# Contents

# Chapter 1

# Functions

## 1.1 bigrandom – random numbers

**Historical Note** The module was written for replacement of the `Python` standard module random, because in the era of `Python` 2.2 (prehistorical period of `NZMATH` ) the random module raises `OverflowError` for long integer arguments for the randrange function, which is the only function having a use case in `NZMATH` .

After the creation of `Python` 2.3, it was theoretically possible to use `random.randrange`, since it started to accept long integer as its argument. Use of it was, however, not considered, since there had been the `bigrandom` module. It was lucky for us. In fall of 2006, we found a bug in `random.randrange` and reported it (see issue tracker); the `random.randrange` accepts long integers but returns unreliable result for truly big integers. The bug was fixed for `Python` 2.5.1. You can, therefore, use `random.randrange` instead of `bigrandom.randrange` for `Python` 2.5.1 or higher.

### 1.1.1 random – random number generator

**random() → *float***

Return a random floating point number in the interval $[0, 1)$.

This function is an alias to `random.random` in the `Python` standard library.

### 1.1.2 randrange – random integer generator

**randrange(start: *integer*, stop: *integer*=None, step: *integer*=1 )**
**→ *integer***

Return a random integer in the range.

The argument names do not correspond to their roles, but users are familiar with the range built-in function of `Python` and understand the semantics. Calling with one argument $n$, then the result is an integer in the range $[0, n)$ chosen randomly. With two arguments $n$ and $m$, in $[n, m)$, and with third $l$, in $[n, m) \cap (n + l\mathbb{Z})$.

This function is almost the same as `random.randrange` in the `Python` standard library. See the historical note 1.1.

### Examples

```
>>> randrange(4, 10000, 3)
9919L
>>> randrange(4 * 10**60)
3192591690816225396918232749182359614561283479987677511462015 1L
```

### 1.1.3 map_choice – choice from image of mapping

**map_choice(mapping: *function*, upperbound: *integer* )**
**→ *integer***

Return a choice from a set given as the image of the mapping from natural numbers (more precisely `range(upperbound)`). In other words, it is equivalent to: `random.choice([mapping(i) for i in range(upperbound)])`, if upperbound is small enough for the list size limit.

The mapping can be a partial function, i.e. it may return `None` for some input. However, if the resulting set is empty, it will end up with an infinite loop.

# Bibliography