

## Auto Scheduler

Generated by Doxygen 1.8.11



# Contents

<b>1</b>	<b>Namespace Index</b>	<b>1</b>
1.1	Namespace List . . . . .	1
<b>2</b>	<b>File Index</b>	<b>3</b>
2.1	File List . . . . .	3
<b>3</b>	<b>Namespace Documentation</b>	<b>5</b>
3.1	csp Namespace Reference . . . . .	5
3.1.1	Function Documentation . . . . .	6
3.1.1.1	calc_score() . . . . .	6
3.1.1.2	consecutive_exams() . . . . .	6
3.1.1.3	prof_preference() . . . . .	6
3.1.2	Variable Documentation . . . . .	6
3.1.2.1	alpha . . . . .	6
3.1.2.2	available_slots . . . . .	6
3.1.2.3	c . . . . .	6
3.1.2.4	common_students . . . . .	6
3.1.2.5	count_t . . . . .	6
3.1.2.6	course_numstudents . . . . .	6
3.1.2.7	course_slot . . . . .	6
3.1.2.8	courses . . . . .	6
3.1.2.9	courses_copy . . . . .	6
3.1.2.10	days . . . . .	6
3.1.2.11	iterate . . . . .	6

3.1.2.12	iterator	7
3.1.2.13	minlist	7
3.1.2.14	minslots	7
3.1.2.15	n	7
3.1.2.16	num_aval_slots	7
3.1.2.17	o_q	7
3.1.2.18	o_x	7
3.1.2.19	prob	7
3.1.2.20	prof_input	7
3.1.2.21	rand	7
3.1.2.22	s	7
3.1.2.23	slot_courses	7
3.1.2.24	slot_list	7
3.1.2.25	slot_numstudents	7
3.1.2.26	slots_per_day	7
3.1.2.27	t_limit	7
3.1.2.28	temperature	7
3.1.2.29	total_capacity	8
3.1.2.30	total_slots	8
3.1.2.31	val	8
3.1.2.32	var	8
3.1.2.33	weight	8
3.2	generate_course_list Namespace Reference	8
3.2.1	Variable Documentation	8
3.2.1.1	cols	8
3.2.1.2	course_list	8
3.2.1.3	f	8
3.2.1.4	line1	8
3.3	generate_room_list Namespace Reference	8
3.3.1	Variable Documentation	9

3.3.1.1	<a href="#">cols</a>	9
3.3.1.2	<a href="#">f</a>	9
3.3.1.3	<a href="#">line_without_newline</a>	9
3.3.1.4	<a href="#">room_list</a>	9
3.4	<a href="#">generate_student_list Namespace Reference</a>	9
3.4.1	<a href="#">Variable Documentation</a>	9
3.4.1.1	<a href="#">cols</a>	9
3.4.1.2	<a href="#">f</a>	9
3.4.1.3	<a href="#">line_without_newline</a>	9
3.4.1.4	<a href="#">student_list</a>	9
3.5	<a href="#">room_allocation Namespace Reference</a>	9
3.5.1	<a href="#">Variable Documentation</a>	10
3.5.1.1	<a href="#">available_rooms</a>	10
3.5.1.2	<a href="#">aval_rooms</a>	10
3.5.1.3	<a href="#">course_num</a>	10
3.5.1.4	<a href="#">course_rooms</a>	10
3.5.1.5	<a href="#">r</a>	10
3.5.1.6	<a href="#">room_capacity</a>	10
3.5.1.7	<a href="#">room_status</a>	10
3.5.1.8	<a href="#">sorted_room_list</a>	10
<b>4</b>	<b><a href="#">File Documentation</a></b>	<b>11</b>
4.1	<a href="#">csp.py File Reference</a>	11
4.2	<a href="#">generate_course_list.py File Reference</a>	12
4.3	<a href="#">generate_room_list.py File Reference</a>	12
4.4	<a href="#">generate_student_list.py File Reference</a>	12
4.5	<a href="#">room_allocation.py File Reference</a>	13
	<b><a href="#">Index</a></b>	<b>15</b>



# Chapter 1

## Namespace Index

### 1.1 Namespace List

Here is a list of all namespaces with brief descriptions:

<a href="#">csp</a>	5
<a href="#">generate_course_list</a>	8
<a href="#">generate_room_list</a>	8
<a href="#">generate_student_list</a>	9
<a href="#">room_allocation</a>	9





## Chapter 2

# File Index

### 2.1 File List

Here is a list of all files with brief descriptions:

<a href="#">csp.py</a>	11
<a href="#">generate_course_list.py</a>	12
<a href="#">generate_room_list.py</a>	12
<a href="#">generate_student_list.py</a>	12
<a href="#">room_allocation.py</a>	13



## Chapter 3

# Namespace Documentation

### 3.1 csp Namespace Reference

#### Functions

- def `consecutive_exams` ()
- def `prof_preference` ()
- def `calc_score` ()

#### Variables

- int `total_capacity` = 0
- int `slots_per_day` = 4
- int `days` = 6
- `total_slots` = `days*slots_per_day`
- dictionary `course_numstudents` = {}
- list `courses` = []
- dictionary `common_students` = {}
- dictionary `available_slots` = {}
- dictionary `course_slot` = {}
- dictionary `slot_courses` = {}
- dictionary `slot_numstudents` = {}
- dictionary `prof_input` = {}
- int `n` = 1
- list `weight` = [100\*n , 100\*n , 25\*n]
- int `temperature` = 30000
- *Here we attempt to get a random feasible solution (ie a solution which satisfies all hard constraints).*
- float `alpha` = 0.8
- `t_limit` = `math.pow(10,-11)`
- int `iterator` = 10
- `c` = `course_list[course]`
- *Create a 2d array of the common students in the course1 and course2.*
- `courses_copy` = `list(courses)`
- int `val` = 0
- `minslots` = `total_slots`
- list `minlist` = []
- `num_aval_slots` = `len(available_slots[course])`

- `s = secrets.choice(available_slots[c])`
- `int count_t = 0`
- `slot_list = list(range(1,total_slots+1))`
- `o_x = calc_score()`
- `o_q = calc_score()`
- `var = o_x-o_q`
- `prob = math.exp(var/temperature)`
- `rand = numpy.random.choice([0,1], p=[prob, 1-prob])`
- `int iterate = 0`

*Hill Climbing.*

### 3.1.1 Function Documentation

3.1.1.1 `def csp.calc_score ( )`

3.1.1.2 `def csp.consecutive_exams ( )`

3.1.1.3 `def csp.prof_preference ( )`

### 3.1.2 Variable Documentation

3.1.2.1 `float csp.alpha = 0.8`

3.1.2.2 `dictionary csp.available_slots = {}`

3.1.2.3 `csp.c = course_list[course]`

Create a 2d array of the common students in the course1 and course2.

STEP ANHEALING.

Here we generate neighbourhood solutions. And try to get to the most optimized solution by iterating over solutions.

3.1.2.4 `dictionary csp.common_students = {}`

3.1.2.5 `int csp.count_t = 0`

3.1.2.6 `dictionary csp.course_numstudents = {}`

3.1.2.7 `dictionary csp.course_slot = {}`

3.1.2.8 `list csp.courses = [ ]`

3.1.2.9 `csp.courses_copy = list(courses)`

3.1.2.10 `int csp.days = 6`

3.1.2.11 `int csp.iterate = 0`

Hill Climbing.

In approximately 5-10% of cases, the simulated annealing algorithm had not sufficiently explored the neighbourhood of its best solution.

```
3.1.2.12 int csp.iterator = 10

3.1.2.13 list csp.minlist = []

3.1.2.14 csp.minslots = total_slots

3.1.2.15 int csp.n = 1

3.1.2.16 csp.num_aval_slots = len(available_slots[course])

3.1.2.17 csp.o_q = calc_score()

3.1.2.18 csp.o_x = calc_score()

3.1.2.19 csp.prob = math.exp(var/temperature)

3.1.2.20 dictionary csp.prof_input = {}

3.1.2.21 csp.rand = numpy.random.choice([0,1], p=[prob, 1-prob])

3.1.2.22 csp.s = secrets.choice(available_slots[c])

3.1.2.23 dictionary csp.slot_courses = {}

3.1.2.24 csp.slot_list = list(range(1,total_slots+1))

3.1.2.25 dictionary csp.slot_numstudents = {}

3.1.2.26 int csp.slots_per_day = 4

3.1.2.27 csp.t_limit = math.pow(10,-11)

3.1.2.28 int csp.temperature = 30000
```

Here we attempt to get a random feasible solution(ie a solution which satisfies all hard constraints).

- `That` is we first randomly allot a course to to a slot.
- `Then` we allot slots to courses in similar fashion.
- `If` we fail in this attempt we restart the loop and allot the course some other random slot. Note- Here we are considering that there exists finitely many solutions for the given number of slots.

3.1.2.29 `int csp.total_capacity = 0`

3.1.2.30 `csp.total_slots = days*slots_per_day`

3.1.2.31 `int csp.val = 0`

3.1.2.32 `csp.var = o_x-o_q`

3.1.2.33 `list csp.weight = [100*n , 100*n , 25*n]`

## 3.2 `generate_course_list` Namespace Reference

### Variables

- `f` = `open("course_students.txt", "r")`
- dictionary `course_list` = {}
- `line1` = `line.rstrip(",")`
- `cols` = `line1.split(",")`

### 3.2.1 Variable Documentation

3.2.1.1 `generate_course_list.cols = line1.split(",")`

3.2.1.2 dictionary `generate_course_list.course_list` = {}

3.2.1.3 `generate_course_list.f = open("course_students.txt", "r")`

3.2.1.4 `generate_course_list.line1 = line.rstrip(",")`

## 3.3 `generate_room_list` Namespace Reference

### Variables

- `f` = `open("room_capacity.txt", "r")`
- dictionary `room_list` = {}
- `line_without_newline` = `line.rstrip()`
- `cols` = `line_without_newline.split(",")`

### 3.3.1 Variable Documentation

3.3.1.1 generate\_room\_list.cols = line\_without\_newline.split(",")

3.3.1.2 generate\_room\_list.f = open("room\_capacity.txt", "r")

3.3.1.3 generate\_room\_list.line\_without\_newline = line.rstrip()

3.3.1.4 dictionary generate\_room\_list.room\_list = {}

## 3.4 generate\_student\_list Namespace Reference

### Variables

- `f` = open("student\_courses.txt", "r")
- dictionary `student_list` = {}
- `line_without_newline` = line.rstrip()
- `cols` = line\_without\_newline.split(",")

### 3.4.1 Variable Documentation

3.4.1.1 generate\_student\_list.cols = line\_without\_newline.split(",")

3.4.1.2 generate\_student\_list.f = open("student\_courses.txt", "r")

3.4.1.3 generate\_student\_list.line\_without\_newline = line.rstrip()

3.4.1.4 dictionary generate\_student\_list.student\_list = {}

## 3.5 room\_allocation Namespace Reference

### Variables

- dictionary `room_status` = {}
- dictionary `course_rooms` = {}
- dictionary `aval_rooms` = {}
- list `available_rooms` = []
- dictionary `room_capacity` = {}
- `course_num` = dict(course\_numstudents)
- `sorted_room_list` = sorted(room\_list, key=lambda x: int(room\_list[x]))
- `r` = `aval_rooms`[course][len(`aval_rooms`[course])-1]

### 3.5.1 Variable Documentation

3.5.1.1 list `room_allocation.available_rooms = []`

3.5.1.2 dictionary `room_allocation.aval_rooms = {}`

3.5.1.3 `room_allocation.course_num = dict(course_numstudents)`

3.5.1.4 dictionary `room_allocation.course_rooms = {}`

3.5.1.5 `room_allocation.r = aval_rooms[course][len(aval_rooms[course])-1]`

3.5.1.6 dictionary `room_allocation.room_capacity = {}`

3.5.1.7 dictionary `room_allocation.room_status = {}`

3.5.1.8 `room_allocation.sorted_room_list = sorted(room_list, key=lambda x: int(room_list[x]))`



## Chapter 4

# File Documentation

### 4.1 csp.py File Reference

#### Namespaces

- `csp`

#### Functions

- `def csp.consecutive_exams ()`
- `def csp.prof_preference ()`
- `def csp.calc_score ()`

#### Variables

- `int csp.total_capacity = 0`
- `int csp.slots_per_day = 4`
- `int csp.days = 6`
- `csp.total_slots = days*slots_per_day`
- `dictionary csp.course_numstudents = {}`
- `list csp.courses = []`
- `dictionary csp.common_students = {}`
- `dictionary csp.available_slots = {}`
- `dictionary csp.course_slot = {}`
- `dictionary csp.slot_courses = {}`
- `dictionary csp.slot_numstudents = {}`
- `dictionary csp.prof_input = {}`
- `int csp.n = 1`
- `list csp.weight = [100*n , 100*n , 25*n]`
- `int csp.temperature = 30000`

*Here we attempt to get a random feasible solution (ie a solution which satisfies all hard constraints).*

- `float csp.alpha = 0.8`
- `csp.t_limit = math.pow(10,-11)`
- `int csp.iterator = 10`
- `csp.c = course_list[course]`

*Create a 2d array of the common students in the course1 and course2.*

- `csp.courses_copy` = list(courses)
- int `csp.val` = 0
- `csp.minslots` = total\_slots
- list `csp.minlist` = []
- `csp.num_aval_slots` = len(available\_slots[course])
- `csp.s` = secrets.choice(available\_slots[c])
- int `csp.count_t` = 0
- `csp.slot_list` = list(range(1,total\_slots+1))
- `csp.o_x` = calc\_score()
- `csp.o_q` = calc\_score()
- `csp.var` = o\_x-o\_q
- `csp.prob` = math.exp(var/temperature)
- `csp.rand` = numpy.random.choice([0,1], p=[prob, 1-prob])
- int `csp.iterate` = 0

*Hill Climbing.*

## 4.2 generate\_course\_list.py File Reference

### Namespaces

- `generate_course_list`

### Variables

- `generate_course_list.f` = open("course\_students.txt", "r")
- dictionary `generate_course_list.course_list` = {}
- `generate_course_list.line1` = line.rstrip(",")
- `generate_course_list.cols` = line1.split(",")

## 4.3 generate\_room\_list.py File Reference

### Namespaces

- `generate_room_list`

### Variables

- `generate_room_list.f` = open("room\_capacity.txt", "r")
- dictionary `generate_room_list.room_list` = {}
- `generate_room_list.line_without_newline` = line.rstrip()
- `generate_room_list.cols` = line\_without\_newline.split(",")

## 4.4 generate\_student\_list.py File Reference

### Namespaces

- `generate_student_list`

### Variables

- `generate_student_list.f` = `open("student_courses.txt", "r")`
- dictionary `generate_student_list.student_list` = {}
- `generate_student_list.line_without_newline` = `line.rstrip()`
- `generate_student_list.cols` = `line_without_newline.split(",")`

## 4.5 room\_allocation.py File Reference

### Namespaces

- `room_allocation`

### Variables

- dictionary `room_allocation.room_status` = {}
- dictionary `room_allocation.course_rooms` = {}
- dictionary `room_allocation.aval_rooms` = {}
- list `room_allocation.available_rooms` = []
- dictionary `room_allocation.room_capacity` = {}
- `room_allocation.course_num` = `dict(course_numstudents)`
- `room_allocation.sorted_room_list` = `sorted(room_list, key=lambda x: int(room_list[x]))`
- `room_allocation.r` = `aval_rooms[course][len(aval_rooms[course])-1]`



# Index

alpha  
    csp, 6  
available\_rooms  
    room\_allocation, 10  
available\_slots  
    csp, 6  
aval\_rooms  
    room\_allocation, 10  
  
c  
    csp, 6  
calc\_score  
    csp, 6  
cols  
    generate\_course\_list, 8  
    generate\_room\_list, 9  
    generate\_student\_list, 9  
common\_students  
    csp, 6  
consecutive\_exams  
    csp, 6  
count\_t  
    csp, 6  
course\_list  
    generate\_course\_list, 8  
course\_num  
    room\_allocation, 10  
course\_numstudents  
    csp, 6  
course\_rooms  
    room\_allocation, 10  
course\_slot  
    csp, 6  
courses  
    csp, 6  
courses\_copy  
    csp, 6  
csp, 5  
    alpha, 6  
    available\_slots, 6  
    c, 6  
    calc\_score, 6  
    common\_students, 6  
    consecutive\_exams, 6  
    count\_t, 6  
    course\_numstudents, 6  
    course\_slot, 6  
    courses, 6  
    courses\_copy, 6  
    days, 6  
    iterate, 6  
    iterator, 6  
    minlist, 7  
    minslots, 7  
    n, 7  
    num\_aval\_slots, 7  
    o\_q, 7  
    o\_x, 7  
    prob, 7  
    prof\_input, 7  
    prof\_preference, 6  
    rand, 7  
    s, 7  
    slot\_courses, 7  
    slot\_list, 7  
    slot\_numstudents, 7  
    slots\_per\_day, 7  
    t\_limit, 7  
    temperature, 7  
    total\_capacity, 7  
    total\_slots, 8  
    val, 8  
    var, 8  
    weight, 8  
csp.py, 11  
  
days  
    csp, 6  
  
f  
    generate\_course\_list, 8  
    generate\_room\_list, 9  
    generate\_student\_list, 9  
  
generate\_course\_list, 8  
    cols, 8  
    course\_list, 8  
    f, 8  
    line1, 8  
generate\_course\_list.py, 12  
generate\_room\_list, 8  
    cols, 9  
    f, 9  
    line\_without\_newline, 9  
    room\_list, 9  
generate\_room\_list.py, 12  
generate\_student\_list, 9  
    cols, 9  
    f, 9  
    line\_without\_newline, 9

- student\_list, 9
- generate\_student\_list.py, 12
- iterate
  - csp, 6
- iterator
  - csp, 6
- line1
  - generate\_course\_list, 8
- line\_without\_newline
  - generate\_room\_list, 9
  - generate\_student\_list, 9
- minlist
  - csp, 7
- minslots
  - csp, 7
- n
  - csp, 7
- num\_aval\_slots
  - csp, 7
- o\_q
  - csp, 7
- o\_x
  - csp, 7
- prob
  - csp, 7
- prof\_input
  - csp, 7
- prof\_preference
  - csp, 6
- r
  - room\_allocation, 10
- rand
  - csp, 7
- room\_allocation, 9
  - available\_rooms, 10
  - aval\_rooms, 10
  - course\_num, 10
  - course\_rooms, 10
  - r, 10
  - room\_capacity, 10
  - room\_status, 10
  - sorted\_room\_list, 10
- room\_allocation.py, 13
- room\_capacity
  - room\_allocation, 10
- room\_list
  - generate\_room\_list, 9
- room\_status
  - room\_allocation, 10
- s
  - csp, 7
- slot\_courses
  - csp, 7
- slot\_list
  - csp, 7
- slot\_numstudents
  - csp, 7
- slots\_per\_day
  - csp, 7
- sorted\_room\_list
  - room\_allocation, 10
- student\_list
  - generate\_student\_list, 9
- t\_limit
  - csp, 7
- temperature
  - csp, 7
- total\_capacity
  - csp, 7
- total\_slots
  - csp, 8
- val
  - csp, 8
- var
  - csp, 8
- weight
  - csp, 8