**evo.gen**

Quick Reference Guide

|  |  |
| --- | --- |
| version 0.0 | Viktor Milkevych |

CONTENTS

[class POPULATION 4](#_Toc201219688)

[Population 4](#_Toc201219689)

[set\_population 4](#_Toc201219690)

[set\_population 5](#_Toc201219691)

[aging 5](#_Toc201219692)

[size 6](#_Toc201219693)

[capacity 6](#_Toc201219694)

[clear 6](#_Toc201219695)

[reshape 7](#_Toc201219696)

[id\_at 7](#_Toc201219697)

[id\_at 8](#_Toc201219698)

[sire\_at 8](#_Toc201219699)

[sire\_at 9](#_Toc201219700)

[dame\_at 9](#_Toc201219701)

[dame\_at 9](#_Toc201219702)

[age\_at 10](#_Toc201219703)

[age\_at 10](#_Toc201219704)

[alive\_at 11](#_Toc201219705)

[alive\_at 11](#_Toc201219706)

[isgenotyped\_at 12](#_Toc201219707)

[isgenotyped\_at 12](#_Toc201219708)

[sex\_at 12](#_Toc201219709)

[sex\_at 13](#_Toc201219710)

[phenotype\_at 13](#_Toc201219711)

[phenotype\_at 14](#_Toc201219712)

[breedingvalue\_at 14](#_Toc201219713)

[breedingvalue\_at 15](#_Toc201219714)

[get\_ld 15](#_Toc201219715)

[get\_popid 16](#_Toc201219716)

[class GROUP 17](#_Toc201219717)

[Group 17](#_Toc201219718)

[size 17](#_Toc201219719)

[size\_at 18](#_Toc201219720)

[clear 18](#_Toc201219721)

[remove 19](#_Toc201219722)

[move 19](#_Toc201219723)

[add 20](#_Toc201219724)

[add 20](#_Toc201219725)

[add 20](#_Toc201219726)

[mate 21](#_Toc201219727)

[mate 21](#_Toc201219728)

[regroup\_newborn 22](#_Toc201219729)

[aging 22](#_Toc201219730)

[genotype 23](#_Toc201219731)

[kill 23](#_Toc201219732)

[make\_observation 24](#_Toc201219733)

[make\_observation 24](#_Toc201219734)

[make\_observation 25](#_Toc201219735)

[make\_observation 26](#_Toc201219736)

[make\_observation 27](#_Toc201219737)

[select 27](#_Toc201219738)

[class TRAIT 29](#_Toc201219739)

[Trait 29](#_Toc201219740)

[Trait 29](#_Toc201219741)

[set\_trait 31](#_Toc201219742)

[reset\_trait 32](#_Toc201219743)

[get\_observations 34](#_Toc201219744)

[get\_observations 34](#_Toc201219745)

[get\_observations 35](#_Toc201219746)

[get\_observations 36](#_Toc201219747)

[get\_observations 37](#_Toc201219748)

[clear 38](#_Toc201219749)

[is\_cleared 39](#_Toc201219750)

# class POPULATION

|  |
| --- |
| Population( )  **Description** |
| Construct an empty Population object. This is a default class constructor with no parameters.  **Parameters** |
| None.  **Return value** |
| None.  **Example (use in Python)** |
| import evogen  pop = evogen.Population() # create the class instance |

|  |
| --- |
| void  set\_population (size\_t *nindividuals*,  const std::string & *genotype\_structure*,  float *ref\_allele\_probability*,  int *n\_ploidy* )  **Description** |
| Stochastically generates an initial population with no LD. The method simulates a specified number of unrelated individuals with simulated genome of properties determined by the specific parameters of the method. In order to assign an LD structure for such population it should pass some rounds of random maiting.  **Parameters** |
| *nindividuals* – a number of individuals in the population; positive integer.  *genotype\_structure* – the file name where the genome structure is described.  *ref\_allele\_probability* – the frequency of reference allele at each of loci of the simulated genome; floating point number in the interval [0,1].  *n\_ploidy* – the ploidy of the simulated genome; integer value.  **Return value** |
| None.  **Example (use in Python)** |
| import evogen  pop = evogen.Population() # create the class instance  pop.set\_population(5, "gen\_struct\_pop.dat", 0.4, 4) |

|  |
| --- |
| void  set\_population ( const std::string & *haplotypes\_fname*,  const std::string & *genotype\_structure*,  bool *with\_pedigree* )  **Description** |
| Generaates population using an existing data of haplotypes (simulated or real). The method uses a predefined (provided in a data file) haplotypes, genome structure and pedigree (optional).  **Parameters** |
| *haplotypes\_fname* – the file name where the genome’s haplotypes is provided.  *genotype\_structure* – the file name where the genome structure is described.  *with\_pedigree* – TRUE or FALSE indicating whether to include or to not include an initial pedigree information (provided in a haplotypes data file) in a generated population.  **Return value** |
| None.  **Example**  **(use in Python)** |
| import evogen  pop = evogen.Population() # create the class instance  pop.set\_population("haplotypes\_pop.dat", "gen\_struct\_pop.dat", TRUE) |

|  |
| --- |
| void  aging ( int *delta\_t* )  **Description** |
| Increase the age of every individual in a population:  new\_age = current\_age + delta\_t.  **Parameters** |
| *delta\_t* - the amount of time added to the current age of a particular individual; the time unit is arbitrary.  **Return value** |
| None.  **Example (use in Python)** |
| import evogen  pop = evogen.Population() # create the class instance  pop.set\_population(5, “gen\_struct\_pop.dat”, 0.4, 2) # simulate population  pop.aging(3) |

|  |
| --- |
| size\_t  size ( )  **Description** |
| Get the current size of a population in terms of number of active (alive) individuals.  **Parameters** |
| None.  **Return value** |
| Positive integer value.  **Example (use in Python)** |
| import evogen  pop = evogen.Population() # create the class instance  pop.set\_population(5, “gen\_struct\_pop.dat”, 0.4, 2) # simulate population  n\_individ = pop.size() |

|  |
| --- |
| size\_t  capacity ( )  **Description** |
| Get the entire size of a population in terms of number of all individuals (alive + disabled).  **Parameters** |
| None.  **Return value** |
| Positive integer value.  **Example (use in Python)** |
| import evogen  pop = evogen.Population() # create the class instance  pop.set\_population(5, “gen\_struct\_pop.dat”, 0.4, 2) # simulate population  n\_individ = pop.capacity() |

|  |
| --- |
| void  clear ( )  **Description** |
| Removes all active individuals from a population and clears the allocated memory.  **Parameters** |
| None.  **Return value** |
| None.  **Example (use in Python)** |
| import evogen  pop = evogen.Population() # create the class instance  pop.set\_population(5, “gen\_struct\_pop.dat”, 0.4, 2) # simulate population  pop.clear() |

|  |
| --- |
| void  reshape ( )  **Description** |
| Optimizes the memory allocated to a population by clearing the memory allocated for inactive (disabled) individuals and rearrannging the list of individuals in a population. After the call of this method the positions of individuals in the population might be changed. Therefore, any existing association with a specific Group (see the methods for the class GROUP) will be disabled.  **Parameters** |
| None.  **Return value** |
| None.  **Example (use in Python)** |
| import evogen  pop = evogen.Population() # create the class instance  pop.set\_population(5, “gen\_struct\_pop.dat”, 0.4, 2) # simulate population  for i in range( pop.size ):  if i%2 == 0:  pop.alive\_at(i, false) # disable every second individual in the population pop  pop.reshape() |

|  |
| --- |
| void  id\_at (size\_t *at*, unsigned long *id* )  **Description** |
| Assign a specific identity number (ID) to a specific individual in a population. This method overwrites a simulated ID by a specific one defined by the user.  **Parameters** |
| *at* – the position of an individual in the population’s list of individuals.  *id* – positive integer value will be assigned to the individual’s ID.  **Return value** |
| None.  **Example (use in Python)** |
| import evogen  import random  pop = evogen.Population() # create the class instance  pop.set\_population(5, “gen\_struct\_pop.dat”, 0.4, 2) # simulate population  for i in range( pop.size ):  r\_num = random.randint(0, 1000)  pop.id\_at(i, r\_num) |

|  |
| --- |
| unsigned long  id\_at (size\_t *at* )  **Description** |
| Get (retrieve) an identity number (ID) of a specific individual in a population.  **Parameters** |
| *at* – the position of an individual in the population’s list of individuals.  **Return value** |
| Positive integer value, an individual’s ID.  **Example (use in Python)** |
| import evogen  pop = evogen.Population() # create the class instance  pop.set\_population(5, “gen\_struct\_pop.dat”, 0.4, 2) # simulate population  id = pop.id\_at(0) # ID of the individual at the position 0 in the population. |

|  |
| --- |
| void  sire\_at (size\_t *at*, unsigned long *id* )  **Description** |
| Assign a sire to a specific individual in a population. This method overwrites a simulated ID by a specific one defined by the user.  **Parameters** |
| *at* – the position of an individual in the population’s list of individuals.  *id* – positive integer value will be assigned to the individual’s property named as sire.  **Return value** |
| None.  **Example (use in Python)** |
| import evogen  pop = evogen.Population() # create the class instance  pop.set\_population(5, “gen\_struct\_pop.dat”, 0.4, 2) # simulate population  pop.sire\_at(0, 10089) # individual at the position 0 will get new sire with id 10089 |
| unsigned long  sire\_at ( size\_t *at* )  **Description** |
| Get (retrieve) a sire’s identity number (ID) of a specific individual in a population.  **Parameters** |
| *at* – the position of an individual in the population’s list of individuals.  **Return value** |
| Positive integer value.  **Example (use in Python)** |
| import evogen  pop = evogen.Population() # create the class instance  pop.set\_population(5, “gen\_struct\_pop.dat”, 0.4, 2) # simulate population  sire\_id = pop.sire\_at(0) # sire’s ID of the individual at the position 0 in the population. |

|  |
| --- |
| void  dame\_at ( size\_t *at*, unsigned long *id* )  **Description** |
| Assign a dame to a specific individual in a population. This method overwrites a simulated ID by a specific one defined by the user.  **Parameters** |
| *at* – the position of an individual in the population’s list of individuals.  *id* – positive integer value will be assigned to the individual’s property named as dame.  **Return value** |
| None.  **Example (use in Python)** |
| import evogen  pop = evogen.Population() # create the class instance  pop.set\_population(5, “gen\_struct\_pop.dat”, 0.4, 2) # simulate population  pop.dame\_at(0, 10089) # individual at the position 0 will get new dame with id 10089 |

|  |
| --- |
| unsigned long  dame\_at ( size\_t *at* )  **Description** |
| Get (retrieve) a dame’s identity number (ID) of a specific individual in a population.  **Parameters** |
| *at* – the position of an individual in the population’s list of individuals.  **Return value** |
| Positive integer value.  **Example (use in Python)** |
| import evogen  pop = evogen.Population() # create the class instance  pop.set\_population(5, “gen\_struct\_pop.dat”, 0.4, 2) # simulate population  dame\_id = pop.dame\_at(0) # dame’s ID of the individual at the position 0 in the population. |

|  |
| --- |
| void  age\_at ( size\_t *at*, int *age* )  **Description** |
| Assign (modify) an age for a specific individual in a population.  **Parameters** |
| *at* – the position of an individual in the population’s list of individuals,  *age* – (positive or negative) integer value to be a new age of the individual.  **Return value** |
| None.  **Example (use in Python)** |
| import evogen  pop = evogen.Population() # create the class instance  pop.set\_population(5, “gen\_struct\_pop.dat”, 0.4, 2) # simulate population  pop.age\_at(0, 23) # individual at the position 0 will get new age of 23. |

|  |
| --- |
| int  age\_at ( size\_t *at* )  **Description** |
| Get (retrieve) an age of a specific individual in the population.  **Parameters** |
| *at* – the position of an individual in the population’s list of individuals.  **Return value** |
| Positive or negative integer value.  **Example (use in Python)** |
| import evogen  pop = evogen.Population() # create the class instance  pop.set\_population(5, “gen\_struct\_pop.dat”, 0.4, 2) # simulate population  age = pop.age\_at(0) # age of the individual at the position 0. |

|  |
| --- |
| void  alive\_at ( size\_t *at*, bool *alive* )  **Description** |
| Set (modify) the living status of a specific individual in the population.  **Parameters** |
| *at* – the position of an individual in the population’s list of individuals.  *alive* – TRUE of FALSE indicating the new status.  **Return value** |
| None.  **Example (use in Python)** |
| import evogen  pop = evogen.Population() # create the class instance  pop.set\_population(5, “gen\_struct\_pop.dat”, 0.4, 2) # simulate population  pop.alive\_at(0, FALSE) # the individual at the position 0 is disabled. |

|  |
| --- |
| bool  alive\_at ( size\_t *at* )  **Description** |
| Get (retrieve) the living status of a specific individual in the population.  **Parameters** |
| *at* – the position of an individual in the population’s list of individuals.  **Return value** |
| Boolean (TRUE of FALSE) value indicating the individual’s status.  **Example (use in Python)** |
| import evogen  pop = evogen.Population() # create the class instance  pop.set\_population(5, “gen\_struct\_pop.dat”, 0.4, 2) # simulate population  is\_active = pop.alive\_at(0) # checking if the individual at the position 0 is active or disabled. |

|  |
| --- |
| void  isgenotyped\_at ( size\_t *at*, bool *genotyped* )  **Description** |
| Sets (modify) the genotyping status of a specific individual in the population.  **Parameters** |
| *at* – the position of an individual in the population’s list of individuals.  *genotyped* – TRUE of FALSE indicating the new status.  **Return value** |
| None.  **Example (use in Python)** |
| import evogen  pop = evogen.Population() # create the class instance  pop.set\_population(5, “gen\_struct\_pop.dat”, 0.4, 2) # simulate population  pop.isgenotyped\_at(0, TRUE) # the individual at the position 0 is genotyped. |

|  |
| --- |
| bool  isgenotyped\_at ( size\_t *at* )  **Description** |
| Get (retrieve) the genotyping status of a specific individual in the population.  **Parameters** |
| *at* – the position of an individual in the population’s list of individuals.  **Return value** |
| Boolean (TRUE of FALSE) value indicating the individual’s status.  **Example (use in Python)** |
| import evogen  pop = evogen.Population() # create the class instance  pop.set\_population(5, “gen\_struct\_pop.dat”, 0.4, 2) # simulate population  is\_genotyped = pop.isgenotyped\_at(0) # checking if the individual at the position 0 is genotyped. |

|  |
| --- |
| void  sex\_at ( size\_t *at*, int *sex* )  **Description** |
| Assign (modify) a sex of a specific individual in a population. This method overwrite the existing sex of an individual.  **Parameters** |
| *at* – the position of an individual in the population’s list of individuals,  *sex* – (positive/negative) integer value to be a new sex of the individual.  **Return value** |
| None.  **Example (use in Python)** |
| import evogen  pop = evogen.Population() # create the class instance  pop.set\_population(5, “gen\_struct\_pop.dat”, 0.4, 2) # simulate population  pop.sex\_at(0, 1) |

|  |
| --- |
| short  sex\_at ( size\_t *at* )  **Description** |
| Get (retrieve) the sex of a specific individual in the population.  **Parameters** |
| *at* – the position of an individual in the population’s list of individuals.  **Return value** |
| Positive or negative integer value.  **Example (use in Python)** |
| import evogen  pop = evogen.Population() # create the class instance  pop.set\_population(5, “gen\_struct\_pop.dat”, 0.4, 2) # simulate population  sex = pop.sex\_at(0) # gets a sex value of the individual at the position 0. |

|  |
| --- |
| void  phenotype\_at ( size\_t *at*, pybind11::array\_t<float> *phen* )  **Description** |
| Assign (modify) phenotypic observations (trait values) for a specific individual in a population.  **Parameters** |
| *at* – the position of an individual in the population’s list of individuals,  *phen* – a python array (list or numpy array) of a floating point numbes representing phenotypic observations. The size of *phen* should be equal to the number of traits assigned to the population.  **Return value** |
| None.  **Example (use in Python)** |
| import evogen  pop = evogen.Population() # create the class instance  pop.set\_population(5, “gen\_struct\_pop.dat”, 0.4, 2) # simulate population  obs = [ 40.0, 5.0, 0.5 ] # three traits observation  pop.phenotype\_at(0, obs) # modify observations for the individual at the position 0. |

|  |
| --- |
| pybind11::array\_t<float>  phenotype\_at ( size\_t *at* )  **Description** |
| Get (retrieve) phenotypic observations (trait values) for a specific individual in a population.  **Parameters** |
| *at* – the position of an individual in the population’s list of individuals.  **Return value** |
| A numpy array of an individuals phenotype of floating point numbes.  **Example (use in Python)** |
| import evogen  pop = evogen.Population() # create the class instance  pop.set\_population(5, “gen\_struct\_pop.dat”, 0.4, 2) # simulate population  in\_obs = [ 40.0, 5.0, 0.5 ] # three traits observation  pop.phenotype\_at(0, in\_obs) # modify observations for the individual at the position 0  out\_obs = pop.phenotype\_at(0) # get observations of the individual at the position 0. |

|  |
| --- |
| void  breedingvalue\_at ( size\_t *at*, pybind11::array\_t<float> *bv* )  **Description** |
| Assign (modify) breeding values for a specific individual in a population.  **Parameters** |
| *at* – the position of an individual in the population’s list of individuals.  *bv* – a python array (list or numpy array) of a floating point numbes of breeding values. The size of bv should be equal to the number of traits assigned to the population.  **Return value** |
| None.  **Example (use in Python)** |
| import evogen  pop = evogen.Population() # create the class instance  pop.set\_population(5, “gen\_struct\_pop.dat”, 0.4, 2) # simulate population  bvs = [ 0.01, 0.05, -0.5 ] # three traits observation  pop.breedingvalue\_at(0, bvs) # modify BVs for the individual at the position 0. |

|  |
| --- |
| pybind11::array\_t<float>  breedingvalue\_at ( size\_t *at* )  **Description** |
| Get (retrieve) breeding values for a specific individual in a population.  **Parameters** |
| *at* – the position of an individual in the population’s list of individuals.  **Return value** |
| A numpy array of a breeding values of an individuaals. Floating point numbes.  **Example (use in Python)** |
| import evogen  pop = evogen.Population() # create the class instance  pop.set\_population(5, “gen\_struct\_pop.dat”, 0.4, 2) # simulate population  in\_bvs = [ 0.01, 0.05, -0.5 ] # three traits observation  pop.breedingvalue\_at(0, in\_bvs) # modify BVs for the individual at the position 0.  out\_bvs = pop.breedingvalue\_at(0) # get BVs of the individual at the position 0. |

|  |
| --- |
| void  get\_ld (const std::string &*out\_file*, bool *full\_info*, int *which\_chr*, unsigned int *snp\_step*)  **Description** |
| Get (retrieve) linkage disequilibrium (LD) data for a population.  **Parameters** |
| *out\_file* – the name of ASCII file where LD data will be writen.  *full\_info* – logical variable indicating if only the averaged LD data will be prvided (*full\_info = False*) or, besides the averaged LD, variants frequencies and LD matrix will be provided as well (*full\_info = True*).  *which\_chr* – specific chromosome number for which the LD should be calculated; if *which\_chr = -1*, the LD data will be provided for all available chromosomes.  *snp\_step* – SNP interval for calculating paired correlations; if *snp\_step = 1*, the correlations will be calculated for each pair of snp in a chromosome; if *snp\_step = 2*, the correlation will be calculated foreach second pair of snp, etc.  **Return value** |
| None.  **Example (use in Python)** |
| import evogen  pop = evogen.Population() # create the class instance  pop.set\_population(5, “gen\_struct\_pop.dat”, 0.4, 2) # simulate population  pop.get\_ld("ld.txt", False, 0, 10) # prints only averaged LD to ld.txt for 1st chromosome with correlation between every 10th snp |

|  |
| --- |
| size\_t  get\_popid ()  **Description** |
| Get (retrieve) unique population ID number.  **Parameters** |
| None.  **Return value** |
| Positive integer.  **Example (use in Python)** |
| import evogen  pop = evogen.Population() # create the class instance  pop.set\_population(5, “gen\_struct\_pop.dat”, 0.4, 2) # simulate population  pop\_id = pop.get\_popid() |

# class GROUP

|  |
| --- |
| Group ( )  **Description** |
| Construct an empty Group object. This is a default class constructor with no parameters.  **Parameters** |
| None.  **Return value** |
| None.  **Example (use in Python)** |
| import evogen  grp = evogen.Group() # create the class instance |

|  |
| --- |
| size\_t  size ( )  **Description** |
| Return the number of distinct populations consisting the group.  **Parameters** |
| None.  **Return value** |
| Positive integer value.  **Example (use in Python)** |
| import evogen  pop = evogen.Population() # create the class instance  pop.set\_population(5, “gen\_struct\_pop.dat”, 0.4, 2) # simulate population  grp = evogen.Group() # create the class instance  grp.add( pop ) # add population pop to the group grp  sz = grp.size( ) # will return 1 |

|  |
| --- |
| size\_t  size\_at( size\_t *at* )  **Description** |
| Returns the number of individuals in the group belonnging to a population at the position *at* in the group.  **Parameters** |
| *at* – position of a specific population in the group.  **Return value** |
| Positive integer value.  **Example (use in Python)** |
| import evogen  pop = evogen.Population() # create the class instance  pop.set\_population(5, “gen\_struct\_pop.dat”, 0.4, 2) # simulate population  grp = evogen.Group() # create the class instance  grp.add( pop ) # add population pop to the group grp  sz = grp.size\_at(0) # will return 5 |

|  |
| --- |
| void  clear( )  **Description** |
| Clears the group and releases the ocupied memory.  **Parameters** |
| None.  **Return value** |
| None.  **Example (use in Python)** |
| import evogen  pop = evogen.Population() # create the class instance  pop.set\_population(5, “gen\_struct\_pop.dat”, 0.4, 2) # simulate population  grp = evogen.Group() # create the class instance  grp.add( pop ) # add population pop to the group grp  grp.clear( ) |

|  |
| --- |
| void  remove( )  **Description** |
| For those individuals assigned to a calling group removes them from their original populations and from the group. This method also clears the calling group.  **Parameters** |
| None.  **Return value** |
| None.  **Example (use in Python)** |
| import evogen  pop = evogen.Population() # create the class instance  pop.set\_population(5, “gen\_struct\_pop.dat”, 0.4, 2) # simulate population  grp = evogen.Group() # create the class instance  grp.add( pop ) # add population pop to the group grp  grp.remove( ) # the goup will be empty as well as the population pop |

|  |
| --- |
| void  move ( Population & *pop* )  **Description** |
| Relocate all individuals assigned to the calling group to the population *pop*. The relocated individuals will be removed from their original populations.  **Parameters** |
| *pop* – the instance of a Population class.  **Return value** |
| None.  **Example (use in Python)** |
| import evogen  pop1 = evogen.Population() # create the class instance  pop2 = evogen.Population() # create the class instance  pop1.set\_population(5, “gen\_struct\_pop.dat”, 0.4, 2) # simulate population  grp = evogen.Group() # create the class instance  grp.add(pop1) # add individuals from the population pop1 to the group grp.  grp.move(pop2) # relocate all individuals from the population pop1 to the population pop2. |

|  |
| --- |
| void  add ( Population & *pop* )  **Description** |
| Adds (assign) all individuals in the population *pop* to the group. The assigned individuals will not change their location and population status (they will still belong to the population *pop*).  **Parameters** |
| *pop* – the instance of a Population class.  **Return value** |
| None.  **Example (use in Python)** |
| import evogen  pop = evogen.Population() # create the class instance  pop.set\_population(5, “gen\_struct\_pop.dat”, 0.4, 2) # simulate population  grp = evogen.Group() # create the class instance  grp.add( pop ) # add population pop to the group grp. |

|  |
| --- |
| void  add ( Population & *pop*, size\_t *which\_one* )  **Description** |
| Adds the specific individual (determined by its position *which\_one* in the population) from the population *pop* to the group.  **Parameters** |
| *pop* – the instance of a Population class.  *which\_one* – position of an individual in the population *pop*; positive integer value.  **Return value** |
| None.  **Example (use in Python)** |
| import evogen  pop = evogen.Population() # create the class instance  pop.set\_population(5, “gen\_struct\_pop.dat”, 0.4, 2) # simulate population  grp = evogen.Group() # create the class instance  grp.add( pop, 1 ) # add individual at position 1 in the population pop to the group grp. |

|  |
| --- |
| void  add ( Group & *grp* )  **Description** |
| Adds all individuals assigned to the group *grp* to the calling group. The assigned individuals will not change their memory location and population status (belonging).  **Parameters** |
| *grp* – the instance of a Group class.  **Return value** |
| None.  **Example (use in Python)** |
| import evogen  pop = evogen.Population() # create the class instance  pop.set\_population(5, “gen\_struct\_pop.dat”, 0.4, 2) # simulate population  grp = evogen.Group() # create the class instance  grp.add( pop, 1 ) # add individual at position 1 in the population pop to the group grp.  grp2 = evogen.Group() # create the class instance  grp2.add( grp ) |

|  |
| --- |
| void  mate ( )  **Description** |
| Initiates random mating for all individuals assigned to the group. The method uses sexual reproduction with two offspring per dame and the success rate of having exactly two offsprings of 0.8.  **Parameters** |
| None.  **Return value** |
| None.  **Example (use in Python)** |
| import evogen  pop = evogen.Population() # create the class instance  pop.set\_population(5, “gen\_struct\_pop.dat”, 0.4, 2) # simulate population  grp = evogen.Group() # create the class instance  grp.add( pop ) # add population pop to the group grp.  grp.mate() |

|  |
| --- |
| void  mate ( bool *sexual\_reproduction*, int *max\_offspring*, float *success\_rate* )  **Description** |
| Initiates random mating for all individuals in the group.  **Parameters** |
| *sexual\_reproduction* – TRUE or FALSE indicationg the use of sexual reproduction.  *max\_offspring* – maximal number of offsprings per dame. Integer value.  *success\_rate* – the success rate of having exactly two offsprings. Floating point number in the range [0,1].  **Return value** |
| None.  **Example (use in Python)** |
| import evogen  pop = evogen.Population() # create the class instance  pop.set\_population(5, “gen\_struct\_pop.dat”, 0.4, 2) # simulate population  grp = evogen.Group() # create the class instance  grp.add( pop ) # add population pop to the group grp.  grp.mate(TRUE, 2, 0.8) |

|  |
| --- |
| void  regroup\_newborn ( Group & *grp* )  **Description** |
| Move (relocate) all new-born individuals assigned to the calling group to another group *grp*. The individuals will be cleared from the calling group but will retain the connections to their original populations.  **Parameters** |
| *grp* – the instance of a Group class.  **Return value** |
| None.  **Example (use in Python)** |
| import evogen  pop = evogen.Population() # create the class instance  pop.set\_population(5, “gen\_struct\_pop.dat”, 0.4, 2) # simulate population  grp = evogen.Group() # create the class instance  grp.add( pop ) # add population pop to the group grp.  grp.mate(TRUE, 2, 0.8)  grp2 = evogen.Group() # create the class instance  grp.regroup\_newborn( grp2 ) # move these new-born to the new group grp2. |

|  |
| --- |
| void  aging ( int *delta\_t* )  **Description** |
| Adds delta\_t to existing values of age for every individual assigned to the calling group.  **Parameters** |
| *delta\_t* – Integer value by which a specific age will be changed: new\_age = current\_age + delta\_t.  **Return value** |
| None.  **Example (use in Python)** |
| import evogen  pop = evogen.Population() # create the class instance  pop.set\_population(5, “gen\_struct\_pop.dat”, 0.4, 2) # simulate population  grp = evogen.Group() # create the class instance  grp.add( pop ) # add population pop to the group grp  grp.aging(2) |

|  |
| --- |
| void  genotype ( )  **Description** |
| Change a genotyped status (makes it TRUE) for each individual assigned to the calling group.  **Parameters** |
| None.  **Return value** |
| None.  **Example (use in Python)** |
| import evogen  pop = evogen.Population() # create the class instance  pop.set\_population(5, “gen\_struct\_pop.dat”, 0.4, 2) # simulate population  grp = evogen.Group() # create the class instance  grp.add( pop ) # add population pop to the group grp  grp.genotype() |

|  |
| --- |
| void  kill ( )  **Description** |
| Disables all individuals assigned to the calling group. This method changes the alive status of an individual to FALSE in their original populations.  **Parameters** |
| None.  **Return value** |
| None.  **Example (use in Python)** |
| import evogen  pop = evogen.Population() # create the class instance  pop.set\_population(5, “gen\_struct\_pop.dat”, 0.4, 2) # simulate population  grp = evogen.Group() # create the class instance  grp.add( pop ) # add population pop to the group grp  grp.kill() |

|  |
| --- |
| void  make\_observation ( Trait & *trt*, pybind11::array\_t<float> *env* )  **Description** |
| Collect phenotypic observations (calculate trait values) for all individualls assigned to the calling group.  **Parameters** |
| *trt* – the instance of the Trait class.  *env* – the array (list or numpy array) determining environmntal conditions for an observing trait. Each value in *env* is a floating point number in the range [0,1]. The size of *env* is the number of correlated traits in the *trt* object.  **Return value** |
| None.  **Example (use in Python)** |
| import evogen  pop = evogen.Population() # create the class instance  pop.set\_population(5, “gen\_struct\_pop.dat”, 0.4, 2) # simulate population  trt = evogen.Trait( … some parameters … ) # create the trait object  grp = evogen.Group() # create the class instance  grp.add( pop ) # add population pop to the group grp  env = [ 0.5, 0.0, 0.1 ] # three correlated traits  grp.make\_observation(trt, env) |

|  |
| --- |
| void  make\_observation ( Trait & *trt*,  pybind11::array\_t<float> *env*,  const std::string & *trvalues* )  **Description** |
| Collect phenotypic observations (calculate trait values) for all individualls assigned to the calling group and write out the results to a file.  **Parameters** |
| *trt* – the instance of the Trait class.  *env* – the array (list or numpy array) determining environmntal conditions determined the observed trait. Each value in *env* is a floating point number in the range [0,1]. The size of *env* is the number of correlated traits in the *trt* object.  *trvalues* – file name for output of trait values.  **Return value** |
| None.  **Example (use in Python)** |
| import evogen  pop = evogen.Population() # create the class instance  pop.set\_population(5, “gen\_struct\_pop.dat”, 0.4, 2) # simulate population  trt = evogen.Trait( … some parameters … ) # create the trait object  grp = evogen.Group() # create the class instance  grp.add( pop ) # add population pop to the group grp  env = [ 0.5, 0.0, 0.1 ] # three correlated traits  grp.make\_observation(trt, env, “observations.txt”) |

|  |
| --- |
| void  make\_observation ( Trait & *trt*,  pybind11::array\_t<float> *env*,  const std::string & *trvalues*,  const std::string & *genotypes* )  **Description** |
| Collect phenotypic observations (calculate trait values) for all individualls assigned to the calling group and write out the results in terms of calculated trait values to a file as well as the genotypes of measured individuals to the separate file.  **Parameters** |
| *trt* – the instance of the Trait class.  *env* – the array (list or numpy array) determining environmntal conditions determined the observed trait. Each value in *env* is a floating point number in the range [0,1]. The size of *env* is the number of correlated traits in the *trt* object.  *trvalues* – file name for output of trait values.  *genotypes* – file name for output of individual genotypes.  **Return value** |
| None.  **Example (use in Python)** |
| import evogen  pop = evogen.Population() # create the class instance  pop.set\_population(5, “gen\_struct\_pop.dat”, 0.4, 2) # simulate population  trt = evogen.Trait( … some parameters … ) # create the trait object  grp = evogen.Group() # create the class instance  grp.add( pop ) # add population pop to the group grp  env = [ 0.5, 0.0, 0.1 ] # three correlated traits  grp.make\_observation(trt, env, “observations.txt” , “genotypes.txt”) |

|  |
| --- |
| void  make\_observation ( Trait & *trt*,  pybind11::array\_t<float> *env*,  pybind11::array\_t<float> *trvalues* )  **Description** |
| Collect phenotypic observations (calculate trait values) for all individualls assigned to the calling group and write out the results to a python array.  **Parameters** |
| *trt* – the instance of the Trait class.  *env* – the array (list or numpy array) determining environmntal conditions determined the observed trait. Each value in *env* is a floating point number in the range [0,1]. The size of *env* is the number of correlated traits in the *trt* object.  *trvalues* – numpy array of type float32 and arbitrary shape for output of trait values..  **Return value** |
| None.  **Example (use in Python)** |
| import evogen  import numpy as np  pop = evogen.Population() # create the class instance  pop.set\_population(5, “gen\_struct\_pop.dat”, 0.4, 2) # simulate population  trt = evogen.Trait( … some parameters … ) # create the trait object  grp = evogen.Group() # create the class instance  grp.add( pop ) # add population pop to the group grp  env = [ 0.5, 0.0, 0.1 ] # three correlated traits  obs = np.zeros( shape=(1), dtype=np.float32 ) # note, the importans of the correct typpe!  grp.make\_observation(trt, env, obs) |

|  |
| --- |
| void  make\_observation ( Trait & *trt*,  pybind11::array\_t<float> *env*,  pybind11::array\_t<float> *trvalues*,  pybind11::array\_t<int> *genotypes* )  **Description** |
| Collect phenotypic observations (calculate trait values) for all individualls in the calling group and write out the results in terms of calculated trait values to a python array as well as the genotypes of measured individuals to the separate python array.  **Parameters** |
| *trt* – the instance of the Trait class.  *env* – the array (list or numpy array) determining environmntal conditions determined the observed trait. Each value in *env* is a floating point number in the range [0,1]. The size of *env* is the number of correlated traits in the *trt* object.  *trvalues* – numpy array of type *float32* and arbitrary shape for output of trait values.  *genotypes* – numpy array of type *int32* and arbitrary shape for output of individual genotypes.  **Return value** |
| None.  **Example (use in Python)** |
| import evogen  import numpy as np  pop = evogen.Population() # create the class instance  pop.set\_population(5, “gen\_struct\_pop.dat”, 0.4, 2) # simulate population  trt = evogen.Trait( … some parameters … ) # create the trait object  grp = evogen.Group() # create the class instance  grp.add( pop ) # add population pop to the group grp  env = [ 0.5, 0.0, 0.1 ] # three correlated traits  obs = np.zeros( shape=(1), dtype=np.float32 ) # note, the importans of correct typpe!  gen = np.zeros( shape=(1), dtype=np.int32 ) # note, the importans of correct typpe!  grp.make\_observation (trt, env, obs, gen) |

|  |
| --- |
| void  select (const std::string *category*, const std::string *mode*, int *num*, float *val*)  **Description** |
| Select individuals in the calling group by specifyed category and mode. After the call of this method a calling group will consist only individuals that sutisfy conditions within a specified category. Consecutive calls of this method for the same group but with different selection options (changes in specified categoty for instance) will gardually reduce a number of individuals in a calling group until it will consist only individuals sutisfied all applied selection requirenments.  **Parameters** |
| *category* – parameter specifying a selection category; a string variable. The parameter defines the category (based on specific individual’s property) by which the selection should be based. Available options for the *category* parameter:  phen (select by phenotype);  bv (select by breeding value);  age (select by age);  sex (select by sex);  alive (select by live status);  active (select by active status);  genot (select by genotyping status);  id (select by individual’s identifier)  *mode* –the mode of selection within the cpecified category; a string variable. Available options for the *mode* parameter:  rand (random selection);  best (selection of individuals having highest values of individual’s property specified by the *category* parameter);  worst (selection of individuals having smalest values of individual’s property specified by the *category* parameter).  *num* –parameter specifying an upper bound of a number of candidates to be selected; an integer variable. A number of actually selected is within the range [0, num].  *val* –auxiliary parameter; an floating point number. Specific input options for the *val* parameter depends on cpecified category. If a category parameter points to a logical property (sex, alive, active, genot), *val = {0,1}*. For a category parameter pointing to a vector-type continious numerical property (phen, bv):  *val = -1* determines the use of averaged (among vector elements) value of specified property;  *val = [0,n]* determines which element of selected propertie’s vector to be used for selection, for example if multiple correlated traits is used phen and bv are vectors, so setting val = 0 means a very first element of a specific property will be used; here n is a size of vector.  If the selection category is *id*, *val* can be arbitrary integer number.  For a category parameter pointing to a single-valued continious numerical property (age, agr):  if val = -1 or mode = rand  **Return value** |
| None. A calling group will be updated.  **Example (use in Python)** |
| import evogen  import numpy as np  pop = evogen.Population() # create the class instance  pop.set\_population(5, “gen\_struct\_pop.dat”, 0.4, 2) # simulate population  grp = evogen.Group() # create the class instance  grp.add( pop ) # add population pop to the group grp  grp.select("sex", "rand", 25, 1) # selecting 25 mails from pop randomly  grp.select("phen", "best", 10, -1) # then among selected males selecting 10 that have best phenotypes |

# class TRAIT

|  |
| --- |
| Trait ( )  **Description** |
| Construct an empty Trait object. This is a default class constructor with no parameters.  **Parameters** |
| None.  **Return value** |
| None.  **Example (use in Python)** |
| import evogen  T = evogen.Trait() |

|  |
| --- |
| Trait ( Population & *pop*,  pybind11::array\_t<float> *trmean*,  pybind11::array\_t<float> *qtl\_prop\_chrom*,  pybind11::array\_t<float> *corr\_g*,  pybind11::array\_t<float> *varr\_g*,  pybind11::array\_t<float> *corr\_e*,  pybind11::array\_t<float> *varr\_e*,  pybind11::array\_t<float> *envr*,  size\_t *dist\_model*,  pybind11::array\_t<float> *dist\_par* )  **Description** |
| Construct an object of class Trait (create the class instance) and determines the specific configuraton of correlated traits. None-default class constructor. Note, the default constructor should not be called in this case.  **Parameters** |
| *pop* – instance of the Population class.  *trmean* – python array (list or numpy array) of floating point numbers representing expected mean values of correlated traits; the number of elements in the array is a number of correlated traits.  *qtl\_prop\_chrom* - python array (list or numpy array) of floating point numbers representing the proportion of SNPs selected (at random) as QTLs; the number of elements in the array is a number of correlated traits; each value is in the range [0,1].  *corr\_g* - python array (list or numpy array) of floating point numbers representing a square matrix of genomic correlations among traits; the dimension of the matrix is n-by-n, where n is a number of correlated traits; each value is in the range [0,1].  *varr\_g* - python array (list or numpy array) of floating point numbers representing expected values genomic variances for each of correlated traits; the number of elements in the array is a number of correlated traits.  *corr\_e* - python array (list or numpy array) of floating point numbers representing a square matrix of environmental (residual) correlations among traits; the dimension of the matrix is n-by-n, where n is a number of correlated traits; each value is in the range [0,1].  *varr\_e* - python array (list or numpy array) of floating point numbers representing expected values environmental (residual) variances for each of correlated traits; the number of elements in the array is a number of correlated traits.  *envr* - python array (list or numpy array) of floating point numbers representing environtal conditions for each of correlated traits; the number of elements in the array is a number of correlated traits; each value is in the range [0,1].  *dist\_mode* – positive integer value *{1,2,3}* indicating the mode of sampling dominance effects;  *dist\_mode =* 1 – sampling from a Uniform distribution,  *dist\_mode =* 2 – sampling from a Normal distribution,  *dist\_mode =* 3 – sampling from a Gamma distribution.  *dist\_par* – python array (list or numpy array) representing specific distribution parameters used for sampling dominance effects;  if *dist\_mode = 1*, *dist\_par = [a, b]*, where *a* and *b* are floating point numbers indicating the lower and upper bounds of the Uniform distributon;  if *dist\_mode = 2*, *dist\_par = [a, b]*, where *a* and *b* are floating point numbers indicating the mean and standard deviation respectively of the Normal distribution;  if *dist\_mode = 3*, *dist\_par = [a]*, where *a* is the floating point number indicating the rate parameters of the Gamma distribution (expected value of dominance effects in loci).  **Return value** |
| None.  **Example (use in Python)** |
| import evogen  pop = evogen.Population() # create the class instance  pop.set\_population(5, “gen\_struct\_pop.dat”, 0.4, 2) # simulate population  tr\_mean = [ 40.0, 5.0, 0.5 ] # (1) trait means  qtl\_prop = [ 0.65, 0.65, 0.65, 0.65 ] # (2) proportion of snps selected as qtls  qtl\_prop = [ 1.0, 1.0, 1.0, 1.0 ]  cor\_g = [ [1.0, 0.5, 0.7], [0.5, 1.0, 0.2], [0.7, 0.2, 1.0] ] # (3) genomic correlations  cor\_e = [ [1.0, 0.3, 0.5], [0.3, 1.0, 0.4], [0.5, 0.4, 1.0] ] # (4) rsidual correlations  var\_g = [ 100.0, 10.0, 0.1 ] # (5) genomic variances  var\_e = [ 200.0, 20.0, 0.3 ] # (6) residual variances  env = [ 0.0, 0.0, 0.0 ] # (7) enviironment  # (i) For uniform distribution, model 1  k\_range\_U = [ -1.0, 1.0 ] # range of k parameter  # (ii) For normal distribution, model 2  k\_range\_N = [ 0.0, 0.5 ] # mean & std  # (iii) For gamma distribution, mdodel 3  k\_range\_G = [ 0.05 ] # expected value of k in loci; is '1/b' param. in the distribution  mode = 1  T = evogen.Trait(pop, tr\_mean, qtl\_prop, cor\_g, var\_g, cor\_e, var\_e, env, mode, k\_range\_U) |

|  |
| --- |
| void  set\_trait ( Population & *pop*,  pybind11::array\_t<float> *trmean*,  pybind11::array\_t<float> *qtl\_prop\_chrom*,  pybind11::array\_t<float> *corr\_g*,  pybind11::array\_t<float> *varr\_g*,  pybind11::array\_t<float> *corr\_e*,  pybind11::array\_t<float> *varr\_e*,  pybind11::array\_t<float> *envr*,  size\_t *dist\_model*,  pybind11::array\_t<float> *dist\_par* )  **Description** |
| Determines the specific configuraton of correlated traits. The default Trait class constructor should be called before this method can be invoced.  **Parameters** |
| *pop* – instance of the Population class.  *trmean* – python array (list or numpy array) of floating point numbers representing expected mean values of correlated traits; the number of elements in the array is a number of correlated traits.  *qtl\_prop\_chrom* - python array (list or numpy array) of floating point numbers representing the proportion of SNPs selected (at random) as QTLs; the number of elements in the array is a number of correlated traits; each value is in the range [0,1].  *corr\_g* - python array (list or numpy array) of floating point numbers representing a square matrix of genomic correlations among traits; the dimension of the matrix is n-by-n, where n is a number of correlated traits; each value is in the range [0,1].  *varr\_g* - python array (list or numpy array) of floating point numbers representing expected values genomic variances for each of correlated traits; the number of elements in the array is a number of correlated traits.  *corr\_e* - python array (list or numpy array) of floating point numbers representing a square matrix of environmental (residual) correlations among traits; the dimension of the matrix is n-by-n, where n is a number of correlated traits; each value is in the range [0,1].  *varr\_e* - python array (list or numpy array) of floating point numbers representing expected values environmental (residual) variances for each of correlated traits; the number of elements in the array is a number of correlated traits.  *envr* - python array (list or numpy array) of floating point numbers representing environtal conditions for each of correlated traits; the number of elements in the array is a number of correlated traits; each value is in the range [0,1].  *dist\_mode* – positive integer value *{1,2,3}* indicating the mode of sampling dominance effects;  *dist\_mode =* 1 – sampling from a Uniform distribution,  *dist\_mode =* 2 – sampling from a Normal distribution,  *dist\_mode =* 3 – sampling from a Gamma distribution.  *dist\_par* – python array (list or numpy array) representing specific distribution parameters used for sampling dominance effects;  if *dist\_mode = 1*, *dist\_par = [a, b]*, where *a* and *b* are floating point numbers indicating the lower and upper bounds of the Uniform distributon;  if *dist\_mode = 2*, *dist\_par = [a, b]*, where *a* and *b* are floating point numbers indicating the mean and standard deviation respectively of the Normal distribution;  if *dist\_mode = 3*, *dist\_par = [a]*, where *a* is the floating point number indicating the rate parameters of the Gamma distribution (expected value of dominance effects in loci).  **Return value** |
| None.  **Example (use in Python)** |
| import evogen  pop = evogen.Population() # create the class instance  pop.set\_population(5, “gen\_struct\_pop.dat”, 0.4, 2) # simulate population  tr\_mean = [ 40.0, 5.0, 0.5 ] # (1) trait means  qtl\_prop = [ 0.65, 0.65, 0.65, 0.65 ] # (2) proportion of snps selected as qtls  qtl\_prop = [ 1.0, 1.0, 1.0, 1.0 ]  cor\_g = [ [1.0, 0.5, 0.7], [0.5, 1.0, 0.2], [0.7, 0.2, 1.0] ] # (3) genomic correlations  cor\_e = [ [1.0, 0.3, 0.5], [0.3, 1.0, 0.4], [0.5, 0.4, 1.0] ] # (4) rsidual correlations  var\_g = [ 100.0, 10.0, 0.1 ] # (5) genomic variances  var\_e = [ 200.0, 20.0, 0.3 ] # (6) residual variances  env = [ 0.0, 0.0, 0.0 ] # (7) enviironment  # (i) For uniform distribution, model 1  k\_range\_U = [ -1.0, 1.0 ] # range of k parameter  # (ii) For normal distribution, model 2  k\_range\_N = [ 0.0, 0.5 ] # mean & std  # (iii) For gamma distribution, mdodel 3  k\_range\_G = [ 0.05 ] # expected value of k in loci; is '1/b' param. in the distribution  mode = 1  T = evogen.Trait() # default constructor, create class instance  T.set\_trait(pop, tr\_mean, qtl\_prop, cor\_g, var\_g, cor\_e, var\_e, env, mode, k\_range\_U) |

|  |
| --- |
| void  reset\_trait ( Population & *pop*,  pybind11::array\_t<float> *trmean*,  pybind11::array\_t<float> *qtl\_prop\_chrom*,  pybind11::array\_t<float> *corr\_g*,  pybind11::array\_t<float> *varr\_g*,  pybind11::array\_t<float> *corr\_e*,  pybind11::array\_t<float> *varr\_e*,  pybind11::array\_t<float> *envr*,  size\_t *dist\_model*,  pybind11::array\_t<float> *dist\_par* )  **Description** |
| Redetermines the specific configuraton of correlated traits. This method should be applied to already existant and defined instannce of the Trait class.  **Parameters** |
| *pop* – instance of the Population class.  *trmean* – python array (list or numpy array) of floating point numbers representing expected mean values of correlated traits; the number of elements in the array is a number of correlated traits.  *qtl\_prop\_chrom* - python array (list or numpy array) of floating point numbers representing the proportion of SNPs selected (at random) as QTLs; the number of elements in the array is a number of correlated traits; each value is in the range [0,1].  *corr\_g* - python array (list or numpy array) of floating point numbers representing a square matrix of genomic correlations among traits; the dimension of the matrix is n-by-n, where n is a number of correlated traits; each value is in the range [0,1].  *varr\_g* - python array (list or numpy array) of floating point numbers representing expected values genomic variances for each of correlated traits; the number of elements in the array is a number of correlated traits.  *corr\_e* - python array (list or numpy array) of floating point numbers representing a square matrix of environmental (residual) correlations among traits; the dimension of the matrix is n-by-n, where n is a number of correlated traits; each value is in the range [0,1].  *varr\_e* - python array (list or numpy array) of floating point numbers representing expected values environmental (residual) variances for each of correlated traits; the number of elements in the array is a number of correlated traits.  *envr* - python array (list or numpy array) of floating point numbers representing environtal conditions for each of correlated traits; the number of elements in the array is a number of correlated traits; each value is in the range [0,1].  *dist\_mode* – positive integer value *{1,2,3}* indicating the mode of sampling dominance effects;  *dist\_mode =* 1 – sampling from a Uniform distribution,  *dist\_mode =* 2 – sampling from a Normal distribution,  *dist\_mode =* 3 – sampling from a Gamma distribution.  *dist\_par* – python array (list or numpy array) representing specific distribution parameters used for sampling dominance effects;  if *dist\_mode = 1*, *dist\_par = [a, b]*, where *a* and *b* are floating point numbers indicating the lower and upper bounds of the Uniform distributon;  if *dist\_mode = 2*, *dist\_par = [a, b]*, where *a* and *b* are floating point numbers indicating the mean and standard deviation respectively of the Normal distribution;  if *dist\_mode = 3*, *dist\_par = [a]*, where *a* is the floating point number indicating the rate parameters of the Gamma distribution (expected value of dominance effects in loci).  **Return value** |
| None.  **Example (use in Python)** |
| import evogen  pop = evogen.Population() # create the class instance  pop.set\_population(5, “gen\_struct\_pop.dat”, 0.4, 2) # simulate population  tr\_mean = [ 40.0, 5.0, 0.5 ] # (1) trait means  qtl\_prop = [ 0.65, 0.65, 0.65, 0.65 ] # (2) proportion of snps selected as qtls  qtl\_prop = [ 1.0, 1.0, 1.0, 1.0 ]  cor\_g = [ [1.0, 0.5, 0.7], [0.5, 1.0, 0.2], [0.7, 0.2, 1.0] ] # (3) genomic correlations  cor\_e = [ [1.0, 0.3, 0.5], [0.3, 1.0, 0.4], [0.5, 0.4, 1.0] ] # (4) rsidual correlations  var\_g = [ 100.0, 10.0, 0.1 ] # (5) genomic variances  var\_e = [ 200.0, 20.0, 0.3 ] # (6) residual variances  env = [ 0.0, 0.0, 0.0 ] # (7) enviironment  # (i) For uniform distribution, model 1  k\_range\_U = [ -1.0, 1.0 ] # range of k parameter  # (ii) For normal distribution, model 2  k\_range\_N = [ 0.0, 0.5 ] # mean & std  # (iii) For gamma distribution, mdodel 3  k\_range\_G = [ 0.05 ] # expected value of k in loci; is '1/b' param. in the distribution  mode = 1  T = evogen.Trait() # default constructor  T.set\_trait(pop, tr\_mean, qtl\_prop, cor\_g, var\_g, cor\_e, var\_e, env, mode, k\_range\_U)  qtl\_prop = [ 0.5, 0.6, 0.1, 0.9 ] # make channges to the parameter, then reset trait  T.reset\_trait(pop, tr\_mean, qtl\_prop, cor\_g, var\_g, cor\_e, var\_e, env, mode, k\_range\_U) |

|  |
| --- |
| void  get\_observations ( Population & *pop*, pybind11::array\_t<float> *env* )  **Description** |
| Collects observations (calculates traits values) for every individual in a specific population.  **Parameters** |
| *pop* – instance of Population class for which observations should be collected.  *env* – python array (list or numpy array) of floating point numbers describing environmental conditions; the array size is the number of correlated traits; values are from the range of [0,1].  **Return value** |
| None.  **Example (use in Python)** |
| import evogen  pop = evogen.Population() # create the class instance  pop.set\_population(5, “gen\_struct\_pop.dat”, 0.4, 2) # simulate population  tr\_mean = [ 40.0, 5.0, 0.5 ] # (1) trait means  qtl\_prop = [ 0.65, 0.65, 0.65, 0.65 ] # (2) proportion of snps selected as qtls  qtl\_prop = [ 1.0, 1.0, 1.0, 1.0 ]  cor\_g = [ [1.0, 0.5, 0.7], [0.5, 1.0, 0.2], [0.7, 0.2, 1.0] ] # (3) genomic correlations  cor\_e = [ [1.0, 0.3, 0.5], [0.3, 1.0, 0.4], [0.5, 0.4, 1.0] ] # (4) rsidual correlations  var\_g = [ 100.0, 10.0, 0.1 ] # (5) genomic variances  var\_e = [ 200.0, 20.0, 0.3 ] # (6) residual variances  env = [ 0.0, 0.0, 0.0 ] # (7) enviironment  # (i) For uniform distribution, model 1  k\_range\_U = [ -1.0, 1.0 ] # range of k parameter  # (ii) For normal distribution, model 2  k\_range\_N = [ 0.0, 0.5 ] # mean & std  # (iii) For gamma distribution, mdodel 3  k\_range\_G = [ 0.05 ] # expected value of k in loci; is '1/b' param. in the distribution  mode = 1  T = evogen.Trait() # default constructor  T.set\_trait(pop, tr\_mean, qtl\_prop, cor\_g, var\_g, cor\_e, var\_e, env, mode, k\_range\_U)  env = [ 0.1, 0.5, 0.0 ]  T.get\_observations(pop, env) |

|  |
| --- |
| void  get\_observations ( Population & *pop*,  pybind11::array\_t<float> *env*,  pybind11::array\_t<float> *t* )  **Description** |
| Collects observations (calculates traits values) for every individual in a specific population.  **Parameters** |
| *pop* – instance of Population class for which observations should be collected.  *env* – python array (list or numpy array) of floating point numbers describing environmental conditions; the array size is the number of correlated traits; values are from the range of [0,1].  *t* – numpy array of type float32 and arbitrary shape for output of trait values.  **Return value** |
| None.  **Example (use in Python)** |
| import evogen  import numpy as np  pop = evogen.Population() # create the class instance  pop.set\_population(5, “gen\_struct\_pop.dat”, 0.4, 2) # simulate population  tr\_mean = [ 40.0, 5.0, 0.5 ] # (1) trait means  qtl\_prop = [ 0.65, 0.65, 0.65, 0.65 ] # (2) proportion of snps selected as qtls  qtl\_prop = [ 1.0, 1.0, 1.0, 1.0 ]  cor\_g = [ [1.0, 0.5, 0.7], [0.5, 1.0, 0.2], [0.7, 0.2, 1.0] ] # (3) genomic correlations  cor\_e = [ [1.0, 0.3, 0.5], [0.3, 1.0, 0.4], [0.5, 0.4, 1.0] ] # (4) rsidual correlations  var\_g = [ 100.0, 10.0, 0.1 ] # (5) genomic variances  var\_e = [ 200.0, 20.0, 0.3 ] # (6) residual variances  env = [ 0.0, 0.0, 0.0 ] # (7) enviironment  # (i) For uniform distribution, model 1  k\_range\_U = [ -1.0, 1.0 ] # range of k parameter  # (ii) For normal distribution, model 2  k\_range\_N = [ 0.0, 0.5 ] # mean & std  # (iii) For gamma distribution, mdodel 3  k\_range\_G = [ 0.05 ] # expected value of k in loci; is '1/b' param. in the distribution  mode = 1  T = evogen.Trait() # default constructor  T.set\_trait(pop, tr\_mean, qtl\_prop, cor\_g, var\_g, cor\_e, var\_e, env, mode, k\_range\_U)  obs = np.zeros( shape=(1), dtype=np.float32 ) # note, the importans of correct typpe!  env = [ 0.1, 0.5, 0.0 ]  T.get\_observations(pop, env, obs) |

|  |
| --- |
| void  get\_observations ( Population & *pop*,  pybind11::array\_t<float> *env*,  pybind11::array\_t<float> *t*,  pybind11::array\_t<int> *g* )  **Description** |
| Collects observations (calculates traits values) for every individual in a specific population.  **Parameters** |
| *pop* – instance of Population class for which observations should be collected.  *env* – python array (list or numpy array) of floating point numbers describing environmental conditions; the array size is the number of correlated traits; values are from the range of [0,1].  *t* – numpy array of type float32 and arbitrary shape for output of trait values.  *g* – numpy array of type int32 and arbitrary shape for output of individual genotypes.  **Return value** |
| None.  **Example (use in Python)** |
| import evogen  import numpy as np  pop = evogen.Population() # create the class instance  pop.set\_population(5, “gen\_struct\_pop.dat”, 0.4, 2) # simulate population  tr\_mean = [ 40.0, 5.0, 0.5 ] # (1) trait means  qtl\_prop = [ 0.65, 0.65, 0.65, 0.65 ] # (2) proportion of snps selected as qtls  qtl\_prop = [ 1.0, 1.0, 1.0, 1.0 ]  cor\_g = [ [1.0, 0.5, 0.7], [0.5, 1.0, 0.2], [0.7, 0.2, 1.0] ] # (3) genomic correlations  cor\_e = [ [1.0, 0.3, 0.5], [0.3, 1.0, 0.4], [0.5, 0.4, 1.0] ] # (4) rsidual correlations  var\_g = [ 100.0, 10.0, 0.1 ] # (5) genomic variances  var\_e = [ 200.0, 20.0, 0.3 ] # (6) residual variances  env = [ 0.0, 0.0, 0.0 ] # (7) enviironment  # (i) For uniform distribution, model 1  k\_range\_U = [ -1.0, 1.0 ] # range of k parameter  # (ii) For normal distribution, model 2  k\_range\_N = [ 0.0, 0.5 ] # mean & std  # (iii) For gamma distribution, mdodel 3  k\_range\_G = [ 0.05 ] # expected value of k in loci; is '1/b' param. in the distribution  mode = 1  T = evogen.Trait() # default constructor  T.set\_trait(pop, tr\_mean, qtl\_prop, cor\_g, var\_g, cor\_e, var\_e, env, mode, k\_range\_U)  obs = np.zeros( shape=(1), dtype=np.float32 ) # note, the importans of correct typpe!  env = [ 0.1, 0.5, 0.0 ]  obs = np.zeros( shape=(1), dtype=np.float32 ) # note, the importans of correct typpe!  gen = np.zeros( shape=(1), dtype=np.int32 ) # note, the importans of correct typpe!  T.get\_observations(pop, env, obs, gen) |

|  |
| --- |
| void  get\_observations ( Population & *pop*,  pybind11::array\_t<float> *env*,  const std::string & *t* )  **Description** |
| Collects observations (calculates traits values) for every individual in a specific population.  **Parameters** |
| *pop* – instance of Population class for which observations should be collected.  *env* – python array (list or numpy array) of floating point numbers describing environmental conditions; the array size is the number of correlated traits; values are from the range of [0,1].  *t* – file name for output of trait values.  **Return value** |
| None.  **Example (use in Python)** |
| import evogen  pop = evogen.Population() # create the class instance  pop.set\_population(5, “gen\_struct\_pop.dat”, 0.4, 2) # simulate population  tr\_mean = [ 40.0, 5.0, 0.5 ] # (1) trait means  qtl\_prop = [ 0.65, 0.65, 0.65, 0.65 ] # (2) proportion of snps selected as qtls  qtl\_prop = [ 1.0, 1.0, 1.0, 1.0 ]  cor\_g = [ [1.0, 0.5, 0.7], [0.5, 1.0, 0.2], [0.7, 0.2, 1.0] ] # (3) genomic correlations  cor\_e = [ [1.0, 0.3, 0.5], [0.3, 1.0, 0.4], [0.5, 0.4, 1.0] ] # (4) rsidual correlations  var\_g = [ 100.0, 10.0, 0.1 ] # (5) genomic variances  var\_e = [ 200.0, 20.0, 0.3 ] # (6) residual variances  env = [ 0.0, 0.0, 0.0 ] # (7) enviironment  # (i) For uniform distribution, model 1  k\_range\_U = [ -1.0, 1.0 ] # range of k parameter  # (ii) For normal distribution, model 2  k\_range\_N = [ 0.0, 0.5 ] # mean & std  # (iii) For gamma distribution, mdodel 3  k\_range\_G = [ 0.05 ] # expected value of k in loci; is '1/b' param. in the distribution  mode = 1  T = evogen.Trait() # default constructor  T.set\_trait(pop, tr\_mean, qtl\_prop, cor\_g, var\_g, cor\_e, var\_e, env, mode, k\_range\_U)  env = [ 0.1, 0.5, 0.0 ]  obs = “observations.txt”  T.get\_observations(pop, env, obs) |

|  |
| --- |
| void  get\_observations ( Population & *pop*,  pybind11::array\_t<float> *env*,  const std::string & *t*,  const std::string & *g* )  **Description** |
| Collects observations (calculates traits values) for every individual in a specific population.  **Parameters** |
| *pop* – instance of Population class for which observations should be collected.  *env* – python array (list or numpy array) of floating point numbers describing environmental conditions; the array size is the number of correlated traits; values are from the range of [0,1].  *t* – file name for output of trait values.  *g* – file name for output of individual genotypes.  **Return value** |
| None.  **Example (use in Python)** |
| import evogen  pop = evogen.Population() # create the class instance  pop.set\_population(5, “gen\_struct\_pop.dat”, 0.4, 2) # simulate population  tr\_mean = [ 40.0, 5.0, 0.5 ] # (1) trait means  qtl\_prop = [ 0.65, 0.65, 0.65, 0.65 ] # (2) proportion of snps selected as qtls  qtl\_prop = [ 1.0, 1.0, 1.0, 1.0 ]  cor\_g = [ [1.0, 0.5, 0.7], [0.5, 1.0, 0.2], [0.7, 0.2, 1.0] ] # (3) genomic correlations  cor\_e = [ [1.0, 0.3, 0.5], [0.3, 1.0, 0.4], [0.5, 0.4, 1.0] ] # (4) rsidual correlations  var\_g = [ 100.0, 10.0, 0.1 ] # (5) genomic variances  var\_e = [ 200.0, 20.0, 0.3 ] # (6) residual variances  env = [ 0.0, 0.0, 0.0 ] # (7) enviironment  # (i) For uniform distribution, model 1  k\_range\_U = [ -1.0, 1.0 ] # range of k parameter  # (ii) For normal distribution, model 2  k\_range\_N = [ 0.0, 0.5 ] # mean & std  # (iii) For gamma distribution, mdodel 3  k\_range\_G = [ 0.05 ] # expected value of k in loci; is '1/b' param. in the distribution  mode = 1  T = evogen.Trait() # default constructor  T.set\_trait(pop, tr\_mean, qtl\_prop, cor\_g, var\_g, cor\_e, var\_e, env, mode, k\_range\_U)  env = [ 0.1, 0.5, 0.0 ]  obs = “observations.txt”  gen = “genotypes.txt”  T.get\_observations(pop, env, obs, gen) |

|  |
| --- |
| void  clear ( )  **Description** |
| Clears a specific trait object’s set-up, and release the ocupied memory.  In order to reuse the cleared trait object the method reset\_trait(…) should be called.  **Parameters** |
| None.  **Return value** |
| None.  **Example (use in Python)** |
| import evogen  pop = evogen.Population() # create the class instance  pop.set\_population(5, “gen\_struct\_pop.dat”, 0.4, 2) # simulate population  tr\_mean = [ 40.0, 5.0, 0.5 ] # (1) trait means  qtl\_prop = [ 0.65, 0.65, 0.65, 0.65 ] # (2) proportion of snps selected as qtls  qtl\_prop = [ 1.0, 1.0, 1.0, 1.0 ]  cor\_g = [ [1.0, 0.5, 0.7], [0.5, 1.0, 0.2], [0.7, 0.2, 1.0] ] # (3) genomic correlations  cor\_e = [ [1.0, 0.3, 0.5], [0.3, 1.0, 0.4], [0.5, 0.4, 1.0] ] # (4) rsidual correlations  var\_g = [ 100.0, 10.0, 0.1 ] # (5) genomic variances  var\_e = [ 200.0, 20.0, 0.3 ] # (6) residual variances  env = [ 0.0, 0.0, 0.0 ] # (7) enviironment  # (i) For uniform distribution, model 1  k\_range\_U = [ -1.0, 1.0 ] # range of k parameter  # (ii) For normal distribution, model 2  k\_range\_N = [ 0.0, 0.5 ] # mean & std  # (iii) For gamma distribution, mdodel 3  k\_range\_G = [ 0.05 ] # expected value of k in loci; is '1/b' param. in the distribution  mode = 1  T = evogen.Trait() # default constructor  T.set\_trait(pop, tr\_mean, qtl\_prop, cor\_g, var\_g, cor\_e, var\_e, env, which\_model, k\_range\_U)  T.clear() |

|  |
| --- |
| bool  is\_cleared ( )  **Description** |
| Checks if the trait object is cleared.  **Parameters** |
| None.  **Return value** |
| Boolean value (TRUE or FALSE).  **Example (use in Python)** |
| import evogen  pop = evogen.Population() # create the class instance  pop.set\_population(5, “gen\_struct\_pop.dat”, 0.4, 2) # simulate population  tr\_mean = [ 40.0, 5.0, 0.5 ] # (1) trait means  qtl\_prop = [ 0.65, 0.65, 0.65, 0.65 ] # (2) proportion of snps selected as qtls  qtl\_prop = [ 1.0, 1.0, 1.0, 1.0 ]  cor\_g = [ [1.0, 0.5, 0.7], [0.5, 1.0, 0.2], [0.7, 0.2, 1.0] ] # (3) genomic correlations  cor\_e = [ [1.0, 0.3, 0.5], [0.3, 1.0, 0.4], [0.5, 0.4, 1.0] ] # (4) rsidual correlations  var\_g = [ 100.0, 10.0, 0.1 ] # (5) genomic variances  var\_e = [ 200.0, 20.0, 0.3 ] # (6) residual variances  env = [ 0.0, 0.0, 0.0 ] # (7) enviironment  # (i) For uniform distribution, model 1  k\_range\_U = [ -1.0, 1.0 ] # range of k parameter  # (ii) For normal distribution, model 2  k\_range\_N = [ 0.0, 0.5 ] # mean & std  # (iii) For gamma distribution, mdodel 3  k\_range\_G = [ 0.05 ] # expected value of k in loci; is '1/b' param. in the distribution  mode = 1  T = evogen.Trait() # default constructor  T.set\_trait(pop, tr\_mean, qtl\_prop, cor\_g, var\_g, cor\_e, var\_e, env, which\_model, k\_range\_U)  T.clear()  status = T.is\_cleared() # will return TRUE |

|  |
| --- |
| **Description** |
| **Parameters** |
| **Return value** |
| **Example (use in Python)** |
|  |