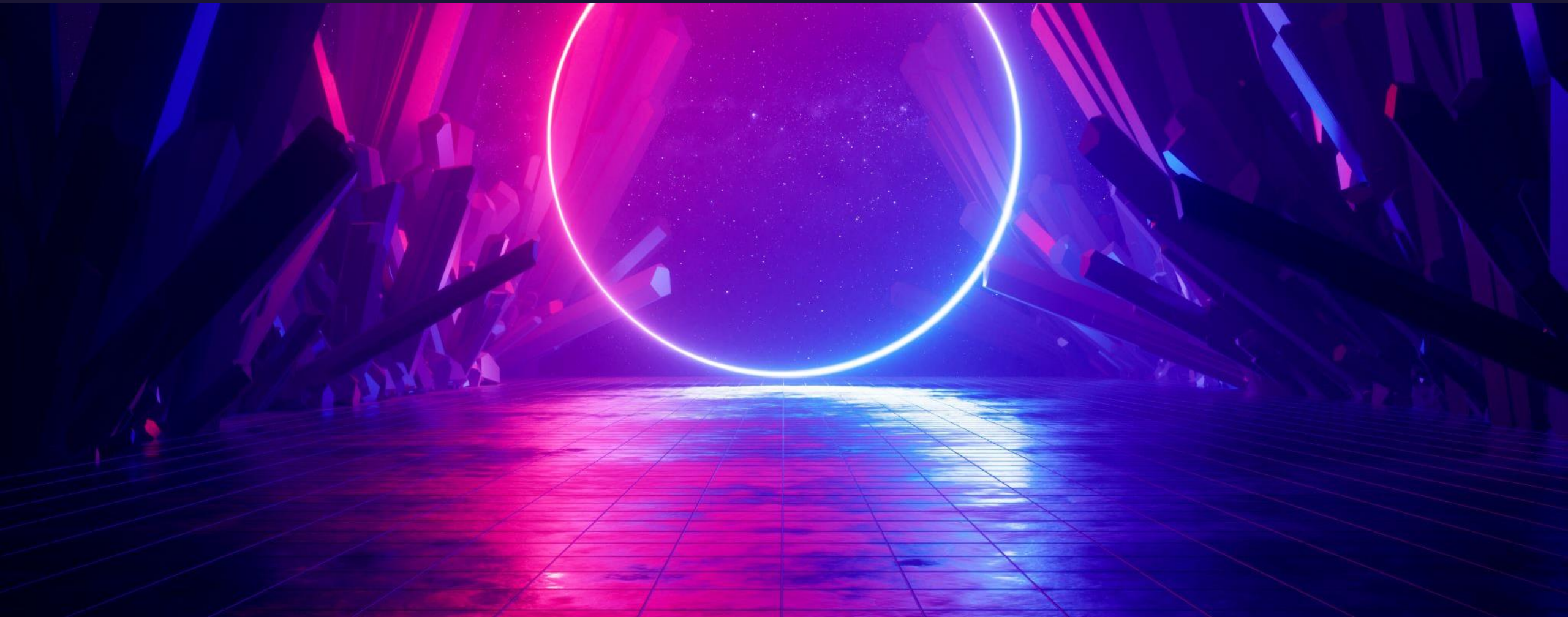


Programowanie w Pythonie

Łukasz Mioduszewski, UKSW 2022

Biblioteki math, random, statistics, scipy, inne



Biblioteka math – ciekawsze funkcje

- `ceil(x)` – zwraca `x` zaokrąglone w górę (jeśli `x` nie jest floatem, `ceil` odwołuje się do wbudowanej metody `__ceil__(self)`, o ile `x` ją ma)
- `fmod(x,y)` – zwraca `x mod y` zgodną ze standardem C (lepsza dokładność numeryczna dla floatów niż wbudowane w Pythona `x % y`)
- `remainder(x,y)` – reszta z dzielenia `x` przez `y`, może być ujemna
- `fsum(iterable)` – dokładnie sumuje elementy obiektu `iterable` (lepiej niż `sum`)
- `gcd(*integers)` – największy wspólny dzielnik dowolnej liczby argumentów
- `lcm(*integers)` – najmniejsza wspólna wielokrotność dowolnej liczby argumentów

Biblioteka math – ciekawsze funkcje

- `prod(iterable, *, start=1)` – iloczyn wszystkich elementów w iterable
- `trunc(x)` – utnij część ułamkową (floor dla dodatnich, ceil dla ujemnych)
- `exp(x) = ex`, `exp2(x) = 2x`, `expm1(x) = ex-1`, `pow(x,y) = xy`, `log`, `log2`, `log10`, `sqrt...`
- Trygonometria (`sin`, `cos`, `atan...`) w radianach, konwersja: `degrees(x)` i `radians(x)`
- `isfinite(x)`, `isinf(x)`, `isnan(x)` – sprawdzają czy x jest (nie)skończony lub NaN
- `dist(p,q) - sqrt(sum((px - qx) ** 2.0 for px, qx in zip(p, q)))`
- `hypot(*coordinates) - sqrt(sum(x**2 for x in coordinates))`
- `comb(n,k)` – współczynnik dwumianowy $n! / (k! * (n - k)!)$

Biblioteka math – stałe i inne

- e , π , $\tau = 2\pi$, inf , nan
- Biblioteka `cmath` zawiera odpowiedniki funkcji z `math` dla liczb zespolonych



Biblioteka random

- Dobry, szybki ale deterministyczny algorytm Mersenne Twister – nie nadaje się do kryptografii ani zabezpieczeń (do tego służy moduł secrets)
- Funkcja `random()` losuje float z zakresu `[0.0,1.0)`
- Można tworzyć podklasy klasy `Random`
- Domyślnie generator liczb losowych korzysta z czasu systemowego albo `os.urandom()` jeśli jest dostępne. Aby ustawić powtarzalne liczby losowe, wybierz `random.seed(a)`, gdzie `a` to `int`, `float`, `str`, `bytes`, albo `bytearray`.

Biblioteka random – funkcje

- `getstate()` pobiera aktualny stan generatora, `setstate(state)` go ustawia
- `random.randbytes(n)` zwraca `n` losowych bajtów
- `randrange(start, stop[, step])` zwraca losowy element z `range(start, stop, step)`
- `randint(a,b) = randrange(a,b+1)`
- `getrandbits(k)` zwraca nieujemną liczbę całkowitą z `k` losowych bitów



Biblioteka random – funkcje dla sekwencji

- `choice(seq)` zwraca losowy element z `seq`
- `choices(population, weights=None, *, cum_weights=None, k=1)` zwraca `k` losowych elementów z `population` z użyciem wag `weights` lub kumulatywnych
- `shuffle(x)` – tasuje `x` (musi być mutowalny)
- `sample(population, k, *, counts=None)` wybiera `k` losowych unikalnych elementów z `population`, `counts` pozwala powiedzieć ile razy elementy występują



Biblioteka random – funkcje "typu float"

- `uniform(a,b)` jest równoważne $a + (b-a) * \text{random}()$.
- `gauss(mu=0.0, sigma=1.0)` rozkład Gaussa (normalny)
- i wiele innych...



Biblioteka statistics - funkcje


- `mean(data)` – zwykła średnia
- `fmean(data, weights=None)` – szybka średnia konwertująca na float
- `geometric_mean()`, `harmonic_mean()` – średnie geometryczna i harmoniczna
- `median()` – mediana (jeśli jest parzysta liczba elementów, to średnia z dwóch)
- `median_low()` i `median_high()` – mediana biorąca mniejszy(wiekszy) z dwóch
- `mode()` – najczęściej występujący element (jeśli jest remis, to pierwszy)
- `multimode()` – lista najczęstszych elementów (dłuższa niż 1 gdy jest remis)

Biblioteka statistics - funkcje

- `variance(data, xbar=None)` – wariancja wokół punktu `mu` (domyślnie średnia)
- `stdev(data, xbar=None)` – odchylenie standardowe (pierwiastek z wariancji)

$$u(x) = \sqrt{s_{\bar{x}}^2} = \sqrt{\frac{1}{n(n-1)} \sum_{i=1}^n (x_i - \bar{x})^2}.$$

- `pvariance(data, mu=None)` – wariancja wokół punktu `mu` (domyślnie średnia)
- `pstdev(data, mu=None)` – odchylenie standardowe (pierwiastek z wariancji)


$$u(x) = \sqrt{s_{\bar{x}}^2} = \sqrt{\frac{1}{n(n-1)} \sum_{i=1}^n (x_i - \bar{x})^2}.$$

Biblioteka statistics - funkcje

- `quantiles(data, *, n=4, method='exclusive')` – wyznacza n zakresów, które są równoprawdopodobne. Zwraca listę $n-1$ wartości, wyznaczających granice zakresów. Domyślnie wartości brzegowe (początek pierwszego zakresu i koniec ostatniego) są nieznane, dla `method='inclusive'` przyjmuje się że wartości brzegowe wyznacza najmniejsza i największa liczba pośród danych
- `covariance(x,y)` – kowariancja zbiorów x i y (nowość w Pythonie 3.10)
- `correlation(x,y)` – wsp. korelacji Pearsona między zbiorami x i y (-1 jeśli są dokładnie antyskorelowane, +1 jeśli są dokładnie skorelowane) (nowość w 3.10)

Biblioteka statistics - funkcje

- `linear_regression(x, y, /, *, proportional=False)` – dopasowuje dane x oraz y do funkcji $y = a \cdot x + b$ i zwraca tuplę (a,b), chyba że `proportional=True`, wtedy dopasowuje do $y = a \cdot x$ i zwraca a

```
>>> year = [1971, 1975, 1979, 1982, 1983]
>>> films_total = [1, 2, 3, 4, 5]
>>> slope, intercept = linear_regression(year, films_total)
>>> round(slope * 2019 + intercept)
```

16

Biblioteka statistics – obiekty NormalDist

- `NormalDist(mu=0.0, sigma=1.0)` tworzy nowy obiekt `NormalDist` będący rozkładem o średniej `mu` i odchyleniu standardowym `sigma`
- Atrybuty: `mean`, `median`, `mode`, `stdev`, `variance`
- Metoda klasy: `from_samples(data)` tworzy `NormalDist` z danych
dygresja: metody klasy nie muszą mieć argumentu `self`... ale mogą



Metody obiektów NormalDist

- `samples(n, *, seed=None)` – tworzy n losowych liczb z rozkładu (seed służy do reprodukowalności, żeby generator liczb losowych działał tak samo)
- `cdf(x)` – prawdop. że losowa liczba X z rozkładu jest mniejsza od x , czyli $P(X \leq x)$
- `inv_cdf(p)` – funkcja odwrotna, szuka takiego x że $P(X \leq x) = p$
- `pdf(x)` – gęstość rozkładu prawdopodobieństwa, $\lim P(x \leq X < x+dx) / dx$
- `overlap(other)` – mierzy przekrycie rozkładu z innym (między 0 a 1)
- `quantiles(n=4)` – zwraca $n-1$ liczb będących granicami kwantyli rozkładu

NormalDist - przykłady

- Oceny mają średnią 1060 i odchylenie standardowe 195, oblicz jaki procent studentów miał oceny pomiędzy 1100 a 1200 (oceny są całkowite)

```
sat = NormalDist(1060, 195)
```

```
fraction = sat.cdf(1200 + 0.5) - sat.cdf(1100 - 0.5)
```

```
round(fraction * 100.0, 1) # wynik to 18.4
```



NormalDist - przykłady

- Na konferencji jest 750 uczestników i 2 sale mieszczące po 500 osób. Na poprzedniej konferencji 65% uczestników słuchało prezentacji o Pythonie, a 35% o Ruby. Zakładając że preferencje i tematy się nie zmieniły, oblicz prawdopodobieństwo że słuchacze prezentacji o Pythonie zmieszczą się w sali.
- ```
from statistics import *
n = 750 # Rozmiar próbki
p = 0.65 # Jaka część woli Pythona
k = 500 # Ile osób mieści się w sali
from math import sqrt # Przybliżenie przy użyciu cdf
NormalDist(mu=n*p, sigma=sqrt(n*p*(1.0-p))).cdf(k + 0.5) # 0.8402
```



# NormalDist - przykłady

- Na konferencji jest 750 uczestników i 2 sale mieszczące po 500 osób. Na poprzedniej konferencji 65% uczestników słuchało prezentacji o Pythonie, a 35% o Ruby. Zakładając że preferencje i tematy się nie zmieniły, oblicz prawdopodobieństwo że słuchacze prezentacji o Pythonie zmieszczą się w sali.
- `from math import comb, fsum # Rozwiązanie używające rozkładu dwumianowego`  
`fsum(comb(n, r) * p**r * (1.0-p)**(n-r) for r in range(k+1)) # 0.8402`

$$f(k, n, p) = \Pr(k; n, p) = \Pr(X = k) = \binom{n}{k} p^k (1 - p)^{n-k}$$

`from random import seed, choices # Przybliżenie przy użyciu symulacji`  
`def trial():`

`return choices(('Python', 'Ruby'), (p, q), k=n).count('Python')`  
`mean(trial() <= k for i in range(10_000)) # 0.8398`

## posteriori.py

```
1 height_male = NormalDist.from_samples([6, 5.92, 5.58, 5.92])
2 height_female = NormalDist.from_samples([5, 5.5, 5.42, 5.75])
3 weight_male = NormalDist.from_samples([180, 190, 170, 165])
4 weight_female = NormalDist.from_samples([100, 150, 130, 150])
5 foot_size_male = NormalDist.from_samples([12, 11, 12, 10])
6 foot_size_female = NormalDist.from_samples([6, 8, 7, 9])
7
8 # nowa osoba
9 ht = 6.0 # height
10 wt = 130 # weight
11 fs = 8 # foot size
12
13 # obliczamy prawdopodobieństwo płci
14 prior_male = 0.5
15 prior_female = 0.5
16 posterior_male = (prior_male * height_male.pdf(ht) *
17 weight_male.pdf(wt) * foot_size_male.pdf(fs))
18
19 posterior_female = (prior_female * height_female.pdf(ht) *
20 weight_female.pdf(wt) * foot_size_female.pdf(fs))
21
22 # Maximum a posteriori (MAP):
23 'male' if posterior_male > posterior_female else 'female'
24 'female'
```

# Dostęp do Internetu i wysyłanie maili

```
>>> from urllib.request import urlopen
>>> with urlopen('http://worldtimeapi.org/api/timezone/etc/UTC.txt') as response:
... for line in response:
... line = line.decode() # Convert bytes to a str
... if line.startswith('datetime'):
... print(line.rstrip()) # Remove trailing newline
...
datetime: 2022-01-01T01:36:47.689215+00:00

>>> import smtplib
>>> server = smtplib.SMTP('localhost')
>>> server.sendmail('soothsayer@example.org', 'jcaesar@example.org',
... """To: jcaesar@example.org
... From: soothsayer@example.org
...
... Beware the Ides of March.
... """)
>>> server.quit()
```

# Data i czas

```
>>> # dates are easily constructed and formatted
>>> from datetime import date
>>> now = date.today()
>>> now
datetime.date(2003, 12, 2)
>>> now.strftime("%m-%d-%y. %d %b %Y is a %A on the %d day of %B.")
'12-02-03. 02 Dec 2003 is a Tuesday on the 02 day of December.'

>>> # dates support calendar arithmetic
>>> birthday = date(1964, 7, 31)
>>> age = now - birthday
>>> age.days
14368
```

# Kompresja danych

- Dostępne biblioteki: zlib, gzip, bz2, lzma, zipfile, tarfile

```
>>> import zlib
>>> s = b'witch which has which witches wrist watch'
>>> len(s)
41
>>> t = zlib.compress(s)
>>> len(t)
37
>>> zlib.decompress(t)
b'witch which has which witches wrist watch'
>>> zlib.crc32(s)
226805979
```

# Wydajność

- Biblioteka `timeit` do pojedynczych komend, biblioteki `profile` i `pstats` do badania czasu wykonywania dużych fragmentów kodu
- Dla `timeit` pierwszy argument to mierzone polecenie, a drugi to "setup"

```
>>> from timeit import Timer
>>> Timer('t=a; a=b; b=t', 'a=1; b=2').timeit()
0.57535828626024577
>>> Timer('a,b = b,a', 'a=1; b=2').timeit()
0.54962537085770791
```

# Proste testy – moduł doctest

- Wystarczy przekleić test do dokumentacji, a moduł doctest sprawdzi czy działa

```
def average(values):
 """Computes the arithmetic mean of a list of numbers.

 >>> print(average([20, 30, 70]))
 40.0
 """

 return sum(values) / len(values)

import doctest
doctest.testmod() # automatically validate the embedded tests
```

# Testy jednostkowe – moduł unittest

- Testy w oddzielnym pliku

```
import unittest

class TestStatisticalFunctions(unittest.TestCase):

 def test_average(self):
 self.assertEqual(average([20, 30, 70]), 40.0)
 self.assertEqual(round(average([1, 5, 7]), 1), 4.3)
 with self.assertRaises(ZeroDivisionError):
 average([])
 with self.assertRaises(TypeError):
 average(20, 30, 70)

unittest.main() # Calling from the command line invokes all tests
```



# Ładne pisanie - biblioteka pprint

```
>>> import pprint
>>> t = [[['black', 'cyan'], 'white', ['green', 'red']], [['magenta',
... 'yellow'], 'blue']]
...
>>> pprint.pprint(t, width=30)
[[['black', 'cyan'],
 'white',
 ['green', 'red']],
 [['magenta', 'yellow'],
 'blue']]
```

# Ładne pisanie - biblioteka textwrap

```
>>> import textwrap
>>> doc = """The wrap() method is just like fill() except that it returns
... a list of strings instead of one big string with newlines to separate
... the wrapped lines."""
...
>>> print(textwrap.fill(doc, width=40))
The wrap() method is just like fill()
except that it returns a list of strings
instead of one big string with newlines
to separate the wrapped lines.
```

# Ćwiczenia – bonusowe punkty za zadania

- Moodle nie pozwala na ocenę powyżej 100 pkt, w związku z tym nie ma możliwości zapisu dodatkowych punktów
- Proszę pod koniec semestru pamiętać o nich, i jeśli komuś będzie brakowało punktów do lepszej oceny to mi przypomnieć

