCounters* class, and these attributes are instances of data classes that store the counter values.

Each of these data classes has attributes that describe the data stored in them.

```
>>> from worldometer.world import WorldCounters

>>> wc = WorldCounters()

>>> wc.world_population.current_population
8065299074

>>> wc.government_and_economics.computers_produced_this_year
180248430

>>> wc.society_and_media.internet_users_in_the_world_today
5895566559
```

**class** worldometer.world.counters.**WorldCounters**

Contains a reference to each section of the home page counters.

**source\_path**

The source path for the counters.

**Type**

str

**world\_population**

An instance of the *WorldPopulation* class that stores all counters related to population data.

**Type**

*WorldPopulation*

**government\_and\_economics**

An instance of the *GovernmentAndEconomics* class that stores all counters related to government and economic data.

**Type**

*GovernmentAndEconomics*

**society\_and\_media**

An instance of the *SocietyAndMedia* class that stores all counters related to society and media data.

**Type**

*SocietyAndMedia*

**environment**

An instance of the *Environment* class that stores all counters related to environmental data.

**Type**

*Environment*

**food**

An instance of the *Food* class that stores all counters related to food data.

**Type**

*Food*

**water**

An instance of the *Water* class that stores all counters related to water data.

**Type**

*Water*

**energy**

An instance of the *Energy* class that stores all counters related to energy data.

**Type**

*Energy*

**health**

An instance of the *Health* class that stores all counters related to health data.

**Type**

*Health*

**Notes**

For precise and up-to-date information on each section and its counters, please check the [worldometers homepage](#).

```
class worldometer.world.counters.WorldPopulation(_data: dataclasses.InitVar[Dict[str, Union[int, float, NoneType]]])
```

Counters related to world population data.

**current\_population****Type**

Union[int, float, None]

**births\_today****Type**

Union[int, float, None]

**births\_this\_year****Type**

Union[int, float, None]

**deaths\_today****Type**

Union[int, float, None]

**deaths\_this\_year****Type**

Union[int, float, None]

**net\_population\_growth\_today****Type**

Union[int, float, None]

**net\_population\_growth\_this\_year****Type**

Union[int, float, None]

```
class worldometer.world.counters.GovernmentAndEconomics(_data: dataclasses.InitVar[Dict[str, Union[int, float, NoneType]]])
```

Counters related to government and economic data.

**public\_healthcare\_expenditure\_today****Type**

Union[int, float, None]

**public\_education\_expenditure\_today****Type**

Union[int, float, None]

**public\_military\_expenditure\_today****Type**

Union[int, float, None]

**cars\_produced\_this\_year****Type**

Union[int, float, None]

**bicycles\_produced\_this\_year****Type**

Union[int, float, None]

**computers\_produced\_this\_year****Type**

Union[int, float, None]

```
class worldometer.world.counters.SocietyAndMedia(_data: dataclasses.InitVar[Dict[str, Union[int, float, NoneType]]])
```

Counters related to society and media data.

**new\_book\_titles\_published\_this\_year**

**Type**

Union[int, float, None]

**newspapers\_circulated\_today**

**Type**

Union[int, float, None]

**tv\_sets\_sold\_worldwide\_today**

**Type**

Union[int, float, None]

**cellular\_phones\_sold\_today**

**Type**

Union[int, float, None]

**money\_spent\_on\_videogames\_today**

**Type**

Union[int, float, None]

**internet\_users\_in\_the\_world\_today**

**Type**

Union[int, float, None]

**emails\_sent\_today**

**Type**

Union[int, float, None]

**blog\_posts\_written\_today**

**Type**

Union[int, float, None]

**tweets\_sent\_today**

**Type**

Union[int, float, None]

**google\_searches\_today**

**Type**

Union[int, float, None]

```
class worldometer.world.counters.Environment(_data: dataclasses.InitVar[Dict[str, Union[int, float,
NoneType]]])
```

Counters related to environmental data.

**forest\_loss\_this\_year**

**Type**

Union[int, float, None]

**land\_lost\_to\_soil\_erosion\_this\_year**

**Type**

Union[int, float, None]

**co2\_emissions\_this\_year**

**Type**

Union[int, float, None]

**desertification\_this\_year**

**Type**

Union[int, float, None]

**toxic\_chemicals\_released\_in\_the\_environment\_this\_year**

**Type**

Union[int, float, None]

**class** worldometer.world.counters.**Food**(*\_data: dataclasses.InitVar[Dict[str, Union[int, float, NoneType]]]*)

Counters related to food data.

**undernourished\_people\_in\_the\_world**

**Type**

Union[int, float, None]

**overweight\_people\_in\_the\_world**

**Type**

Union[int, float, None]

**obese\_people\_in\_the\_world**

**Type**

Union[int, float, None]

**people\_who\_died\_of\_hunger\_today**

**Type**

Union[int, float, None]

**money\_spent\_for\_obesity\_related\_diseases\_in\_the\_usa\_today**

**Type**

Union[int, float, None]

**money\_spent\_on\_weight\_loss\_programs\_in\_the\_usa\_today**

**Type**

Union[int, float, None]

**class** worldometer.world.counters.**Water**(*\_data: dataclasses.InitVar[Dict[str, Union[int, float, NoneType]]]*)

Counters related to water data.

**water\_used\_this\_year**

**Type**

Union[int, float, None]

**deaths\_caused\_by\_water\_related\_diseases\_this\_year**

**Type**

Union[int, float, None]



**people\_with\_no\_access\_to\_a\_safe\_drinking\_water\_source**

**Type**

Union[int, float, None]

```
class worldometer.world.counters.Energy(_data: dataclasses.InitVar[Dict[str, Union[int, float,
                                                                    NoneType]]])
```

Counters related to energy data.

**energy\_used\_today**

**Type**

Union[int, float, None]

**non\_renewable\_sources**

**Type**

Union[int, float, None]

**renewable\_sources**

**Type**

Union[int, float, None]

**solar\_energy\_striking\_earth\_today**

**Type**

Union[int, float, None]

**oil\_pumped\_today**

**Type**

Union[int, float, None]

**oil\_left**

**Type**

Union[int, float, None]

**days\_to\_the\_end\_of\_oil**

**Type**

Union[int, float, None]

**natural\_gas\_left**

**Type**

Union[int, float, None]

**days\_to\_the\_end\_of\_natural\_gas**

**Type**

Union[int, float, None]

**coal\_left**

**Type**

Union[int, float, None]

**days\_to\_the\_end\_of\_coal**

**Type**

Union[int, float, None]

```
class worldometer.world.counters.Health(_data: dataclasses.InitVar[Dict[str, Union[int, float, NoneType]]])
```

Counters related to health data.

**communicable\_disease\_deaths\_this\_year**

**Type**

Union[int, float, None]

**seasonal\_flu\_deaths\_this\_year**

**Type**

Union[int, float, None]

**deaths\_of\_children\_under\_5\_this\_year**

**Type**

Union[int, float, None]

**abortions\_this\_year**

**Type**

Union[int, float, None]

**deaths\_of\_mothers\_during\_birth\_this\_year**

**Type**

Union[int, float, None]

**hiv\_aids\_infected\_people**

**Type**

Union[int, float, None]

**deaths\_caused\_by\_hiv\_aids\_this\_year**

**Type**

Union[int, float, None]

**deaths\_caused\_by\_cancer\_this\_year**

**Type**

Union[int, float, None]

**deaths\_caused\_by\_malaria\_this\_year**

**Type**

Union[int, float, None]

**cigarettes\_smoked\_today**

**Type**

Union[int, float, None]

**deaths\_caused\_by\_smoking\_this\_year**

**Type**

Union[int, float, None]

**deaths\_caused\_by\_alcohol\_this\_year**

**Type**

Union[int, float, None]

**suicides\_this\_year****Type**

Union[int, float, None]

**money\_spent\_on\_illegal\_drugs\_this\_year****Type**

Union[int, float, None]

**road\_traffic\_accident\_fatalities\_this\_year****Type**

Union[int, float, None]

### 3.2.2 Country Codes

All countries have specific codes that represent them in some way. Get all of these codes with the [CountryCodes](#) class:

```
>>> from worldometer.world import CountryCodes

>>> cc = CountryCodes()

>>> cc.data[0]
CountryCodesData(
  country='Afghanistan',
  calling_code='93',
  three_letter_iso='AF',
  two_letter_iso='AFG',
  three_digit_iso_numeric=4
)
```

**class worldometer.world.country\_codes.CountryCodes**

Represents the data table of some codes used by each country.

**source\_path**

The data source path.

**Type**

str

**new\_column\_names**

The new column names that will be used to replace those of the original table.

**Type**

tuple

## Notes

Check the source table in [Worldometers Country Codes](#).

**property data:** `List[CountryCodesData]`

Get a list of all the data from the table.

Each index in the list contains an object representing a data row of the table.

```
class worldometer.world.country_codes.CountryCodesData(country: str, calling_code: str,  
                                                         three_letter_iso: str, two_letter_iso: str,  
                                                         three_digit_iso_numeric: str)
```

Represents a data row from the respective table.

**country**

**Type**  
str

**calling\_code**

**Type**  
str

**three\_letter\_iso**

**Type**  
str

**two\_letter\_iso**

**Type**  
str

**three\_digit\_iso\_numeric**

**Type**  
str

## 3.3 population package

### Index

- *population package*
  - *Population by country*
  - *Population by region*
  - *Population by year*
  - *Largest cities in the world*
  - *Most populous countries*
  - *World population projections*
  - *Regions population*

The population package provides access to various data related to world populations, regions, and their countries. Additionally, it includes historical data ranging from 1950 to the present day and future projections extending to 2050.

To understand the significance of each of these data sets, their sources, and any other related information, please visit the official page at <https://www.worldometers.info/population>

### 3.3.1 Population by country

```
>>> from worldometer.world.population import CountriesByPopulation

>>> cp = CountriesByPopulation()

>>> cp.data[0]
CountriesByPopulationData(
  idx=1,
  country='India',
  population=1428627663,
  yearly_change='0.81 %',
  net_change=11454490,
  density=481,
  land_area=2973190,
  migrants=-486136,
  fertility_rate=2.0,
  median_age=28.0,
  urban_population='36 %',
  world_share='17.76 %'
)
```

**class** worldometer.world.population.countries\_by\_population.CountriesByPopulation

Represents the data table of countries in the world by population.

**source\_path**

The data source path.

**Type**

str

**new\_column\_names**

The new column names that will be used to replace those of the original table.

**Type**

tuple

#### Notes

Check the source table in [Countries in the world by population](#).

**property data:** List[CountriesByPopulationData]

Get a list of all the data from the table.

Each index in the list contains an object representing a data row of the table.

```
class worldometer.world.population.countries_by_population.CountriesByPopulationData(idx:  
                                                                 int,  
                                                                 coun-  
                                                                 try:  
                                                                 str,  
                                                                 popu-  
                                                                 la-  
                                                                 tion:  
                                                                 int,  
                                                                 yearly_change:  
                                                                 str,  
                                                                 net_change:  
                                                                 int,  
                                                                 den-  
                                                                 sity:  
                                                                 int,  
                                                                 land_area:  
                                                                 int,  
                                                                 mi-  
                                                                 grants:  
                                                                 int,  
                                                                 fertil-  
                                                                 ity_rate:  
                                                                 float,  
                                                                 me-  
                                                                 dian_age:  
                                                                 float,  
                                                                 ur-  
                                                                 ban_population:  
                                                                 str,  
                                                                 world_share:  
                                                                 str)
```

Represents a data row from the respective table.

**idx**

**Type**  
int

**country**

**Type**  
str

**population**

**Type**  
int

**yearly\_change**

**Type**  
str

**net\_change**

**Type**  
int

**density**

Type  
int

**land\_area**

Type  
int

**migrants**

Type  
int

**fertility\_rate**

Type  
float

**median\_age**

Type  
float

**urban\_population**

Type  
str

**world\_share**

Type  
str

### 3.3.2 Population by region

```
>>> from worldometer.world.population import WorldPopulationByRegion

>>> pr = WorldPopulationByRegion()

>>> pr.current()[0]
CurrentWorldPopulationByRegionData(
    idx=1,
    region='Asia',
    population=4753079727,
    yearly_change='0.64 %',
    net_change=30444963,
    density=153,
    area=31033131,
    migrants=-1487191,
    fertility_rate=1.934,
    median_age=32,
    urban_population='52.6 %',
    world_share='59.1 %'
)
```

(continues on next page)

(continued from previous page)

```
>>> pr.past()[0]
PastWorldPopulationByRegionData(
  idx=1,
  region='Asia',
  population=1379048370,
  world_share='55.2 %'
)

>>> pr.future()[0]
FutureWorldPopulationByRegionData(
  idx=1,
  region='Asia',
  population=5292947571,
  world_share='54.5 %'
)
```

**class** worldometer.world.population.by\_region.**WorldPopulationByRegion**

Represents the data table of regions in the world by population.

**source\_path**

The data source path.

**Type**

str

**new\_column\_names**

The new column names that will be used to replace those of the original table.

**Type**

tuple

## Notes

Check the source table in [Regions in the world by population](#).

**current()** → List[*CurrentWorldPopulationByRegionData*]

Get a list of all the current data from the table.

These data are related to the current year.

Each index in the list contains an object representing a data row of the table.

**future()** → List[*FutureWorldPopulationByRegionData*]

Get a list of all future data from the table.

These data are an estimate for the year 2050.

Each index in the list contains an object representing a data row of the table.

**past()** → List[*PastWorldPopulationByRegionData*]

Get a list of all historical data from the table.

These data pertain to the year 1950.

Each index in the list contains an object representing a data row of the table.



```

class worldometer.world.population.by_region.CurrentWorldPopulationByRegionData(idx: int,
                                                                              region: str,
                                                                              population:
                                                                              int,
                                                                              yearly_change:
                                                                              str,
                                                                              net_change:
                                                                              int, density:
                                                                              int, area:
                                                                              int,
                                                                              migrants:
                                                                              int, fertil-
                                                                              ity_rate:
                                                                              float, me-
                                                                              dian_age:
                                                                              int, ur-
                                                                              ban_population:
                                                                              str,
                                                                              world_share:
                                                                              str)

```

Represents a data row from the respective table.

**idx**

Type  
int

**region**

Type  
str

**population**

Type  
int

**yearly\_change**

Type  
str

**net\_change**

Type  
int

**density**

Type  
int

**area**

Type  
int

**migrants**

**Type**  
int

**fertility\_rate**

**Type**  
float

**median\_age**

**Type**  
int

**urban\_population**

**Type**  
str

**world\_share**

**Type**  
str

```
class worldometer.world.population.by_region.PastWorldPopulationByRegionData(idx: int, region: str, population: int, world_share: str)
```

Represents a data row from the respective table.

**idx**

**Type**  
int

**region**

**Type**  
str

**population**

**Type**  
int

**world\_share**

**Type**  
str

```
class worldometer.world.population.by_region.FutureWorldPopulationByRegionData(idx: int, region: str, population: int, world_share: str)
```

Represents a data row from the respective table.

```

idx
    Type
    int

region
    Type
    str

population
    Type
    int

world_share
    Type
    str

```

### 3.3.3 Population by year

```

>>> from worldometer.world.population import WorldPopulationByYear

>>> py = WorldPopulationByYear()

>>> py.data[0]
WorldPopulationByYearData(
  year=2023,
  world_population=8045311447,
  yearly_change='0.88 %',
  net_change=70206291.0,
  density=54.0
)

```

**class worldometer.world.population.by\_year.WorldPopulationByYear**

Represents the data table of the world population by year.

**source\_path**

The data source path.

**Type**  
str

**new\_column\_names**

The new column names that will be used to replace those of the original table.

**Type**  
tuple

## Notes

Check the source table in the [world population by year](#).

**property data:** `List[WorldPopulationByYearData]`

Get a list of all the data from the table.

Each index in the list contains an object representing a data row of the table.

```
class worldometer.world.population.by_year.WorldPopulationByYearData(year: int,  
                                                                       world_population: int,  
                                                                       yearly_change: str,  
                                                                       net_change: float, density:  
                                                                       float)
```

Represents a data row from the respective table.

**year**

Type  
int

**world\_population**

Type  
int

**yearly\_change**

Type  
str

**net\_change**

Type  
float

**density**

Type  
float

## 3.3.4 Largest cities in the world

```
>>> from worldometer.world.population import LargestCities  
  
>>> lc = LargestCities()  
  
>>> lc.data[0]  
LargestCitiesData(  
    rank=1,  
    urban_area='Tokyo-Yokohama',  
    population_estimate=37843000,  
    country='Japan',  
    land_area=8547,  
    density=4400  
)
```

**class** worldometer.world.population.largest\_cities.LargestCities

Represents the data table of the largest cities in the world.

**source\_path**

The data source path.

**Type**

str

**new\_column\_names**

The new column names that will be used to replace those of the original table.

**Type**

tuple

## Notes

Check the source table in the [largest cities in the world](#).

**property data:** List[LargestCitiesData]

Get a list of all the data from the table.

Each index in the list contains an object representing a data row of the table.

**class** worldometer.world.population.largest\_cities.LargestCitiesData(*rank: int, urban\_area: str, population\_estimate: str, country: str, land\_area: int, density: int*)

Represents a data row from the respective table.

**rank**

**Type**

int

**urban\_area**

**Type**

str

**population\_estimate**

**Type**

str

**country**

**Type**

str

**land\_area**

**Type**

int

**density**

**Type**

int

### 3.3.5 Most populous countries

```
>>> from worldometer.world.population import MostPopulousCountries

>>> pc = MostPopulousCountries()

>>> pc.current()[0]
CurrentMostPopulousCountriesData(
  idx=1,
  country='India',
  population=1428627663,
  yearly_change='0.81 %',
  world_share='17.8 %'
)

>>> pc.past()[0]
PastMostPopulousCountriesData(
  idx=1,
  country='China',
  population=543979233,
  world_share='21.8 %',
  rank='(2)'
)

>>> pc.future()[0]
FutureMostPopulousCountriesData(
  idx=1,
  country='India',
  population=1670490596,
  world_share='17.2 %',
  rank='(1)'
)
```

**class** worldometer.world.population.most\_populous\_countries.MostPopulousCountries

Represents the data table of most populous countries in the world.

**source\_path**

The data source path.

**Type**

str

**new\_column\_names**

The new column names that will be used to replace those of the original table.

**Type**

tuple

## Notes

Check the source table in [Most populous countries in the world](#).

**current()** → List[*CurrentMostPopulousCountriesData*]

Get a list of all the current data from the table.

These data are related to the current year.

Each index in the list contains an object representing a data row of the table.

**future()** → List[*FutureMostPopulousCountriesData*]

Get a list of all future data from the table.

These data are an estimate for the year 2050.

Each index in the list contains an object representing a data row of the table.

**past()** → List[*PastMostPopulousCountriesData*]

Get a list of all historical data from the table.

These data pertain to the year 1950.

Each index in the list contains an object representing a data row of the table.

```
class worldometer.world.population.most_populous_countries.CurrentMostPopulousCountriesData(idx:
                                                                    int,
                                                                    coun-
                                                                    try:
                                                                    str,
                                                                    pop-
                                                                    u-
                                                                    la-
                                                                    tion:
                                                                    int,
                                                                    yearly_change:
                                                                    str,
                                                                    world_share:
                                                                    str)
```

Represents a data row from the respective table.

**idx**

Type

int

**country**

Type

str

**population**

Type

int

**yearly\_change**

Type

str

**world\_share**

**Type**  
str

```
class worldometer.world.population.most_populous_countries.PastMostPopulousCountriesData(idx:  
                                                                 int,  
                                                                 coun-  
                                                                 try:  
                                                                 str,  
                                                                 pop-  
                                                                 u-  
                                                                 la-  
                                                                 tion:  
                                                                 int,  
                                                                 world_share:  
                                                                 str,  
                                                                 rank:  
                                                                 str)
```

Represents a data row from the respective table.

**idx**

**Type**  
int

**country**

**Type**  
str

**population**

**Type**  
int

**world\_share**

**Type**  
str

**rank**

**Type**  
str

```
class worldometer.world.population.most_populous_countries.FutureMostPopulousCountriesData(idx:  
                                                                 int,  
                                                                 coun-  
                                                                 try:  
                                                                 str,  
                                                                 pop-  
                                                                 u-  
                                                                 la-  
                                                                 tion:  
                                                                 int,  
                                                                 world_share:  
                                                                 str,  
                                                                 rank:  
                                                                 str)
```



Represents a data row from the respective table.

**idx**

**Type**  
int

**country**

**Type**  
str

**population**

**Type**  
int

**world\_share**

**Type**  
str

**rank**

**Type**  
str

### 3.3.6 World population projections

```
>>> from worldometer.world.population import WorldPopulationProjections
>>> pp = WorldPopulationProjections()
>>> pp.data[0]
WorldPopulationProjectionsData(
  year=2023,
  world_population=8045311447,
  yearly_change='0.88 %',
  net_change=70206291,
  density=54
)
```

**class worldometer.world.population.projections.WorldPopulationProjections**

Represents the data table of the world population projections.

**source\_path**

The data source path.

**Type**  
str

**new\_column\_names**

The new column names that will be used to replace those of the original table.

**Type**  
tuple

## Notes

Check the source table in [World Population Projections](#).

**property data:** `List[WorldPopulationProjectionsData]`

Get a list of all the data from the table.

Each index in the list contains an object representing a data row of the table.

```
class worldometer.world.population.projections.WorldPopulationProjectionsData(year: int,
                                                                              world_population:
                                                                              int,
                                                                              yearly_change:
                                                                              str,
                                                                              net_change:
                                                                              int, density:
                                                                              int)
```

Represents a data row from the respective table.

**year**

Type  
int

**world\_population**

Type  
int

**yearly\_change**

Type  
str

**net\_change**

Type  
int

**density**

Type  
int

### 3.3.7 Regions population

```
>>> from worldometer.world.population import (
    AsiaPopulation,
    AfricaPopulation,
    EuropePopulation,
    LatinAmericanAndTheCaribbeanPopulation,
    NorthernAmericanPopulation,
    OceaniaPopulation
)

>>> ap = AsiaPopulation()
```

(continues on next page)

(continued from previous page)

```

>>> ap.live()
4762699828

>>> ap.subregions()[0]
SubregionData(
  area='Southern Asia',
  population='(2,027,578,876)'
)

>>> ap.historical()[0]
HistoricalData(
  year=2023,
  population=4753079727,
  yearly_change_percent='0.64 %',
  yearly_change=30444963,
  migrants=-1487191,
  median_age=31.9,
  fertility_rate=1.93,
  density=153,
  urban_population_percent='52.6 %',
  urban_population=2500201501,
  world_share='59.1 %',
  world_population=8045311447,
  rank=nan
)

>>> ap.forecast()[0]
ForecastData(
  year=2025,
  population=4816249054,
  yearly_change_percent='0.64 %',
  yearly_change=30384996,
  migrants=-1555419,
  median_age=32.7,
  fertility_rate=1.93,
  density=155,
  urban_population_percent='53.8 %',
  urban_population=2589655469,
  world_share='61.4 %',
  world_population=8191988453,
  rank=nan
)

```

**class worldometer.world.population.regions.AsiaPopulation**

Represents the data tables of Asia's populations.

**source\_path**

The data source path.

**Type**

str

**new\_column\_names**

The new column names that will be used to replace those of the original tables.

**Type**  
tuple

### Notes

Check the source tables in [Asia Population](#).

**forecast()** → List[[ForecastData](#)]

Get a list of all forecast data from the table.

Each index in the list contains an object representing a data row of the table.

**historical()** → List[[HistoricalData](#)]

Get a list of all historical data from the table.

Each index in the list contains an object representing a data row of the table.

**live()** → int | float | None

Get a live population counter for the respective region.

**subregions()** → List[[SubregionData](#)]

Get a list of all the subregions' data from the table.

Each index in the list contains an object representing a data row of the table.

**class** worldometer.world.population.regions.**AfricaPopulation**

Represents the data tables of Africa's populations.

**source\_path**

The data source path.

**Type**  
str

**new\_column\_names**

The new column names that will be used to replace those of the original tables.

**Type**  
tuple

### Notes

Check the source tables in [Africa Population](#).

**forecast()** → List[[ForecastData](#)]

Get a list of all forecast data from the table.

Each index in the list contains an object representing a data row of the table.

**historical()** → List[[HistoricalData](#)]

Get a list of all historical data from the table.

Each index in the list contains an object representing a data row of the table.

**live()** → int | float | None

Get a live population counter for the respective region.

**subregions()** → List[*SubregionData*]

Get a list of all the subregions' data from the table.

Each index in the list contains an object representing a data row of the table.

**class** worldometer.world.population.regions.**EuropePopulation**

Represents the data tables of Europe's populations.

**source\_path**

The data source path.

**Type**

str

**new\_column\_names**

The new column names that will be used to replace those of the original tables.

**Type**

tuple

## Notes

Check the source tables in [Europe Population](#).

**forecast()** → List[*ForecastData*]

Get a list of all forecast data from the table.

Each index in the list contains an object representing a data row of the table.

**historical()** → List[*HistoricalData*]

Get a list of all historical data from the table.

Each index in the list contains an object representing a data row of the table.

**live()** → int | float | None

Get a live population counter for the respective region.

**subregions()** → List[*SubregionData*]

Get a list of all the subregions' data from the table.

Each index in the list contains an object representing a data row of the table.

**class** worldometer.world.population.regions.**LatinAmericanAndTheCaribbeanPopulation**

Represents the data tables of Latin American and Caribbean populations.

**source\_path**

The data source path.

**Type**

str

**new\_column\_names**

The new column names that will be used to replace those of the original tables.

**Type**

tuple

## Notes

Check the source tables in [Latin America and the Caribbean Population](#).

**forecast()** → List[*ForecastData*]

Get a list of all forecast data from the table.

Each index in the list contains an object representing a data row of the table.

**historical()** → List[*HistoricalData*]

Get a list of all historical data from the table.

Each index in the list contains an object representing a data row of the table.

**live()** → int | float | None

Get a live population counter for the respective region.

**subregions()** → List[*SubregionData*]

Get a list of all the subregions' data from the table.

Each index in the list contains an object representing a data row of the table.

**class** worldometer.world.population.regions.**NorthernAmericanPopulation**

Represents the data tables of Northern American populations.

**source\_path**

The data source path.

**Type**

str

**new\_column\_names**

The new column names that will be used to replace those of the original tables.

**Type**

tuple

## Notes

Check the source tables in [Northern American Population](#).

**forecast()** → List[*ForecastData*]

Get a list of all forecast data from the table.

Each index in the list contains an object representing a data row of the table.

**historical()** → List[*HistoricalData*]

Get a list of all historical data from the table.

Each index in the list contains an object representing a data row of the table.

**live()** → int | float | None

Get a live population counter for the respective region.

**subregions()** → List[*SubregionData*]

Get a list of all the subregions' data from the table.

Each index in the list contains an object representing a data row of the table.

**class** worldometer.world.population.regions.OceaniaPopulation

Represents the data tables of the Oceania populations.

**source\_path**

The data source path.

**Type**

str

**new\_column\_names**

The new column names that will be used to replace those of the original tables.

**Type**

tuple

## Notes

Check the source tables in [Oceania Population](#).

**forecast()** → List[*ForecastData*]

Get a list of all forecast data from the table.

Each index in the list contains an object representing a data row of the table.

**historical()** → List[*HistoricalData*]

Get a list of all historical data from the table.

Each index in the list contains an object representing a data row of the table.

**live()** → int | float | None

Get a live population counter for the respective region.

**subregions()** → List[*SubregionData*]

Get a list of all the subregions' data from the table.

Each index in the list contains an object representing a data row of the table.

**class** worldometer.world.population.regions.SubregionData(*area: int, population: str*)

Represents a data row from the respective table.

**area**

**Type**

int

**population**

**Type**

str

**class** worldometer.world.population.regions.HistoricalData(*year: int, population: int, yearly\_change\_percent: str, yearly\_change: int, migrants: int, median\_age: float, fertility\_rate: float, density: int, urban\_population\_percent: str, urban\_population: int, world\_share: str, world\_population: int, rank: int*)

Represents a data row from the respective table.

**year**

Type  
int

**population**

Type  
int

**yearly\_change\_percent**

Type  
str

**yearly\_change**

Type  
int

**migrants**

Type  
int

**median\_age**

Type  
float

**fertility\_rate**

Type  
float

**density**

Type  
int

**urban\_population\_percent**

Type  
str

**urban\_population**

Type  
int

**world\_share**

Type  
str

**world\_population**

Type  
int



**rank**

**Type**  
int

```
class worldometer.world.population.regions.ForecastData(year: int, population: int,
    yearly_change_percent: str, yearly_change:
    int, migrants: int, median_age: float,
    fertility_rate: float, density: int,
    urban_population_percent: str,
    urban_population: int, world_share: str,
    world_population: int, rank: int)
```

Represents a data row from the respective table.

**year**

**Type**  
int

**population**

**Type**  
int

**yearly\_change\_percent**

**Type**  
str

**yearly\_change**

**Type**  
int

**migrants**

**Type**  
int

**median\_age**

**Type**  
float

**fertility\_rate**

**Type**  
float

**density**

**Type**  
int

**urban\_population\_percent**

**Type**  
str

**urban\_population**

Type  
int

**world\_share**

Type  
str

**world\_population**

Type  
int

**rank**

Type  
int

## 3.4 geography package

### Index

- *geography package*
  - *Countries in the world*
  - *Largest countries in the world*

The *geography* package provides access to various data related to geographic information about the world, regions, and their countries.

To understand the significance of each of these data sets, their sources, and any other related information, please visit the official page at <https://www.worldometers.info/geography/how-many-countries-are-there-in-the-world>

### 3.4.1 Countries in the world

```
>>> from worldometer.world.geography import (  
    WorldCountries,  
    AsiaCountries,  
    AfricaCountries,  
    EuropeCountries,  
    LatinAmericanAndTheCaribbeanCountries,  
    NorthernAmericanCountries,  
    OceaniaCountries  
)  
  
>>> wc = WorldCountries()  
  
>>> wc.total  
195
```

(continues on next page)

(continued from previous page)

```

>>> wc.countries()[0]
WorldCountriesData(
  idx=1,
  country='India',
  population=1428627663,
  world_share='17.76 %',
  land_area=2973190
)

>>> ac = AfricaCountries()

>>> ac.total
54

>>> ac.countries()[0]
CountryData(
  idx=1,
  country='Nigeria',
  population=223804632,
  subregion='Western Africa'
)

>>> ac.dependencies()[0]
DependencyData(
  idx=1,
  territory='Réunion',
  population=981796,
  dependency_of='France'
)

```

**class worldometer.world.geography.countries.WorldCountries**

Represents the data table of a list of countries in the world.

**source\_path**

The data source path.

**Type**

str

**new\_column\_names**

The new column names that will be used to replace those of the original table.

**Type**

tuple

**total**

The total number of countries in the world.

**Type**

int

## Notes

Check the source table in [List of countries](#).

**countries()** → List[[WorldCountriesData](#)]

Get a list of all the countries' data from the table.

Each index in the list contains an object representing a data row of the table.

**class** worldometer.world.geography.countries.**AsiaCountries**

Represents the data tables of Asia's countries.

**source\_path**

The data source path.

**Type**

str

**new\_column\_names**

The new column names that will be used to replace those of the original tables.

**Type**

tuple

**total**

The total number of countries in the region.

**Type**

int

## Notes

Check the source tables in [Countries in Asia](#).

**countries()** → List[[CountryData](#)]

Get a list of all the data for countries in the region from the table.

Each index in the list contains an object representing a data row of the table.

**dependencies()** → List[[DependencyData](#)]

Get a list of all the data for dependencies in the region from the table.

Each index in the list contains an object representing a data row of the table.

**class** worldometer.world.geography.countries.**AfricaCountries**

Represents the data tables of Africa's countries.

**source\_path**

The data source path.

**Type**

str

**new\_column\_names**

The new column names that will be used to replace those of the original tables.

**Type**

tuple

**total**

The total number of countries in the region.

**Type**

int

**Notes**

Check the source tables in [Countries in Africa](#).

**countries()** → List[[CountryData](#)]

Get a list of all the data for countries in the region from the table.

Each index in the list contains an object representing a data row of the table.

**dependencies()** → List[[DependencyData](#)]

Get a list of all the data for dependencies in the region from the table.

Each index in the list contains an object representing a data row of the table.

**class** worldometer.world.geography.countries.**EuropeCountries**

Represents the data tables of Europe's countries.

**source\_path**

The data source path.

**Type**

str

**new\_column\_names**

The new column names that will be used to replace those of the original tables.

**Type**

tuple

**total**

The total number of countries in the region.

**Type**

int

**Notes**

Check the source tables in [Countries in Europe](#).

**countries()** → List[[CountryData](#)]

Get a list of all the data for countries in the region from the table.

Each index in the list contains an object representing a data row of the table.

**dependencies()** → List[[DependencyData](#)]

Get a list of all the data for dependencies in the region from the table.

Each index in the list contains an object representing a data row of the table.

**class** worldometer.world.geography.countries.**LatinAmericanAndTheCaribbeanCountries**

Represents the data tables of Latin American And The Caribbean countries.

**source\_path**

The data source path.

**Type**

str

**new\_column\_names**

The new column names that will be used to replace those of the original tables.

**Type**

tuple

**total**

The total number of countries in the region.

**Type**

int

**Notes**

Check the source tables in [Countries in Latin American And The Caribbean](#).

**countries()** → List[*CountryData*]

Get a list of all the data for countries in the region from the table.

Each index in the list contains an object representing a data row of the table.

**dependencies()** → List[*DependencyData*]

Get a list of all the data for dependencies in the region from the table.

Each index in the list contains an object representing a data row of the table.

**class** worldometer.world.geography.countries.**NorthernAmericanCountries**

Represents the data tables of Northern American countries.

**source\_path**

The data source path.

**Type**

str

**new\_column\_names**

The new column names that will be used to replace those of the original tables.

**Type**

tuple

**total**

The total number of countries in the region.

**Type**

int

## Notes

Check the source tables in [Countries in Northern American](#).

**countries()** → List[[CountryData](#)]

Get a list of all the data for countries in the region from the table.

Each index in the list contains an object representing a data row of the table.

**dependencies()** → List[[DependencyData](#)]

Get a list of all the data for dependencies in the region from the table.

Each index in the list contains an object representing a data row of the table.

**class** worldometer.world.geography.countries.**OceaniaCountries**

Represents the data tables of the Oceania countries.

**source\_path**

The data source path.

**Type**

str

**new\_column\_names**

The new column names that will be used to replace those of the original tables.

**Type**

tuple

**total**

The total number of countries in the region.

**Type**

int

## Notes

Check the source tables in [Countries in Oceania](#).

**countries()** → List[[CountryData](#)]

Get a list of all the data for countries in the region from the table.

Each index in the list contains an object representing a data row of the table.

**dependencies()** → List[[DependencyData](#)]

Get a list of all the data for dependencies in the region from the table.

Each index in the list contains an object representing a data row of the table.

**class** worldometer.world.geography.countries.**WorldCountriesData**(*idx: int, country: str, population: int, world\_share: str, land\_area: int*)

Represents a data row from the respective table.

**idx**

**Type**

int

**country**

**Type**  
str

**population**

**Type**  
int

**world\_share**

**Type**  
str

**land\_area**

**Type**  
int

```
class worldometer.world.geography.countries.CountryData(idx: int, country: str, population: int,  
                                                         subregion: str)
```

Represents a data row from the respective table.

**idx**

**Type**  
int

**country**

**Type**  
str

**population**

**Type**  
int

**subregion**

**Type**  
str

```
class worldometer.world.geography.countries.DependencyData(idx: int, territory: str, population: int,  
                                                             dependency_of: str)
```

Represents a data row from the respective table.

**idx**

**Type**  
int

**territory**

**Type**  
str

**population**

**Type**  
int



**dependency\_of**

**Type**

str

### 3.4.2 Largest countries in the world

```
>>> from worldometer.world.geography import LargestCountries

>>> lc = LargestCountries()

>>> lc.data[0]
LargestCountriesData(
  idx=1,
  country='Russia',
  total_area_km2=17098242,
  total_area_mi2=6601665,
  land_area_km2=16376870,
  land_area_mi2=6323142,
  percentage_of_world_landmass='11.0 %'
)
```

**class** worldometer.world.geography.largest\_countries.LargestCountries

Represents the data table of the largest countries in the world (by area).

**source\_path**

The data source path.

**Type**

str

**new\_column\_names**

The new column names that will be used to replace those of the original table.

**Type**

tuple

#### Notes

Check the source table in the [Largest Countries in the World](#).

**property data:** List[LargestCountriesData]

Get a list of all the data from the table.

Each index in the list contains an object representing a data row of the table.

```
class worldometer.world.geography.largest_countries.LargestCountriesData(idx: int, country: str,
    total_area_km2: int,
    total_area_mi2: int,
    land_area_km2: int,
    land_area_mi2: int,
    percent-
    age_of_world_landmass:
    str)
```

Represents a data row from the respective table.

**idx**

**Type**  
int

**country**

**Type**  
str

**total\_area\_km2**

**Type**  
int

**total\_area\_mi2**

**Type**  
int

**land\_area\_km2**

**Type**  
int

**land\_area\_mi2**

**Type**  
int

**percentage\_of\_world\_landmass**

**Type**  
str

## DEMO

---

**Note:** The first time you run any function/method or class, it will download Chromium to ~/.local/share/pypeteer directory. It only happens once. After, it will only open the chromium to render the contents of worldometers.info.

---

Get the data from the live counters available on the [homepage](#):

```
>>> from worldometer.world import WorldCounters
>>> wc = WorldCounters()
>>> wc.world_population.current_population
8065299074
>>> wc.government_and_economics.computers_produced_this_year
180248430
>>> wc.society_and_media.internet_users_in_the_world_today
5895566559
```

Reload data to get the latest:

```
>>> wc.reload_data()
>>> wc.world_population.current_population
8065300592
```

Get help and view information about mapped sections:

```
>>> help(wc)
```



## **WORLDOMETERS.INFO**

“Worldometer is run by an international team of developers, researchers, and volunteers with the goal of making world statistics available in a thought-provoking and time relevant format to a wide audience around the world. It is published by a small and independent digital media company based in the United States. We have no political, governmental, or corporate affiliation. Furthermore, we have no investors, donors, grants, or backers of any type. We are completely independent and self-financed through automated programmatic advertising sold in real time on multiple ad exchanges.”

More info: [worldometers.info/about](http://worldometers.info/about)



## DATA SOURCES

**[adapted]:** “worldometers.info collects its statistics and data from the most reputable national and international organizations, including the United Nations, the World Health Organization, the Food and Agriculture Organization, OECD and others.

Each Worldometer counter has its specific set of sources, which are listed on its dedicated page (accessible by clicking on the counter text link, when available).

Data, estimates, and projections displayed on worldometers.info counters are for the most part provided by organizations included in the following list of United Nations Statistics Division’s partners.”

More info about data source: [worldometers.info/sources](https://worldometers.info/sources)





## CONTRIBUTIONS

All contributions are welcome!

Found a problem, want to give a tip? [open an issue](#)

Do you have a solution to the problem? [Send me a PR](#)

Did you like this project? [Click on the star](#)



## LICENSE

This project is using the MIT license, see in [MIT LICENSE](#)



## INDICES AND TABLES

- `genindex`
- `modindex`
- `search`



## PYTHON MODULE INDEX

### W

- `worldometer`, [7](#)
- `worldometer.world.counters`, [8](#)
- `worldometer.world.country_codes`, [15](#)
- `worldometer.world.geography.countries`, [39](#)
- `worldometer.world.geography.largest_countries`,  
[45](#)
- `worldometer.world.population.by_region`, [20](#)
- `worldometer.world.population.by_year`, [23](#)
- `worldometer.world.population.countries_by_population`,  
[17](#)
- `worldometer.world.population.largest_cities`,  
[24](#)
- `worldometer.world.population.most_populous_countries`,  
[26](#)
- `worldometer.world.population.projections`, [29](#)
- `worldometer.world.population.regions`, [31](#)





## INDEX

### A

`abortions_this_year` (worldometer.world.counters.Health attribute), 14

`AfricaCountries` (class in worldometer.world.geography.countries), 40

`AfricaPopulation` (class in worldometer.world.population.regions), 32

`area` (worldometer.world.population.by\_region.CurrentWorldPopulationByRegionData attribute), 21

`area` (worldometer.world.population.regions.SubregionData attribute), 35

`AsiaCountries` (class in worldometer.world.geography.countries), 40

`AsiaPopulation` (class in worldometer.world.population.regions), 31

### B

`bicycles_produced_this_year` (worldometer.world.counters.GovernmentAndEconomics attribute), 10

`births_this_year` (worldometer.world.counters.WorldPopulation attribute), 9

`births_today` (worldometer.world.counters.WorldPopulation attribute), 9

`blog_posts_written_today` (worldometer.world.counters.SocietyAndMedia attribute), 11

### C

`calling_code` (worldometer.world.country\_codes.CountryCodesData attribute), 16

`cars_produced_this_year` (worldometer.world.counters.GovernmentAndEconomics attribute), 10

`cellular_phones_sold_today` (worldometer.world.counters.SocietyAndMedia attribute), 11

`cigarettes_smoked_today` (worldometer.world.counters.Health attribute), 14

`co2_emissions_this_year` (worldometer.world.counters.Environment attribute), 11

`coal_left` (worldometer.world.counters.Energy attribute), 13

`communicable_disease_deaths_this_year` (worldometer.world.counters.Health attribute), 14

`computers_produced_this_year` (worldometer.world.counters.GovernmentAndEconomics attribute), 10

`countries()` (worldometer.world.geography.countries.AfricaCountries method), 41

`countries()` (worldometer.world.geography.countries.AsiaCountries method), 40

`countries()` (worldometer.world.geography.countries.EuropeCountries method), 41

`countries()` (worldometer.world.geography.countries.LatinAmericanAndTheCaribbeanC method), 42

`countries()` (worldometer.world.geography.countries.NorthernAmericanCountries method), 43

`countries()` (worldometer.world.geography.countries.OceaniaCountries method), 43

`countries()` (worldometer.world.geography.countries.WorldCountries method), 40

`CountriesByPopulation` (class in worldometer.world.population.countries\_by\_population), 17

`CountriesByPopulationData` (class in worldometer.world.population.countries\_by\_population), 17

`country` (worldometer.world.country\_codes.CountryCodesData attribute), 16

`country` (worldometer.world.geography.countries.CountryData attribute), 44

country (worldometer.world.geography.countries.WorldCountriesData attribute), 43

country (worldometer.world.geography.largest\_countries.LargestCountriesData attribute), 46

country (worldometer.world.population.countries\_by\_population.CountriesByPopulationData attribute), 18

country (worldometer.world.population.largest\_cities.LargestCitiesData attribute), 25

country (worldometer.world.population.most\_populous\_countries.CurrentMostPopulousCountriesData attribute), 27

country (worldometer.world.population.most\_populous\_countries.FutureMostPopulousCountriesData attribute), 29

country (worldometer.world.population.most\_populous\_countries.PastMostPopulousCountriesData attribute), 28

CountryCodes (class in worldometer.world.country\_codes), 15

CountryCodesData (class in worldometer.world.country\_codes), 16

CountryData (class in worldometer.world.geography.countries), 44

current() (worldometer.world.population.by\_region.WorldPopulationByRegion method), 20

current() (worldometer.world.population.most\_populous\_countries.MostPopulousCountries method), 27

current\_population (worldometer.world.counters.WorldPopulation attribute), 9

CurrentMostPopulousCountriesData (class in worldometer.world.population.most\_populous\_countries), 27

CurrentWorldPopulationByRegionData (class in worldometer.world.population.by\_region), 20

## D

data (worldometer.world.country\_codes.CountryCodes property), 16

data (worldometer.world.geography.largest\_countries.LargestCountriesData property), 45

data (worldometer.world.population.by\_year.WorldPopulationByYearData property), 24

data (worldometer.world.population.countries\_by\_population.CountriesByPopulationData property), 17

data (worldometer.world.population.largest\_cities.LargestCitiesData property), 25

data (worldometer.world.population.projections.WorldPopulationProjectionsData property), 30

days\_to\_the\_end\_of\_coal (worldometer.world.counters.Energy attribute), 13

days\_to\_the\_end\_of\_natural\_gas (worldometer.world.counters.Energy attribute), 13

days\_to\_the\_end\_of\_oil (worldometer.world.counters.Energy attribute), 13

deaths\_caused\_by\_alcohol\_this\_year (worldometer.world.counters.Health attribute), 14

deaths\_caused\_by\_hiv\_aids\_this\_year (worldometer.world.counters.Health attribute), 14

deaths\_caused\_by\_malaria\_this\_year (worldometer.world.counters.Health attribute), 14

deaths\_caused\_by\_smoking\_this\_year (worldometer.world.counters.Health attribute), 14

deaths\_caused\_by\_water\_related\_diseases\_this\_year (worldometer.world.counters.Water attribute), 12

deaths\_of\_children\_under\_5\_this\_year (worldometer.world.counters.Health attribute), 14

deaths\_of\_mothers\_during\_birth\_this\_year (worldometer.world.counters.Health attribute), 14

deaths\_this\_year (worldometer.world.counters.WorldPopulation attribute), 10

deaths\_today (worldometer.world.counters.WorldPopulation attribute), 10

density (worldometer.world.population.by\_region.CurrentWorldPopulationByRegionData attribute), 21

density (worldometer.world.population.by\_year.WorldPopulationByYearData attribute), 24

density (worldometer.world.population.countries\_by\_population.CountriesByPopulationData attribute), 18

density (worldometer.world.population.largest\_cities.LargestCitiesData attribute), 25

density (worldometer.world.population.projections.WorldPopulationProjectionsData attribute), 30

density (worldometer.world.population.regions.ForecastData attribute), 37

density (worldometer.world.population.regions.HistoricalData attribute), 36

dependencies() (worldometer.world.geography.countries.AfricaCountries method), 41

dependencies() (worldometer.world.geography.countries.AsiaCountries method), 40

dependencies() (worldometer.world.geography.countries.EuropeCountries method), 41

dependencies() (worldometer.world.geography.countries.LatinAmericanAndTheCaribbeanCountries method), 42

[dependencies\(\)](#) (worldometer.world.geography.countries.NorthernAmericanPopulation method), 32  
[dependencies\(\)](#) (worldometer.world.geography.countries.OceaniaCountries method), 43  
[dependency\\_of](#) (worldometer.world.geography.countries.DependencyData attribute), 44  
[DependencyData](#) (class in worldometer.world.geography.countries), 44  
[desertification\\_this\\_year](#) (worldometer.world.counters.Environment attribute), 12  
**E**  
[emails\\_sent\\_today](#) (worldometer.world.counters.SocietyAndMedia attribute), 11  
[Energy](#) (class in worldometer.world.counters), 13  
[energy](#) (worldometer.world.counters.WorldCounters attribute), 9  
[energy\\_used\\_today](#) (worldometer.world.counters.Energy attribute), 13  
[Environment](#) (class in worldometer.world.counters), 11  
[environment](#) (worldometer.world.counters.WorldCounters attribute), 9  
[EuropeCountries](#) (class in worldometer.world.geography.countries), 41  
[EuropePopulation](#) (class in worldometer.world.population.regions), 33  
**F**  
[fertility\\_rate](#) (worldometer.world.population.by\_region.CurrentWorldPopulationByRegionData attribute), 22  
[fertility\\_rate](#) (worldometer.world.population.countries\_by\_population.CountriesByPopulationData attribute), 19  
[fertility\\_rate](#) (worldometer.world.population.regions.ForecastData attribute), 37  
[fertility\\_rate](#) (worldometer.world.population.regions.HistoricalData attribute), 36  
[Food](#) (class in worldometer.world.counters), 12  
[food](#) (worldometer.world.counters.WorldCounters attribute), 9  
[forecast\(\)](#) (worldometer.world.population.regions.AfricaPopulation method), 32  
[forecast\(\)](#) (worldometer.world.population.regions.AsiaPopulation method), 32  
[forecast\(\)](#) (worldometer.world.population.regions.EuropePopulation method), 33  
[forecast\(\)](#) (worldometer.world.population.regions.LatinAmericanAndTheCaribbeanPopulation method), 34  
[forecast\(\)](#) (worldometer.world.population.regions.NorthernAmericanPopulation method), 34  
[forecast\(\)](#) (worldometer.world.population.regions.OceaniaPopulation method), 35  
[ForecastData](#) (class in worldometer.world.population.regions), 37  
[forest\\_loss\\_this\\_year](#) (worldometer.world.counters.Environment attribute), 11  
[future\(\)](#) (worldometer.world.population.by\_region.WorldPopulationByRegionData method), 20  
[future\(\)](#) (worldometer.world.population.most\_populous\_countries.MostPopulousCountriesData method), 27  
[FutureMostPopulousCountriesData](#) (class in worldometer.world.population.most\_populous\_countries), 28  
[FutureWorldPopulationByRegionData](#) (class in worldometer.world.population.by\_region), 22  
**G**  
[google\\_searches\\_today](#) (worldometer.world.counters.SocietyAndMedia attribute), 11  
[government\\_and\\_economics](#) (worldometer.world.counters.WorldCounters attribute), 10  
[GovernmentAndEconomics](#) (class in worldometer.world.counters), 10  
**H**  
[Health](#) (class in worldometer.world.counters), 13  
[health](#) (worldometer.world.counters.WorldCounters attribute), 9  
[historical\(\)](#) (worldometer.world.population.regions.AfricaPopulation method), 32  
[historical\(\)](#) (worldometer.world.population.regions.AsiaPopulation method), 32  
[historical\(\)](#) (worldometer.world.population.regions.EuropePopulation method), 33  
[historical\(\)](#) (worldometer.world.population.regions.LatinAmericanAndTheCaribbeanPopulation method), 34

method), 34

historical() (worldometer.world.population.regions.NorthernAmericanPopulation attribute), 34

historical() (worldometer.world.population.regions.OceaniaPopulation attribute), 35

HistoricalData (class in worldometer.world.population.regions), 35

hiv\_aids\_infected\_people (worldometer.world.counters.Health attribute), 14

I

idx(worldometer.world.geography.countries.CountryData attribute), 44

idx(worldometer.world.geography.countries.DependencyData attribute), 44

idx(worldometer.world.geography.countries.WorldCountriesData attribute), 43

idx(worldometer.world.geography.largest\_countries.LargestCountriesData attribute), 45

idx(worldometer.world.population.by\_region.CurrentWorldPopulationByRegionData attribute), 21

idx(worldometer.world.population.by\_region.FutureWorldPopulationByRegionData attribute), 22

idx(worldometer.world.population.by\_region.PastWorldPopulationByRegionData attribute), 22

idx(worldometer.world.population.countries\_by\_population.CountriesByPopulationData attribute), 18

idx(worldometer.world.population.most\_populous\_countries.CurrentMostPopulousCountriesData attribute), 27

idx(worldometer.world.population.most\_populous\_countries.FutureMostPopulousCountriesData attribute), 29

idx(worldometer.world.population.most\_populous\_countries.PastMostPopulousCountriesData attribute), 28

internet\_users\_in\_the\_world\_today (worldometer.world.counters.SocietyAndMedia attribute), 11

L

land\_area (worldometer.world.geography.countries.WorldCountriesData attribute), 44

land\_area (worldometer.world.population.countries\_by\_population.CountriesByPopulationData attribute), 19

land\_area (worldometer.world.population.largest\_cities.LargestCitiesData attribute), 25

land\_area\_km2 (worldometer.world.geography.largest\_countries.LargestCountriesData attribute), 46

land\_area\_mi2 (worldometer.world.geography.largest\_countries.LargestCountriesData attribute), 46

land\_lost\_to\_soil\_erosion\_this\_year (worldometer.world.counters.Environment attribute), 11

LargestCities (class in worldometer.world.population.largest\_cities), 24

LargestCitiesData (class in worldometer.world.population.largest\_cities), 25

LargestCountries (class in worldometer.world.geography.largest\_countries), 45

LargestCountriesData (class in worldometer.world.geography.largest\_countries), 45

LatinAmericanAndTheCaribbeanCountries (class in worldometer.world.geography.countries), 41

LatinAmericanAndTheCaribbeanPopulation (class in worldometer.world.population.regions), 33

live() (worldometer.world.population.regions.AfricaPopulation method), 32

live() (worldometer.world.population.regions.AsiaPopulation method), 32

live() (worldometer.world.population.regions.EuropePopulation method), 33

live() (worldometer.world.population.regions.LatinAmericanAndTheCaribbeanPopulation method), 34

live() (worldometer.world.population.regions.NorthernAmericanPopulation method), 34

live() (worldometer.world.population.regions.OceaniaPopulation method), 35

M

median\_age (worldometer.world.population.most\_populous\_countries.CurrentMostPopulousCountriesData attribute), 22

median\_age (worldometer.world.population.countries\_by\_population.CountriesByPopulationData attribute), 19

median\_age (worldometer.world.population.regions.ForecastData attribute), 37

median\_age (worldometer.world.population.regions.HistoricalData attribute), 36

migrants (worldometer.world.population.by\_region.CurrentWorldPopulationByRegionData attribute), 21

migrants (worldometer.world.population.countries\_by\_population.CountriesByPopulationData attribute), 19

migrants (worldometer.world.population.regions.ForecastData attribute), 37

migrants (worldometer.world.population.regions.HistoricalData attribute), 36

models

worldometer, 7

worldometer.world.counters, 8

worldometer.world.country\_codes, 15

worldometer.world.geography.countries, 39 new\_column\_names (worldome-  
 worldometer.world.geography.largest\_countries, ter.world.country\_codes.CountryCodes at-  
 45 tribute), 15  
 worldometer.world.population.by\_region, new\_column\_names (worldome-  
 20 ter.world.geography.countries.AfricaCountries  
 worldometer.world.population.by\_year, 23 attribute), 40  
 worldometer.world.population.countries\_by\_population, new\_column\_names (worldome-  
 17 ter.world.geography.countries.AsiaCountries  
 worldometer.world.population.largest\_cities, attribute), 40  
 24 new\_column\_names (worldome-  
 worldometer.world.population.most\_populous\_countries, ter.world.geography.countries.EuropeCountries  
 26 attribute), 41  
 worldometer.world.population.projections, new\_column\_names (worldome-  
 29 ter.world.geography.countries.LatinAmericanAndTheCaribbeanC  
 worldometer.world.population.regions, 31 attribute), 42  
 money\_spent\_for\_obesity\_related\_diseases\_in\_the\_usa\_today, new\_column\_names (worldome-  
 (worldometer.world.counters.Food attribute), ter.world.geography.countries.NorthernAmericanCountries  
 12 attribute), 42  
 money\_spent\_on\_illegal\_drugs\_this\_year (worl- new\_column\_names (worldome-  
 dometer.world.counters.Health attribute), 15 ter.world.geography.countries.OceaniaCountries  
 money\_spent\_on\_videogames\_today (worldome- attribute), 43  
 ter.world.counters.SocietyAndMedia attribute), new\_column\_names (worldome-  
 11 ter.world.geography.countries.WorldCountries  
 money\_spent\_on\_weight\_loss\_programs\_in\_the\_usa\_today attribute), 39  
 (worldometer.world.counters.Food attribute), new\_column\_names (worldome-  
 12 ter.world.geography.largest\_countries.LargestCountries  
 MostPopulousCountries (class in worldome- attribute), 45  
 ter.world.population.most\_populous\_countries), new\_column\_names (worldome-  
 26 ter.world.population.by\_region.WorldPopulationByRegion  
 attribute), 20  
**N** new\_column\_names (worldome-  
 natural\_gas\_left (worldome- ter.world.population.by\_year.WorldPopulationByYear  
 ter.world.counters.Energy attribute), 13 attribute), 23  
 net\_change (worldome- new\_column\_names (worldome-  
 ter.world.population.by\_region.CurrentWorldPopulationByRegionData ter.world.population.countries\_by\_population.CountriesByPopul  
 attribute), 21 attribute), 17  
 net\_change (worldome- new\_column\_names (worldome-  
 ter.world.population.by\_year.WorldPopulationByYearData ter.world.population.largest\_cities.LargestCities  
 attribute), 24 attribute), 25  
 net\_change (worldome- new\_column\_names (worldome-  
 ter.world.population.countries\_by\_population.CountriesByPopul ter.world.population.most\_populous\_countries.MostPopulousCou  
 attribute), 18 attribute), 26  
 net\_change (worldome- new\_column\_names (worldome-  
 ter.world.population.projections.WorldPopulationProjections ter.world.population.projections.WorldPopulationProjections  
 attribute), 30 attribute), 29  
 net\_population\_growth\_this\_year (worldome- new\_column\_names (worldome-  
 ter.world.counters.WorldPopulation attribute), ter.world.population.regions.AfricaPopulation  
 10 attribute), 32  
 net\_population\_growth\_today (worldome- new\_column\_names (worldome-  
 ter.world.counters.WorldPopulation attribute), ter.world.population.regions.AsiaPopulation  
 10 attribute), 31  
 new\_book\_titles\_published\_this\_year (worl- new\_column\_names (worldome-  
 dometer.world.counters.SocietyAndMedia ter.world.population.regions.EuropePopulation  
 attribute), 10 attribute), 33



new\_column\_names (worldometer.world.population.regions.LatinAmericanAndTheCaribbeanPopulation attribute), 44

new\_column\_names (worldometer.world.population.regions.NorthernAmericanPopulation attribute), 34

new\_column\_names (worldometer.world.population.regions.OceaniaPopulation attribute), 35

newspapers\_circulated\_today (worldometer.world.counters.SocietyAndMedia attribute), 11

non\_renewable\_sources (worldometer.world.counters.Energy attribute), 13

NorthernAmericanCountries (class in worldometer.world.geography.countries), 42

NorthernAmericanPopulation (class in worldometer.world.population.regions), 34

## O

obese\_people\_in\_the\_world (worldometer.world.counters.Food attribute), 12

OceaniaCountries (class in worldometer.world.geography.countries), 43

OceaniaPopulation (class in worldometer.world.population.regions), 34

oil\_left (worldometer.world.counters.Energy attribute), 13

oil\_pumped\_today (worldometer.world.counters.Energy attribute), 13

overweight\_people\_in\_the\_world (worldometer.world.counters.Food attribute), 12

## P

past() (worldometer.world.population.by\_region.WorldPopulationByRegion method), 20

past() (worldometer.world.population.most\_populous\_countries.MostPopulousCountries method), 27

PastMostPopulousCountriesData (class in worldometer.world.population.most\_populous\_countries), 28

PastWorldPopulationByRegionData (class in worldometer.world.population.by\_region), 22

people\_who\_died\_of\_hunger\_today (worldometer.world.counters.Food attribute), 12

people\_with\_no\_access\_to\_a\_safe\_drinking\_water\_source (worldometer.world.counters.Water attribute), 12

percentage\_of\_world\_landmass (worldometer.world.geography.largest\_countries.LargestCountriesData attribute), 46

population (worldometer.world.geography.countries.CountryData attribute), 44

population (worldometer.world.geography.countries.DependencyData attribute), 44

population (worldometer.world.geography.countries.WorldCountriesData attribute), 44

population (worldometer.world.population.by\_region.CurrentWorldPopulationByRegionData attribute), 21

population (worldometer.world.population.by\_region.FutureWorldPopulationByRegionData attribute), 23

population (worldometer.world.population.by\_region.PastWorldPopulationByRegionData attribute), 22

population (worldometer.world.population.countries\_by\_population.CountriesByPopulation attribute), 18

population (worldometer.world.population.most\_populous\_countries.CurrentMostPopulous attribute), 27

population (worldometer.world.population.most\_populous\_countries.FutureMostPopulous attribute), 29

population (worldometer.world.population.most\_populous\_countries.PastMostPopulous attribute), 28

population (worldometer.world.population.regions.ForecastData attribute), 37

population (worldometer.world.population.regions.HistoricalData attribute), 36

population (worldometer.world.population.regions.SubregionData attribute), 35

population\_estimate (worldometer.world.population.largest\_cities.LargestCitiesData attribute), 25

public\_education\_expenditure\_today (worldometer.world.counters.GovernmentAndEconomics attribute), 10

public\_healthcare\_expenditure\_today (worldometer.world.counters.GovernmentAndEconomics attribute), 10

public\_military\_expenditure\_today (worldometer.world.counters.GovernmentAndEconomics attribute), 10

## R

rank (worldometer.world.population.largest\_cities.LargestCitiesData attribute), 25

`rank` (`worldometer.world.population.most_populous_countries.ForecastData` (worldometer.world.geography.countries.WorldCountries attribute), 29  
`rank` (`worldometer.world.population.most_populous_countries.PastMostPopulousCountriesData` (worldometer.world.geography.countries.WorldCountries attribute), 28  
`rank` (`worldometer.world.population.regions.ForecastData` (worldometer.world.geography.largest\_countries.LargestCountries attribute), 38  
`rank` (`worldometer.world.population.regions.HistoricalData` (worldometer.world.geography.largest\_countries.LargestCountries attribute), 36  
`region` (`worldometer.world.population.by_region.CurrentWorldPopulationByRegionData` (worldometer.world.population.by\_region.WorldPopulationByRegion attribute), 21  
`region` (`worldometer.world.population.by_region.FutureWorldPopulationByRegionData` (worldometer.world.population.by\_region.WorldPopulationByRegion attribute), 23  
`region` (`worldometer.world.population.by_region.PastWorldPopulationByRegionData` (worldometer.world.population.by\_region.WorldPopulationByRegion attribute), 22  
`renewable_sources` (worldometer.world.counters.Energy attribute), 13  
`road_traffic_accident_fatalities_this_year` (worldometer.world.counters.Health attribute), 15

## S

`seasonal_flu_deaths_this_year` (worldometer.world.counters.Health attribute), 14  
`society_and_media` (worldometer.world.counters.WorldCounters attribute), 9  
`SocietyAndMedia` (class in worldometer.world.counters), 10  
`solar_energy_striking_earth_today` (worldometer.world.counters.Energy attribute), 13  
`source_path` (worldometer.world.counters.WorldCounters attribute), 8  
`source_path` (worldometer.world.country\_codes.CountryCodes attribute), 15  
`source_path` (worldometer.world.geography.countries.AfricaCountries attribute), 40  
`source_path` (worldometer.world.geography.countries.AsiaCountries attribute), 40  
`source_path` (worldometer.world.geography.countries.EuropeCountries attribute), 41  
`source_path` (worldometer.world.geography.countries.LatinAmericanAndTheCaribbeanCountries attribute), 41  
`source_path` (worldometer.world.geography.countries.NorthernAmericanCountries attribute), 42  
`source_path` (worldometer.world.geography.countries.OceaniaCountries attribute), 43  
`source_path` (worldometer.world.population.most\_populous\_countries.MostPopulousCountries attribute), 26  
`source_path` (worldometer.world.population.projections.WorldPopulationProjections attribute), 29  
`source_path` (worldometer.world.population.regions.AfricaPopulation attribute), 32  
`source_path` (worldometer.world.population.regions.AsiaPopulation attribute), 31  
`source_path` (worldometer.world.population.regions.EuropePopulation attribute), 33  
`source_path` (worldometer.world.population.regions.LatinAmericanAndTheCaribbeanPopulation attribute), 33  
`source_path` (worldometer.world.population.regions.NorthernAmericanPopulation attribute), 34  
`source_path` (worldometer.world.population.regions.OceaniaPopulation attribute), 35  
`subregion` (worldometer.world.geography.countries.CountryData attribute), 44  
`SubregionData` (class in worldometer.world.population.regions), 35  
`subregions()` (worldometer.world.population.regions.AfricaPopulation method), 32  
`subregions()` (worldometer.world.population.regions.AsiaPopulation method), 32  
`subregions()` (worldometer.world.population.regions.OceaniaPopulation method), 32

<i>ter.world.population.regions.EuropePopulation</i> <i>method</i> ), 33	<i>two_letter_iso</i> <i>ter.world.country_codes.CountryCodesData</i> <i>attribute</i> ), 16
<i>subregions()</i> <i>ter.world.population.regions.LatinAmericanAndTheCaribbeanPopulation</i> <i>method</i> ), 34	<b>U</b>
<i>subregions()</i> <i>ter.world.population.regions.NorthernAmericanPopulation</i> <i>method</i> ), 34	<i>undernourished_people_in_the_world</i> ( <i>worldome-</i> <i>ter.world.counters.Food</i> <i>attribute</i> ), 12
<i>subregions()</i> <i>ter.world.population.regions.OceaniaPopulation</i> <i>method</i> ), 35	<i>urban_area</i> ( <i>worldome-</i> <i>ter.world.population.largest_cities.LargestCitiesData</i> <i>attribute</i> ), 25
<i>suicides_this_year</i> ( <i>worldome-</i> <i>ter.world.counters.Health</i> <i>attribute</i> ), 14	<i>urban_population</i> ( <i>worldome-</i> <i>ter.world.population.by_region.CurrentWorldPopulationByRegion</i> <i>attribute</i> ), 22
<b>T</b>	<i>urban_population</i> ( <i>worldome-</i> <i>ter.world.population.countries_by_population.CountriesByPopula-</i> <i>attribute</i> ), 19
<i>territory</i> ( <i>worldome-</i> <i>ter.world.geography.countries.DependencyData</i> <i>attribute</i> ), 44	<i>urban_population</i> ( <i>worldome-</i> <i>ter.world.population.regions.ForecastData</i> <i>attribute</i> ), 37
<i>three_digit_iso_numeric</i> ( <i>worldome-</i> <i>ter.world.country_codes.CountryCodesData</i> <i>attribute</i> ), 16	<i>urban_population</i> ( <i>worldome-</i> <i>ter.world.population.regions.HistoricalData</i> <i>attribute</i> ), 36
<i>three_letter_iso</i> ( <i>worldome-</i> <i>ter.world.country_codes.CountryCodesData</i> <i>attribute</i> ), 16	<i>urban_population_percent</i> ( <i>worldome-</i> <i>ter.world.population.regions.ForecastData</i> <i>attribute</i> ), 37
<i>total</i> ( <i>worldometer.world.geography.countries.AfricaCountries</i> <i>attribute</i> ), 40	<i>urban_population_percent</i> ( <i>worldome-</i> <i>ter.world.population.regions.HistoricalData</i> <i>attribute</i> ), 36
<i>total</i> ( <i>worldometer.world.geography.countries.AsiaCountries</i> <i>attribute</i> ), 40	<b>W</b>
<i>total</i> ( <i>worldometer.world.geography.countries.EuropeCountries</i> <i>attribute</i> ), 41	<i>water</i> ( <i>class in worldometer.world.counters</i> ), 12
<i>total</i> ( <i>worldometer.world.geography.countries.LatinAmericanAndTheCaribbeanCountries</i> <i>attribute</i> ), 42	<i>water</i> ( <i>worldometer.world.counters.WorldCounters</i> <i>at-</i> <i>tribute</i> ), 9
<i>total</i> ( <i>worldometer.world.geography.countries.NorthernAmericanCountries</i> <i>attribute</i> ), 42	<i>water_used_this_year</i> ( <i>worldome-</i> <i>ter.world.counters.Water</i> <i>attribute</i> ), 12
<i>total</i> ( <i>worldometer.world.geography.countries.OceaniaCountries</i> <i>attribute</i> ), 43	<i>world_population</i> ( <i>worldome-</i> <i>ter.world.counters.WorldCounters</i> <i>attribute</i> ), 8
<i>total</i> ( <i>worldometer.world.geography.countries.WorldCountries</i> <i>attribute</i> ), 39	<i>world_population</i> ( <i>worldome-</i> <i>ter.world.population.by_year.WorldPopulationByYearData</i> <i>attribute</i> ), 24
<i>total_area_km2</i> ( <i>worldome-</i> <i>ter.world.geography.largest_countries.LargestCountriesData</i> <i>attribute</i> ), 46	<i>world_population</i> ( <i>worldome-</i> <i>ter.world.population.projections.WorldPopulationProjectionsData</i> <i>attribute</i> ), 30
<i>total_area_mi2</i> ( <i>worldome-</i> <i>ter.world.geography.largest_countries.LargestCountriesData</i> <i>attribute</i> ), 46	<i>world_population</i> ( <i>worldome-</i> <i>ter.world.population.regions.ForecastData</i> <i>attribute</i> ), 38
<i>toxic_chemicals_released_in_the_environment_this_year</i> ( <i>worldometer.world.counters.Environment</i> <i>at-</i> <i>tribute</i> ), 12	<i>world_population</i> ( <i>worldome-</i> <i>ter.world.population.regions.HistoricalData</i> <i>attribute</i> ), 36
<i>tv_sets_sold_worldwide_today</i> ( <i>worldome-</i> <i>ter.world.counters.SocietyAndMedia</i> <i>attribute</i> ), 11	<i>world_share</i> ( <i>worldome-</i> <i>ter.world.geography.countries.WorldCountriesData</i> <i>attribute</i> ), 44
<i>tweets_sent_today</i> ( <i>worldome-</i> <i>ter.world.counters.SocietyAndMedia</i> <i>attribute</i> ), 11	



world\_share (worldometer.world.population.regions.  
 ter.world.population.by\_region.CurrentWorldPopulationByRegionData  
 attribute), 22  
 world\_share (worldometer.world.population.regions.  
 ter.world.population.by\_region.FutureWorldPopulationByRegionData  
 attribute), 23  
 world\_share (worldometer.world.population.regions.  
 ter.world.population.by\_region.PastWorldPopulationByRegionData  
 attribute), 22  
 world\_share (worldometer.world.population.by\_year.  
 ter.world.population.countries\_by\_population.CountriesByPopulationData  
 attribute), 19  
 world\_share (worldometer.world.population.projections.  
 ter.world.population.projections.WorldPopulationProjectionsData  
 attribute), 27  
 world\_share (worldometer.world.population.projections.  
 ter.world.population.projections.WorldPopulationProjectionsData  
 attribute), 29  
 world\_share (worldometer.world.population.projections.  
 ter.world.population.projections.WorldPopulationProjectionsData  
 attribute), 28  
 world\_share (worldometer.world.population.projections.  
 ter.world.population.projections.WorldPopulationProjectionsData  
 attribute), 38  
 world\_share (worldometer.world.population.projections.  
 ter.world.population.projections.WorldPopulationProjectionsData  
 attribute), 36  
 WorldCounters (class in worldometer.world.counters), 8  
 WorldCountries (class in worldometer.world.geography.countries), 39  
 WorldCountriesData (class in worldometer.world.geography.countries), 43  
 worldometer  
 module, 7  
 worldometer.world.counters  
 module, 8  
 worldometer.world.country\_codes  
 module, 15  
 worldometer.world.geography.countries  
 module, 39  
 worldometer.world.geography.largest\_countries  
 module, 45  
 worldometer.world.population.by\_region  
 module, 20  
 worldometer.world.population.by\_year  
 module, 23  
 worldometer.world.population.countries\_by\_population  
 module, 17  
 worldometer.world.population.largest\_cities  
 module, 24  
 worldometer.world.population.most\_populous\_countries  
 module, 26  
 worldometer.world.population.projections  
 module, 29