M2 I a e, U e Pa D de
A e-C a e Ha
2016/2017

Exercice 1 – Listes et dictionnaires

`average(lst)` `lst`
`len` `sum`

`average([3, 7, 1, 2, 3])` `3.2`

```
d
```

average median
[2, 4, 5, 6, 173]

occurrences(lst)
lst
occurrences([1, 2, 4, 3, 4, 1, 1, 3])
{1: 3, 2: 1, 3: 2, 4: 2} 1 3 2 1

```
def occurrences(lst):
    """ Computes the occurrences of the values in a list of integers.
    Args:
        lst: a list of integers.
    Returns:
        A dictionary where each key is a value in lst and each value is the number
        of times the key appears in lst.
    """
    ...
```

unique(lst)
lst
unique([2, 3, 1, 1, 3, 1, 4, 1, 3]) [2, 3, 1, 4]

```
def unique(lst):
    """ Returns the unique values of a list of integers.
    Args:
        lst: a list of integers.
    Returns:
        A list containing the unique values in lst.
    """
    ...
```

Exercice 2 – Compréhension de listes

squares(lst)
lst
squares([1, 2, 3, 4]) [1, 4, 9, 16]

```
def squares(lst):
    """ Squares the values of a list of integers.
    Args:
        lst: a list of integers.
    Returns:
        A list containing the squared elements of lst, in the same order.
    """
    ...
```

stddev(lst) lst
stddev([5, 4, 7, 4, 4, 2, 5, 9]) 2.0 stddev([20, 20, 20])
0.0 average

```
def stddev(lst):
    """ Returns the standard deviation of a list of integers.
    Args:
        lst: a list of integers.
    Returns:
        A float representing the empirical standard deviation of elements in lst.
    """
    ...
```

quicksort(lst)
lst ***Quicksort***
for

```
def quicksort(lst):
    """ Returns a sorted version of lst.
    Args:
        lst: a list of integers.
    Returns:
        A list of the integers from lst, in increasing order.
    """
    ...
```

Exercice 3 – Simulation de variables aléatoires

random random

```
from random import random
```

uniform()
0 1

```
def uniform():
    """ Returns a discrete uniform variable between 0 and 1, i.e., it returns 0
    with probability 0.5 and 1 with probability 0.5.
    """
    …
```

0 1 2 3 4 5 ⅙

```
def uniform():
    """ Returns a discrete uniform variable between 0 and 5, i.e., the probability
    of obtaining 0, 1, 2, 3, 4 or 5 is always 1/6.
    """
    …
```

0 1 n-1
n

```
def uniform(n):
    """ Returns a discrete uniform variable between 0 and n-1, i.e., it returns
    every number between 0 and n-1 with the same probability.
    Args:
        n: a positive integer
    """
    …
```

n p 0.0
1.0 n p

```
def exam_success(n, p):
    """ Returns the number of exams passed by a student.
    Args:
        n: number of independent exams the students sits for.
        p: probability of passing one exam.
    Returns:
        An integer representing the total number of exams passed by the
        student.
    """
    ...
```

Exercice 4 – Paradoxe de Monty Hall

monty_hall(change)
change

randint

random

```
def monty_hall(change):
    """ Simulates a Monty Hall game: a candidate is asked to choose between three
    doors with only one of them containing a reward. Once the candidate has chosen
    a door, the anchorman opens one of the other doors that does not contain the
    reward. The candidate may now change doors for the remaining one. This function
    returns 1 if the candidate found the reward and 0 otherwise.
    Args:
        change: a boolean representing whether the candidate changes doors.
    Returns:
        1 if the candidate found the reward at the end of the game, 0 otherwise.
    """
    ...
```

monty_hall_simulation(n)　　n

```
def monty_hall_simulation(n):
    """ Simulates n Monty Hall games with the candidate changing doors and n Monty
    Hall games with the candidate not changing doors. Returns the frequency with
    which the candidate found the reward in each case.
    Args:
        n: the number of games to simulate.
    Returns:
        A tuple (p1, p2) where p1 (resp. p2) is the frequency with which the
        candidate found the reward when always (resp. never) changing doors.
    """
    ...
```