
**ΠΕΡΙΕΧΟΜΕΝΑ:**

1. Τυχαίοι Ακέραιοι
2. Τυχαίοι Πραγματικοί
3. Τυχαιότητα σε Ακολουθίες
4. Εσωτερική Αναπαράσταση
5. OPC: Rolling Dice

Γιώργος Μ.

Σμαραγδένιος Χορηγός Μαθήματος

Ιωάννης Τ.

Σμαραγδένιος Χορηγός Μαθήματος

# ΜΑΘΗΜΑ 2: Το module random

## 1. Τυχαίοι Ακέραιοι

modules psounis

- Το **module random** περιέχει συναρτήσεις για την παραγωγή τυχαίων αριθμών από διάφορες κατανομές.

**Φύτρα Τυχαίων Αριθμών:**

| συνάρτηση | επεξήγηση |
|---|---|
| seed(a=None) | Αν a=None, τότε χρησιμοποιείται η τρέχουσα ώρα του συστήματος. Αλλιώς διοχετεύουμε κάποιον συγκεκριμένο ακέραιο |

**Παρατηρήσεις:**

- Οι γεννήτριες τυχαίων αριθμών δουλεύουν κάνοντας πράξεις στον προηγούμενο αριθμό για να παραχθεί ο επόμενος.
- Γι' αυτό χρησιμοποιούμε τη φύτρα, για να παραχθεί ο πρώτος αριθμός της τυχαίας σειράς.
- Χρησιμοποιώντας την ίδια φύτρα, παράγεται ακριβώς η ίδια σειρά (το οποίο είναι χρήσιμο για λόγους debugging)
- Η φύτρα παίζει τον ίδιο ρόλο, είτε δουλεύουμε σε ακεραίους είτε σε πραγματικούς αριθμούς.

**Παράδειγμα 1: seed.py**

```
from random import seed, randrange
seed(0)
print([randrange(3) for i in range(5)])
```

**Τυχαίοι Ακέραιοι:**

| συνάρτηση | επεξήγηση |
|---|---|
| randrange(stop) | Τυχαίος Ακέραιος στο [0, stop) |
| randrange(start, stop[, step]) | Τυχαίος Ακέραιος στο [start, stop) (με προαιρετικό βήμα step) |
| randint(a,b) | Τυχαίος ακέραιος στο [a,b] |

**Παρατήρηση:**

- Οι τιμές που παράγονται είναι ισοπίθανες (ομοιόμορφη κατανομή)

**Άσκηση 1:**

Παράγετε τρεις λίστες με 10 τυχαίους ακεραίους η κάθε μία. Οι λίστες να έχουν τυχαίους αριθμούς:

- Λίστα 1:Από το 100 έως το 200
- Λίστα 2: Άρτιοι φυσικοί μικρότεροι του 10
- Λίστα 3: Πολλαπλάσια του 3 μεταξύ του 100 και του 200.

## Τυχαίοι Πραγματικοί (ακολουθούν κατανομές)

| συνάρτηση | επεξήγηση |
|---|---|
| random()<br>uniform(a,b) | Ομοιόμορφη κατανομή<br>Τυχαίος πραγματικός στο [0,1)<br>Τυχαίος πραγματικός στο [a,b] |
| triangular(a, b, mode) | Τριγωνική Κατανομή<br>a,b: άκρα, mode: default το μέσο |
| betavariate(a,b) | Κατανομή βήτα<br>a,b: άκρα, mode: default το μέσο |
| expovariate(l) | Εκθετική Κατανομή<br>l: παράμετρος λ |
| gammavariate(a,b) | Κατανομή γάμμα<br>a: shape, b: 1/scale |
| gauss(m, s) | Κατανομή Gauss<br>m: mu, s: sigma |

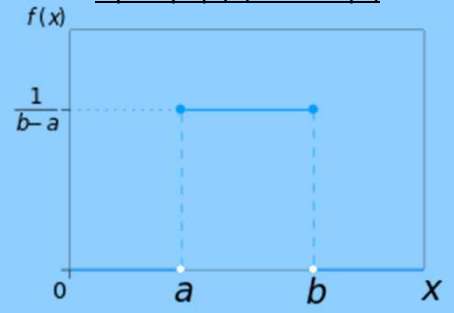
## Άσκηση 2:

Παράγετε 1000000 τυχαίους πραγματικούς που ακολουθούν την τριγωνική κατανομή με άκρα τα 0, 1:

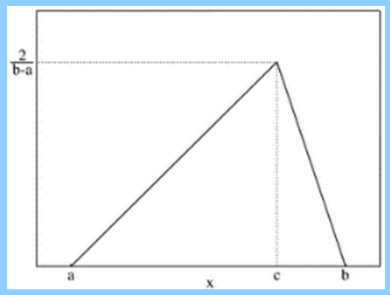
- Μετρήστε πόσοι βρίσκονται σε κάθε διάστημα εύρους 0.1 (δηλ. [0,0.1), [0.1,0.2),…[0.9,1])
- Τυπώστε το ποσοστό των αριθμών που βρίσκονται σε κάθε ένα από τα παραπάνω διαστήματα.

## Αντίστοιχες Συναρτήσεις πυκνότητας πιθανότητας:

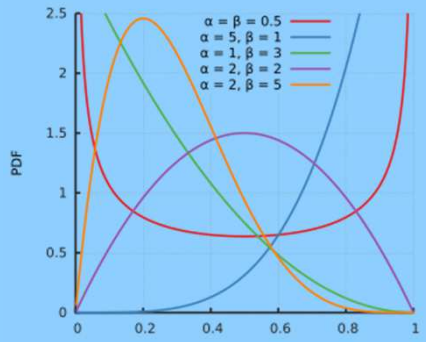
Ομοιόμορφη κατανομή



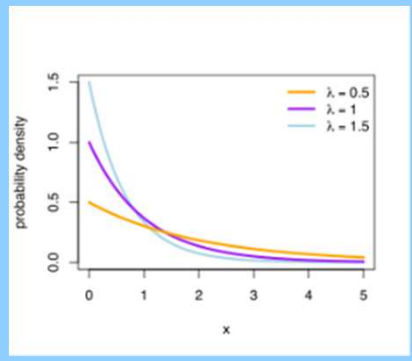

Τριγωνική κατανομή



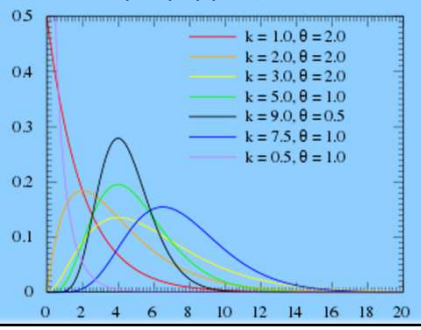

Κατανομή βήτα



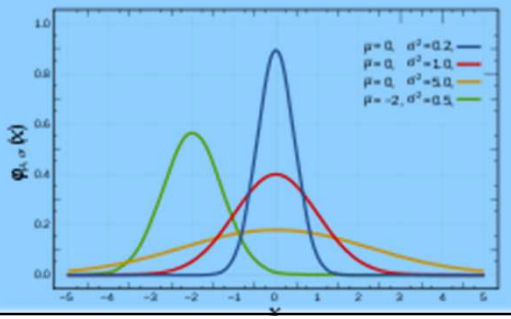

Εκθετική Κατανομή




Κατανομή γάμμα (α=k, b=1/θ)




Κατανομή Gauss



Wikipedia Images

## Τυχαίοι Πραγματικοί (ακολουθούν κατανομές)

| συνάρτηση | επεξήγηση |
|---|---|
| lognormvariate(m, s) | Κανονική Λογαριθμική κατανομή<br>m: mu, s: sigma |
| normvariate(m, s) | Κανονική Κατανομή<br>m: mu, s: sigma |
| vonmisesvariate(m,k) | Κατανομή von Mises<br>m: mu, k: kappa |
| paretovariate(a) | Κατανομή Pareto<br>a: shape |
| weibullvariate(a,b) | Κατανομή Weibull<br>a: scale, b: shape |

**Παρατήρηση:**

- Η normvariate είναι ελαφρά πιο γρήγορη από τη gauss

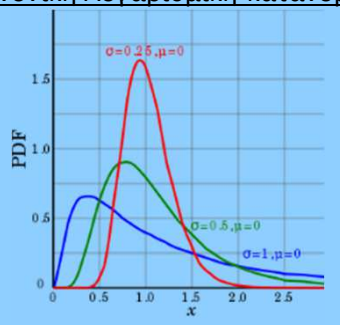
**Άσκηση 2:**

Παράγετε 1000000 τυχαίους πραγματικούς που ακολουθούν την κατανομή Pareto με a=3:

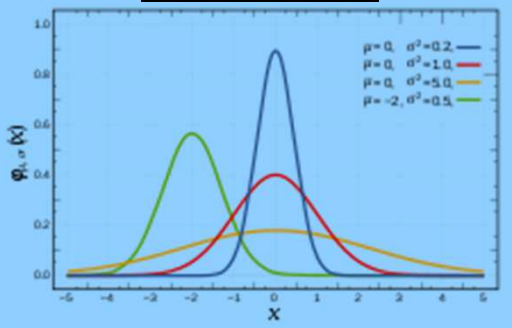
- Δημιουργήστε bins στο εύρη [0,1), [1,2), [2,3), [3,…)
- Τυπώστε το ποσοστό των αριθμών που βρίσκονται σε κάθε ένα από τα παραπάνω διαστήματα.

## Αντίστοιχες Συναρτήσεις πυκνότητας πιθανότητας:

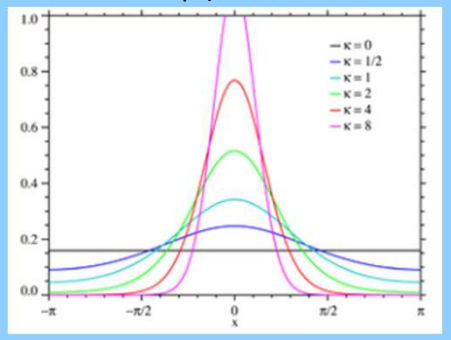
Κανονική Λογαριθμική κατανομή



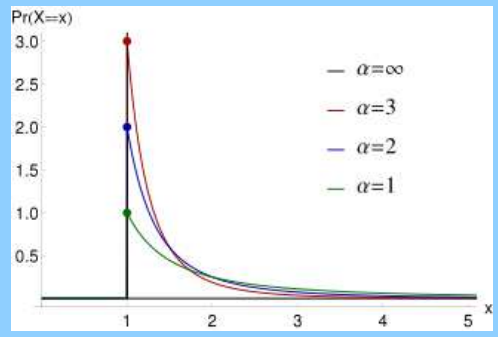

Κανονική κατανομή



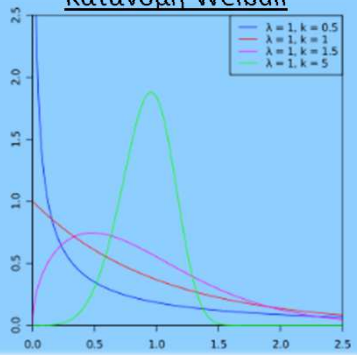

Κατανομή von Mises




Κατανομή Pareto




Κατανομή Weibull



Wikipedia Images

## Τυχαία επιλογή στοιχείων ακολουθίας (choice και choices):

| συνάρτηση | επεξήγηση |
|---|---|
| choice(seq) | Επιστρέφει ένα τυχαίο στοιχείο της ακολουθίας |
| *choices(seq, weights=None,*,cum_weights=None, k=1)* | |
| choices(seq, k=n) | Επιστρέφει μία λίστα, με επανάληψη στοιχείων, με n τυχαία στοιχεία της seq (k: keyword arg) |
| choices(seq, weights, k=n) | Επιστρέφει μία λίστα με n τυχαία στοιχεία της seq (βάρος επιλογής με βάση τη λίστα weights) |
| choices(seq, cum_weights=list, k=n) | Επιστρέφει μία λίστα με n τυχαία στοιχεία (με αθροιστικό βάρος επιλογής με βάση τη λίστα cumulative weights) |

## Αναδιάταξη στοιχείων ακολουθίας:

| συνάρτηση | επεξήγηση |
|---|---|
| shuffle(seq [,random]) | Αναδιατάσσει τα στοιχεία της ακολουθίας (για mutable ακολουθίες) [Προαιρετικό: Διοχετεύουμε συνάρτηση που θα αντικαταστήσει τη random] |
| sample(seq,k) | Επιστρέφει μία λίστα, χωρίς επανάληψη στοιχείων, με k τυχαία στοιχεία |

## Παράδειγμα 2: choices.py

```
from random import choice, choices
my_list = list(range(5))
print(f"Random choice: {choice(my_list)}")
print(f"3 Random choices: {choices(my_list, k=3)}")
print(f"5 Weighted choices: {choices(my_list, [1,100,10,1,1], k=10)}")
print(f"5 Cum.W. choices: {choices(my_list,
                    cum_weights=[1,101,111,112,113], k=10)}")
```

## Άσκηση 3:

Κατασκευάστε:

- Μία λίστα 5 πόλεων (συμβολοσειρές)
- Ένα 2Δ πίνακα 5x5 με τυχαίους αριθμούς στο [0,100)

Έστω ότι ο 2ος πίνακας αντιπροσωπεύει τις χιλιομετρικές αποστάσεις μεταξύ των 5 πόλεων. Κατασκευάστε:

- Δύο τυχαίες διατάξεις των 5 πόλεων (με κατάλληλη κωδικοποίηση)
- Υπολογίστε το άθροισμα των αποστάσεων με βάση τις δύο διατάξεις.
- Επιλέξτε την καλύτερη (μικρότερο άθροισμα αποστάσεων)

## Εσωτερική Αναπαράσταση

- Εκτός της seed(x) που με όρισμα συγκεκριμένο x θα παράγει ακριβώς την ίδια ακολουθία,
- μας δίνεται η δυνατότητα να "σώσουμε" την κατάσταση της γεννήτριας τυχαίων αριθμών, ώστε να την "ενεργοποιήσουμε" και να συνεχίσουμε από το ίδιο σημείο που κάναμε το "σώσιμο".

| συνάρτηση | επεξήγηση |
|---|---|
| getstate() | Επιστρέφει σε ένα αντικείμενο, την κατάσταση της γεννήτριας |
| setstate(state) | Θέτουμε στην γεννήτρια την κατάσταση state (που έχουμε σώσει σε κάποιο προηγούμενο βήμα) |

## Παράδειγμα 3: state.py

```
from random import getstate, setstate, randint
state = None
for i in range(10):
    print(i, randint(1, 10))
    if i == 5:
        state = getstate()
print("============")
setstate(state)
for i in range(6, 10):
    print(i, randint(1, 10))
```

## Η συνάρτηση getrandbits:

| συνάρτηση | επεξήγηση |
|---|---|
| getrandbits(k) | Επιστρέφει έναν τυχαίο ακέραιο με k random bits |

## Παράδειγμα 4: getrandbits.py

```
from random import getrandbits

for i in range(100):
    print(getrandbits(2))
```

## Άσκηση 4:

*Κατασκευάστε έναν τυχαίο ακέραιο με 8 bits και έπειτα τυπώστε την δυαδική του αναπαράσταση.*

## Σημείωση:

- Το επίσημο documentation της Python, αναφέρει ότι δεν θα πρέπει να χρησιμοποιείται η random σε εφαρμογές ασφάλειας (αντίθετα προτείνει το module secret)

**opc.dice.py**

Ο παίκτης ρίχνει ένα ζάρι, αλλά πριν μαντεύει το αποτέλεσμα. Αν κερδίσει, κερδίζει ισόποσα ευρώ, αν χάσει χάνει 1 ευρώ. Το ίδιο κάνει και ο υπολογιστής. Αρχικά και οι δύο έχουν τόσα ευρώ, όσα και οι γύροι που θα παίξουν. Κερδίζει είτε αυτός που έχει τα περισσότερα ευρώ στο τέλος των γύρων, είτε αν τα ευρώ του αντιπάλου του, γίνουν 0.

```
import random, sys

class Dice:
    def __init__(self, n=1):
        self.n = n   # no of dice
    def throw(self):
        return [random.randint(1,6) for i in range(self.n)]

class Player(object):
    def __init__(self, name, capital, guess_function, ndice):
        self.name = name
        self.capital = capital
        self.guess_function = guess_function
        self.dice = Dice(ndice)
    def play_one_round(self):
        self.guess = self.guess_function(self.dice.n)
        self.throw = sum(self.dice.throw())
        if self.guess == self.throw:
            self.capital += self.guess
        else:
            self.capital -= 1
        self.message()
        self.broke()
    def message(self):
        print('%s guessed %d, got %d' % (self.name, self.guess, self.throw))
    def broke(self):
        if self.capital == 0:
            print('%s lost!' % self.name)
            sys.exit(0)  # end the program
def computer_guess(ndice):
    # All guesses have the same probability
    return random.randint(ndice, 6*ndice)
def player_guess(ndice):
    return int(input('Guess the sum of the no of eyes in the next throw: '))
def play(nrounds, ndice=2):
    player = Player('YOU', nrounds, player_guess, ndice)
    computer = Player('Computer', nrounds, computer_guess, ndice)
    for i in range(nrounds):
        player.play_one_round()
        computer.play_one_round()
        print('Status: user has %d euro, machine has %d euro\n' %
              (player.capital, computer.capital))
    if computer.capital > player.capital:
        winner = 'Machine'
    else:
        winner = 'You'
    print(winner, 'won!')
play(8, 1)
```

*Source:http://hplgit.github.io/primer.html/doc/pub/random/random-readable.html*