AKADEMIA GÓRNICZO-HUTNICZA
IM. STANISŁAWA STASZICA W KRAKOWIE

Programowanie w języku Python

Maciej Wielgosz

Wydział Informatyki, Elektroniki i Telekomunikacji

2019, semestr zimowy

AGH

Część I

Python

Python

Python - cechy języka

- interpretowany
- interaktywny
- obiektowo-zorientowany

Python - cechy języka

- prostota
- przejrzystość
- łatwość wykonywania złożonych operacji
- wygodna diagnostyka błędów
- ogromny ekosystem modułów dla najróżniejszych zastosowań
- łatwość łączenia z kodem w innych językach
- skalowalność
- przenośność

Standardy

- 1.0 (1994)
- 2.0 (2000)
- 3.0 (2008)
- 3.4 (2014)
- 3.5 (2015)
- 3.6 (2016)
- 3.7 (2018)

Kompatybilność

Od wersji 3.0 zerwano z wymogiem wstecznej kompatybilności.
Narzędzie *2to3* pozwala wykonać większość translacji.

Uruchamianie

- python3
- python3 test.py

Tryb interaktywny

```
#python3
Python 3.7.4 (default, Jul  9 2019, 18:13:23)
[Clang 10.0.1 (clang-1001.0.46.4)] on darwin
Type "help", "copyright", "credits" or "license"
for more information.
>>> help()
help> input
Help on built-in function input in module builtins:
...
help> quit
>>> quit()
```

Python

Identyfikatory

- zbudowane z liter, cyfr i znaku podkreślenia
- zaczynają się od litery lub znaku podkreślenia
- małe i wielkie litery są odróżniane
- konwencja: nazwy klas zaczynają się od wielkiej litery, zmienne od małej
- istnieje zestaw zarezerwowanych słów kluczowych

Linie

```
1 total = item_one + \
2         item_two + \
3         item_three
4
5 months = ['March', 'April', 'May',
6           'June', 'July']
7
8 import sys; x = 'foo'; sys.stdout.write(x + '\n')
```

Indentacja

```
1 if len(filename) == 0:
2   print("Empty filename")
3   sys.exit()
4 try:
5   file = open(filename, "r")
6 except IOError:
7   print("Error opening file")
8   sys.exit()
```

Drobiazgi

- komentarze

```
1 # komentarz
2 print('Hello'); ala = 7; print(ala) #
      ↪ komentarz
3 '''
4 Napis pełniący rolę
5 wielolinijkowego
6 komentarza
7 '''
```

- interakcja z użytkownikiem

```
1 x = int(input())
```

Python

Typizacja

Zmienna jest tworzona w chwili przypisania wartości, typ zmiennej jest ustalany dynamicznie.

```
>>> x = 'Ala'
>>> type(x)
<class 'str'>
>>> x = 7
>>> type(x)
<class 'int'>
>>> x = str(x)
>>> type(x)
<class 'str'>
```

Podstawowe typy

- liczby: `2, -3, 1.56`
- napisy: `'hello', ''tekst''`
- listy: `[1, 2, 3], ['ala', 'ola', 'ela'], [1, 'ola']`
- krotki:
  `(1,2,3), ('ala', 'ola', 'ela'), (1, ['ola']), (2,)`
- słowniki: `{ 'ala': 'kot', 'ola': 1 }`
- typ boolowski: `True, False`
- nic: `None`

Liczby

```
>>> 2 + 3
5
>>> 9 / 5
1.8
>>> 9 // 5
1
>>> 9 % 5
4
>>> 6 ** 2
36
>>> a = 20
>>> a
20
```

Napisy

```
>>> 'do it'
'do it'
>>> 'doesn\'t'
"doesn't"
>>> "doesn't"
"doesn't"
>>> '"Yes," she said.'
'"Yes," she said.'
>>> "\"Yes,\" she said."
'"Yes," she said.'
```

Napisy

```
>>> s = 'Pierwszy wiersz.\nDrugi wiersz.'
>>> s  # bez funkcji print()
'Pierwszy wiersz.\nDrugi wiersz.'
>>> print(s)  # print()
Pierwszy wiersz.
Drugi wiersz.

>>> print('C:\Dokumenty\nina')
C:\Dokumenty
ina
>>> print('C:\Dokumenty\\nina')
C:\Dokumenty\nina
>>> print(r'C:\Dokumenty\nina')
C:\Dokumenty\nina
```

Napisy

```
>>> 2 * 'la' + 'mido'
'lalamido'
>>> 'Py' 'thon'
'Python'
>>> prefix = 'Py'
>>> prefix + 'thon'
'Python'
>>> w = 'Python'
>>> w[0]
'P'
>>> w[5]
'n'
>>> w[-2]
'o'
```

Napisy

```
>>> w[2:5]
'tho'
>>> w[:2]
'Py'
>>> w[4:]
'on'
>>> len(w)
6
```

Listy

```
>>> szesciany = [1, 8, 27, 64, 125]
>>> szesciany[1]
8
>>> szesciany[-3:]
[27, 64, 125]
>>> szesciany[:]
[1, 8, 27, 64, 125]
>>> szesciany + [216, 343]
>>> szesciany.append(512)
>>> szesciany
[1, 8, 27, 64, 125, 512]
>>> a = ['a', 'b', 'c']
>>> b = [1, 2]
>>> c = [a, b]
>>> c
[['a', 'b', 'c'], [1, 2]]
```

Listy - dodawanie elementów

```
>>> list = ['a']
>>> list = list + [2.0, 5]
>>> list
['a', 2.0, 5]
>>> list.append([True, 'c'])
>>> list
['a', 2.0, 3, [True, 'c']]
>>> list.extend(['b', 'd'])
>>> list
['a', 2.0, 3, [True, 'c'], 'b', 'd']
>>> list.insert(1, 'd')
>>> list
['a', 'd', 2.0, 3, [True, 'c'], 'b', 'd']
```

Python

Operatory

- arytmetyczne: `+, -, *, /, %, **, //`
- porównania: `==, !=, <>, >, <, >=, <=`
- przypisania: `=, +=` ...
- bitowe
- logiczne: `and, or, not`

if

```
1  if punkty >= 90:
2      ocena = '5'
3  elif punkty >= 75:
4      ocena = '4'
5  elif punkty >= 60:
6      ocena = '3'
7  else:
8      ocena = '2'
```

for

```
1 for litera in 'Python':
2     print('litera:', litera)
3
4 warzywa = ['marchew', 'kalafior',  'kapusta']
5 for warzywo in warzywa:
6     print('warzywo:', warzywo)
```

range()

```
for i in range(5):
    print(i)
# 0 1 2 3 4
```

```
for i in range(2, 11, 2):
    print(i)
# 2 4 6 8 10
```

```
print(list(range(2, 11, 2)))
# [2, 4, 6, 8, 10]
```

while

```
liczby = list()
i = 2
while i < 11:
    liczby.append(i)
    i = i + 2
print(liczby)       # [2, 4, 6, 8, 10]
```

```
lines = list()
print('Wprowadź tekst po linijce.')
print('Żeby zakończyć wprowadź pustą linię.')
line = input('Następna linijka: ')
while line != '':
    lines.append(line)
    line = input('Następna linijka: ')   # reset
print(lines)
```

break, else, continue

```
for n in range(2, 100):
    for x in range(2, n):
        if n % x == 0:
            break
    else: # normalny koniec petli
        print(n, 'jest liczbą pierwszą')
```

```
for num in range(1, 20):
    if not num % 2:
        print('Kolejna liczba parzysta:', num)
        continue
    print('Kolejna liczba:', num)
```

pass

```
while True:
    pass # aktywne oczekiwanie
```

```
class minimalClass:
    pass
```

```
def function_to_implement(*args)
    pass
```

Definicja

```
1  def fib2(n):
2      ''' zwraca liczby Fibonacciego mniejsze od n
           ↪ '''
3      wynik = []
4      a, b = 0, 1
5      while a < n:
6          wynik.append(a)
7          a, b = b, a+b
8      return wynik
9
10 x = fib2(10)
11 print(x)
12 print(fib2.__doc__)
```

Argumenty domyślne

```
1  def f(a, L=[]):
2      L.append(a)
3      return L
4  print(f(1)) # [1]
5  print(f(2)) # [1, 2]
6
7  def f(a, L=None):
8      if L is None:
9          L = []
10     L.append(a)
11     return L
12 print(f(1)) # [1]
13 print(f(2)) # [2]
```

Przekazywanie argumentów

```
1  def evilGetLength(ilist):
2    length = len(ilist)
3    del ilist[:] # usunięcie zawartości
4    return length
5
6  list1 = [1, 2]
7  length = evilGetLength(list1) # list1 jest pusta
8  list1 = [1, 2]
9  length = evilGetLength(list1[:]) # przekazanie
       ↪ kopii
10
11 def f(a, b):
12   return a - b
13
14 c = f(b=4, a=2)
```

Lista - operacje

```
list.append(x) # dołącza x
list.extend(L) # dołącza liste L
list.insert(i, x) # wstawia x przed pozycję i
list.remove(x) # usuwa pierwszy x
list.pop(i) # usuwa i-ty element
list.pop # usuwa ostatni element
list.clear() # usuwa wszystkie elementy
list.index(x) # indeks pierwszego x
list.count(x) # liczba wystapień x
list.sort() # sortuje w miejscu
list.reverse() # odwraca w miejscu
list.copy() # zwraca kopie
```

Stos na liście

```
1 stack = [0, 1, 2]
2 stack.append(4)
3 stack.append(5) # [0, 1, 2, 4, 5]
4 stack.pop()
5 stack.pop()
6 stack.pop() # [0, 1]
```

Kolejka

Implementacja przy użyciu listy jest niewydajna: wolne operacje pobierania z początku.

```
1 from collections import deque
2 queue = deque([0, 1, 2])
3 queue.append(4)
4 queue.append(5) # [0, 1, 2, 4, 5]
5 queue.popleft()
6 queue.popleft()
7 queue.popleft() # [4, 5]
```

del

```
1 a = [-1, 0, 2, 7, -5, 11]
2 del a[0] # [0, 2, 7, -5, 11]
3 del a[1:3] # [0, -5, 11]
4 del a[:] # []
5 del a # usuwa listę, a nie tylko jej zawartość
```

Listy składane

```
1 szesciany = []
2 for x in range(10):
3     szesciany.append(x**3)
4
5 szesciany = [x**3 for x in range(10)]
6
7 a = [-4, -2, 0, 2, 4]
8 b = [x for x in a if x > 0]
```

Krotki

```
1 t = 1, 5, 'ala' # (1, 5, 'ala')
2 a = t[0] # 1
3 nt = t, (6, 7) # ((1, 5, 'ala'), (6, 7))
4 t[1] = 3 # błąd, krotki nie można modyfikować
5 empty = () # krotka 0-elementowa
6 singleton = 4, # krotka 1-elementowa
7 singleton = (4,) # to samo
8 x, y, z = t # rozpakowanie krotki
```

Zbiory

```
1  basket = set() # pusty zbiór
2  basket = {'apple', 'orange', 'apple', 'pear'}
3  # {'apple', 'orange', 'pear'}
4  a = set('abracadabra')
5  b = set('alacazam')
6  c = a - b # {'r', 'd', 'b'}
7  c = a | b # {'a', 'c', 'r', 'd', 'b', 'm', 'z',
       ↪ 'l'}
8  c = a & b # {'a', 'c'}
9  c = a ^ b # {'r', 'd', 'b', 'm', 'z', 'l'}
10 c = {x for x in 'abracadabra' if x not in 'abc'}
11 # {'r', 'd'}
```

Słowniki

```
tel = {} # pusty słownik
tel = {'Maja': 4098, 'Jan': 4139}
tel['Basia'] = 4127
del tel['Maja']
tel = dict([('Jan', 4139), ('Jurek', 4127)])
c = {x: x**2 for x in (2, 4, 6)}
# {2: 4, 4: 16, 6: 36}
```

Pętle z wykorzystaniem wbudowanych struktur

```
1  tel = {'Jan': '4563', 'Zbigniew': '4651'}
2  for k, v in tel.items():
3      print(k, v)
4
5  for i, v in enumerate(['raz', 'dwa', 'trzy']):
6      print(i, v)
7
8  for i in reversed(range(1, 10, 2)):
9      print(i)
10
11 warzywa = ['kapusta', 'fasola', 'groch',
       ↪ 'kapusta']
12 for f in sorted(set(warzywa)):
13     print(f)
```

Wyrażenia lambda

```
1 def add(a, b):
2     return a + b
3
4 print((lambda a, b: a + b)(4, 3))
5
6 pairs = [(1, 'one'), (2, 'two'), (3, 'three'),
    ↪ (4, 'four')]
7 pairs.sort(key=lambda pair: pair[1])
8 # [(4, 'four'), (1, 'one'), (3, 'three'), (2,
    ↪ 'two')]
```

Definicja klasy

```
1  class Complex:
2      def __init__(self, realpart, imagpart):
3          self.r = realpart
4          self.i = imagpart
5      def conjugate(self):
6          self.i = -self.i
7
8  x = Complex(2.0, -1.0)
9  a = x.r # 2.0
10 b = x.i # -1.0
11 x.conjugate()
12 print(x.i) # 1.0
```

Obiekty i metody

```
1  class Pies:
2      rodzina = 'Psowate'  # składowa klasy
3      def __init__(self, rasa):
4          self.rasa = rasa # składowa obiektu
5      def f(self):
6          return 'hau hau'
7
8  a = Pies('jamnik')
9  b = Pies('buldog')
10 print(a.rasa) # 'jamnik'
11 print(b.rodzina) # 'Psowate'
12 a.wiek = 8
13 print(a.wiek)
14 del a.wiek
```

Dziedziczenie

```
class KlasaPochodna(KlasaBazowa):
    <instrukcja-1>
    ...
    <instrukcja-N>

class KlasaPochodna(KlasaBazowa1, KlasaBazowa2):
    <instrukcja-1>
    ...
    <instrukcja-N>
```

Dziedziczenie

- wywołanie konstruktora klasy bazowej,
- przeciążanie (redefinicja) metod klasy bazowej w klasie pochodnej,
- definicja nowych pól oraz metod.

```
1 class KlasaPochodna(KlasaBazowa):
2
3     def __init__(self):
4       KlasaBazowa.__init__(self, ....)
5
6     <instrukcja-1>
7     ...
8     <instrukcja-N>
```

Metody statyczne

- metoda nie jest przywiązywana do obiektu tylko do klasy,
- używany jest dekorator *@staticmethod*.

```
1 class Elektrownia:
2   @staticmethod
3   def oblicz_moc(ilość_maszyn):
4     return ilość_maszyn * 5
```

Klasy oraz metody abstrakcyjne

```
1 class AbstrakcyjnaKlasa:
2   def metoda_abstrakcyjna(self):
3     raise NotImplementedError
```

Prywatne składowe

Prywatne składowe nie istnieją, ale ma miejsce dekorowanie nazw:
$\_\_var \rightarrow \_classname\_\_var$

```
>>> class Mapping:
...     def __init__(self, iterable):
...         self.__items_list = iterable
...
>>> m = Mapping([1,2,3])
>>> m.__items_list
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'Mapping' object has no attribute
    ↪ '__items_list'
>>> m._Mapping__items_list
[1, 2, 3]
```

Puste klasy

```
1 class Osoba:
2     pass
3
4 a = Osoba() # Pusty rekord
5
6 # Dodawanie pól
7 a.imie = 'Jan'
8 a.nazwisko = 'Kowalski'
9 a.wiek = 26
```

Iteratory

```
1  class Counter:
2      def __init__(self, low, high):
3          self.current = low
4          self.high = high
5
6      def __iter__(self):
7          return self
8
9      def __next__(self):
10         if self.current > self.high:
11             raise StopIteration
12         else:
13             self.current += 1
14             return self.current - 1
```

Iteratory

```
1 c = Counter(5,10)
2 for i in c:
3   print(i)
4 ...
5 5 6 7 8 9 10
```

Iteratory

Iteratory:

- mogą operować na nieskończonych ciągach,
- oszczędzają zasoby, możliwe jest pobranie kolejnego elementu bez konieczności przechowywanie wszystkiego w pamięci.

Generatory

```
1  def counter_generator(low, high):
2      while low <= high:
3          yield low
4          low += 1
5
6  for i in counter_generator(5,10):
7  ...      print(i)
8  ...
9  5 6 7 8 9 10
```

Generatory

```
1  def r_list():
2      result = []
3      for i in range(10000):
4          result.append(i)
5      return result
6
7  def i_list():
8      for i in range(10000):
9          yield i
```

Wyrażenia generatorowe

```
time x = l_list()   -> 2.81 ms
time x = i_list()   -> 296 us
```

Implementacja generatorowa jest ok. 10 razy szybsza niż tablicowa.

```
x = list(x)
```

Konwersja wartości na napisy

```
>>> s = 'Hello'
>>> str(s)
'Hello'
>>> repr(s)
"'Hello'"
>>> str(1/13)
'0.07692307692307693'
>>> x = 11 * 3.2
>>> s = 'x to ' + repr(x)
>>> print(s)
x to 35.2
>>> powitanie = 'Hello\n'
>>> pwt = repr(powitanie)
>>> print(pwt)
'Hello\n'
```

Formatowanie napisów

```
>>> for x in range(1, 7):
...     print(repr(x).rjust(2),
    ↪ repr(x*x).rjust(3), end=' ')
...     print(repr(x*x*x).rjust(4))
...
 1   1    1
 2   4    8
 3   9   27
 4  16   64
 5  25  125
 6  36  216
```

Formatowanie napisów

```
>>> print('{} wykrzyknął "{}!"'.format('Jan',
    ↪ 'Nie'))
Jan wykrzyknął "Nie!"
>>> print('{1} i {0}'.format('kwadraty',
    ↪ 'trójkąty'))
trójkąty i kwadraty
```

Operacje na plikach

```
>>> f = open('write_file_name', 'w')
>>> f = open('append_file_name', 'a')
>>> f = open('read_file_name', 'r')
>>> f.read()
'Pierwszy wiersz\nDrugi wiersz\n\n'
>>> f.readline()
'Pierwszy wiersz\n'
>>> f.readline()
'Drugi wiersz\n'
>>> f.readline()
'\n'
>>> f.readline()
''
```

Operacje na plikach

```
>>> f.readlines()
['Pierwszy wiersz\n', 'Drugi wiersz\n', '\n']

>>> t = f.readlines()

>>> for line in t:
...     print(line, end='')
...
Pierwszy wiersz
Drugi wiersz
```

Operacje na plikach

```
>>> f.write('Hello\n')
>>> value = 42
>>> f.write(value)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: must be str, not int
>>> s = str(value)
>>> f.write(s)

>>> f.seek(0) # na początek
>>> f.seek(0, 2) # na koniec
```

Operacje na plikach binarnych

```
>>> f = open('workfile', 'rb+')
>>> f.write(b'0123456789abcdef')
16
>>> f.seek(5)      # do 6-go bajtu
5
>>> f.read(1)
b'5'
>>> f.seek(-3, 2) # do 3-go bajtu od końca
13
>>> f.read(1)
b'd'
>>> f.close()
```

Automatyczne zamykanie pliku

```
>>> with open('/tmp/workfile', 'r', encoding =
    ↪ 'utf-8') as f:
...     read_data = f.read()
>>> f.closed
True
```

Błędy składniowe

```
>>> while True print('Hello')
  File "<stdin>", line 1
    while True print('Hello')
                     ^
SyntaxError: invalid syntax
```

Wyjątki

```
>>> 5 * (1/0)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: division by zero
>>> 4 + x*3
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'x' is not defined
>>> '2' + 2
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: Can't convert 'int' object to str
    ↪ implicitly
```

Obsługa

```
1  import sys
2  try:
3      f = open('plik.txt')
4      s = f.readline()
5      i = int(s.strip())
6  except OSError as err:
7      print("Błąd systemu: {0}".format(err))
8  except ValueError:
9      print("Nie można dokonać konwersji.")
10 except:
11     print("Nieoczekiwany wyjątek:",
           ↪ sys.exc_info()[0])
12     raise
```

finally

```
1 def divide(a, b):
2     try:
3         c = a / b
4     except ZeroDivisionError:
5         print('Próba dzielenia przez zero')
6     finally:
7         print('Wykonanie klauzuli finally')
```

i wszystko razem

```
try:
    operation_that_can_throw_ioerror()
except IOError:
    handle_the_exception_somehow()
finally:
    something_we_always_need_to_do()
```

AGH

Część II

Python w obliczeniach

Python w obliczeniach

NumPy

- Wstęp
  - Typy
  - Tablice

tablice

- `ndarray` - tablica
  - wielowymiarowa (dowolny kształt)
  - jednorodna
  - ciągły obszar pamięci

szybciej i prościej

```
>>> from time import time
>>> import numpy as np
>>> def f1(n):
...     t1 = time()
...     X = range(n)
...     Y = range(n)
...     Z = [0]*n
...     for i in range(len(X)):
...         Z[i] = X[i] + Y[i]
...     return time() - t1
...
```

szybciej i prościej

```
>>> def f2(n):
...     t1 = time()
...     X = np.arange(n)
...     Y = np.arange(n)
...     Z = X + Y
...     return time() - t1
...
>>> print(f1(10000000), f2(10000000))
2.677424430847168 0.04493236541748047
```

tablice

```
>>> import numpy as np
# tablica jednowymiarowa, trójelementowa,
# domyślny typ (Python) int
>>> np.array([1,2,3])
array([1, 2, 3])
# tablica dwuwymiarowa, 3x3-elementowa,
# wymuszony typ (C) double
>>> a = [[11,12,13], [21,22,23], [31,32,33]]
>>> np.array(a, dtype=np.double)
array([[ 11.,  12.,  13.],
       [ 21.,  22.,  23.],
       [ 31.,  32.,  33.]])
```

tablice

- typ
- kształt
- układ

NumPy

- Wstęp

- Typy
- Tablice

wybrane wbudowane typy całkowite

typ	opis
`byte`, `ubyte`	C: char, unsigned char
`short`, `ushort`	C: short, unsigned short
`intc`, `uintc`	C: int, unsigned int
`int8`, `uint8`	8 bitów
`int16`, `uint16`	16 bitów
`int32`, `uint32`	32 bity
`int64`, `uint64`	64 bity

wybrane wbudowane typy zmiennopozycyjne

typ	opis
`single`	C: float
`double`	C: double
`float16`	16 bitów
`float32`	32 bity
`float64`	64 bity

wybrane wbudowane typy zespolone

typ	opis
`csingle`	para liczb typu single
`cdouble`	para liczb typu double
`cfloat64`	para liczb typu float32
`cfloat128`	para liczb typu float64

pozostałe wbudowane typy proste

typ	opis
`bool_`	Python: bool
`object_`	Python: object
`void`	typ pusty

NumPy

- Wstęp
- Typy
- Tablice
  - Konstrukcja
  - Operacje

kształt

- krotka
- determinuje wymiar i rozmiary
  - `(3,)` – wektor 3-elementowy
  - `(3,3)` – macierz kwadratowa 3x3
  - `(3,1)` – 3 wiersze, 1 kolumna
  - `(1,3)` – 1 wiersz, 3 kolumny
- atrybut `shape`
  - informacja o kształcie

konstrukcja tablic

- o zadanym kształcie, opcjonalnie wypełnionych
- z danych
- z pliku
- z obszaru pamięci

konstrukcja tablic o zadanym kształcie

- `empty(shape, dtype=float)`
- `empty_like(a, dtype=None)`
- `eye(N, M=None, k=0, dtype=float)`
- `identity(n, dtype=float)`
- `ones(shape, dtype=float)`
- `ones_like(a, dtype=None)`
- `zeros(shape, dtype=float)`
- `zeros_like(a, dtype=None)`
- `full(shape, fill_value, dtype=None)`
- `full_like(a, fill_value, dtype=None)`

konstrukcja tablic o zadanym kształcie

```
>>> import numpy as np
>>> e = np.empty((2,3))
>>> e
array([[  6.92831614e-310,   6.92831614e-310,
    ↪ 1.58101007e-322],
       [  4.74303020e-322,   1.23973244e-316,
          ↪ 6.92831444e-310]])
>>> np.ones_like(e)
array([[ 1.,  1.,  1.],
       [ 1.,  1.,  1.]])
>>> np.zeros_like(e)
array([[ 0.,  0.,  0.],
       [ 0.,  0.,  0.]])
```

konstrukcja tablic o zadanym kształcie

```
>>> np.empty_like(e)
array([[  6.92831614e-310,   6.92831614e-310,
    ↪ 1.09417596e-316],
       [  6.92810406e-310,   8.26821625e-317,
          ↪ 8.26822020e-317]])
>>> np.full((3,), 42)
array([ 42.,  42.,  42.])
>>> np.full((3,), [2,3,5])
array([ 2.,  3.,  5.])
>>> np.eye(2,3)
array([[ 1.,  0.,  0.],
       [ 0.,  1.,  0.]])
```

konstrukcja tablic z danych

```
>>> import math
>>> import numpy as np
>>> a = np.array([0,1,2,3])
>>> a
array([0, 1, 2, 3])
>>> a.shape
(4,)
>>> a.dtype
dtype('int64')
>>> b = np.array([0,1.0,2,3])
>>> b
array([ 0.,  1.,  2.,  3.])
>>> b.dtype
dtype('float64')
```

konstrukcja tablic z danych

```
>>> c = np.array([[math.sqrt(i*j) for j in
    ↪ range(0,3)]
...                for i in range(0,3)])
>>> c
array([[ 0.        ,  0.        ,  0.        ],
       [ 0.        ,  1.        ,  1.41421356],
       [ 0.        ,  1.41421356,  2.        ]])
>>> c.dtype
dtype('float64')
```

konstrukcja tablic

```
>>> import numpy as np
>>> np.fromstring('przykład', dtype=np.uint8)
array([112, 114, 122, 121, 107, 197, 130,  97,
    ↪ 100], dtype=uint8)
>>> np.fromstring('3 2 1', dtype=np.uint8, sep='
    ↪ ')
array([3, 2, 1], dtype=uint8)
>>> iterable = (x*x for x in range(5))
>>> np.fromiter(iterable, np.float)
array([  0.,   1.,   4.,   9.,  16.])
>>> np.fromfunction(lambda i, j: i == j, (3, 3))
array([[ True, False, False],
       [False,  True, False],
       [False, False,  True]], dtype=bool)
```

konstrukcja siatek

- `arange(start, stop, step, dtype=None)`
- `linspace(start, stop, num=50, endpoint=True, retstep=False)`
- `logspace(start, stop, num=50, endpoint=True, base=10.0)`

konstrukcja siatek

```
>>> import numpy as np
>>> np.arange(0,10,2)
array([0, 2, 4, 6, 8])
>>> np.arange(0,10)
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
>>> np.linspace(2,3,4)
array([ 2.        ,  2.33333333,  2.66666667,  3.
    ↪         ])
>>> np.linspace(2,3,num=4,retstep=True)
(array([ 2.        ,  2.33333333,  2.66666667,
    ↪ 3.        ]),
 0.3333333333333333)
>>> np.logspace(2,3,num=4)
array([  100.        ,   215.443469  ,
    ↪ 464.15888336,  1000.        ])
```

konstrukcja macierzy

- `diag(v, k=0)`
  - macierz diagonalna
  - diagonala macierzy
- `diagflat(v, k=0)`
  - macierz diagonalna
- `tri(N, M=None, k=0, dtype=float)`
  - dolna jedynkowa macierz trójkątna
- `tril(m, k=0)`
  - dolny trójkąt macierzy `m`
- `triu(m, k=0)`
  - górny trójkąt macierzy `m`

konstrukcja macierzy

```
>>> import numpy as np
>>> np.diag([2,3])
array([[2, 0],
       [0, 3]])
>>> np.diag([2,3],-1)
array([[0, 0, 0],
       [2, 0, 0],
       [0, 3, 0]])
>>> np.diag(np.diag([2,3],-1),-1)
array([2, 3])
```

konstrukcja macierzy

```
>>> np.diagflat(np.diag([2,3]))
array([[2, 0, 0, 0],
       [0, 0, 0, 0],
       [0, 0, 0, 0],
       [0, 0, 0, 3]])
>>> np.tri(2)
array([[ 1.,  0.],
       [ 1.,  1.]])
>>> np.tril([[1,2],[3,4]])
array([[1, 0],
       [3, 4]])
```

konwersja

- `ndarray.tolist()`
- `asscalar(a)`
- `ndarray.tostring()`

konwersja

```
>>> import numpy as np
>>> np.identity(2).tolist()
[[1.0, 0.0], [0.0, 1.0]]
```

zmiana kształtu

- `ndarray.shape`
- `reshape(a, newshape)`
- `ndarray.reshape(shape)`
- `ravel(a)`
- `ndarray.flatten()`

zmiana kształtu

```
>>> import numpy as np
>>> a = np.arange(4)
>>> a
array([0, 1, 2, 3])
>>> a.shape=(2,2)
>>> a
array([[0, 1],
       [2, 3]])
>>> np.reshape(a, 4)
array([0, 1, 2, 3])
>>> np.ravel(a,order='F')
array([0, 2, 1, 3])
```

zmiana rozmiaru tablicy

- `ndarray.resize(new_shape)` – dopełnianie zerami
- `resize(a, new_shape)` – dopełnianie elementami `a`

zmiana rozmiaru tablicy

```
>>> import numpy as np
>>> a = np.array([[0, 1], [2, 3]])
>>> a.resize((2, 1))
>>> a
array([[0],
       [1]])
```

zmiana rozmiaru tablicy

```
>>> b = np.array([[0, 1], [2, 3]])
>>> b.resize(2, 3)
>>> b
array([[0, 1, 2],
       [3, 0, 0]])
>>> c = a
>>> a.resize((1, 1))
Traceback (most recent call last):
...
ValueError: cannot resize an array that has been
    ↪ referenced ...
```

wstawianie i usuwanie elementów

- `insert(arr, obj, values, axis=None)`
- `append(arr, values, axis=None)`
- `delete(arr, obj, axis=None)`
- `trim_zeros(filt, trim='fb')`

`axis`

jeżeli `None`, tablica jest spłaszczana

wstawianie i usuwanie elementów

```
>>> import numpy as np
>>> a = np.array([[1, 1], [2, 2], [3, 3]])
>>> a
array([[1, 1],
       [2, 2],
       [3, 3]])
>>> np.insert(a, 1, 5)
array([1, 5, 1, 2, 2, 3, 3])
>>> np.insert(a, 1, 5, axis=1)
array([[1, 5, 1],
       [2, 5, 2],
       [3, 5, 3]])
```

transpozycja

- `ndarray.T`
- `transpose(a, axes=None)`
- `swapaxes(a, axis1, axis2)`

transpozycja

```
>>> a
array([[0, 2, 4, 6, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0],
       [0, 2, 4, 6, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0]])
>>> a.T
array([[0, 0, 0, 0],
       [2, 0, 2, 0],
       [4, 0, 4, 0],
       [6, 0, 6, 0],
       [0, 0, 0, 0],
       [0, 0, 0, 0],
       [0, 0, 0, 0]])
```

operacje arytmetyczne

- `+, -, *, /, **`
- działają na pojedynczych elementach
- uzgadnianie kształtu

operacje arytmetyczne

```
>>> import numpy as np
>>> np.arange(3) * np.arange(3)
array([0, 1, 4])
>>> np.arange(3) + np.transpose([np.arange(3)])
array([[0, 1, 2],
       [1, 2, 3],
       [2, 3, 4]])
>>> np.arange(3) * np.transpose([np.arange(3)])
array([[0, 0, 0],
       [0, 1, 2],
       [0, 2, 4]])
```

operatory relacyjne

- `<, <=, ==, !=, >=, >`
- działają na pojedynczych elementach
- uzgadnianie kształtu
- w wyniku tablica boolowska

operatory relacyjne

```
>>> import numpy as np
>>> x = np.arange(8)
>>> array_repr(x > 4)
array([False, False, False, False, False,  True,
    ↪ True,  True],
      dtype=bool)
>>> x == 4
array([False, False, False, False,  True, False,
    ↪ False, False],
      dtype=bool)
```

funkcje uniwersalne

- funkcje działające na elementach tablic, uwzględniające
  - uzgadnianie kształtu
  - rzutowanie typów
- operacje arytmetyczne
- operatory relacyjne
- operacje bitowe
- funkcje elementarne
- możliwość definiowania własnych

funkcje uniwersalne

```
>>> import numpy as np
>>> a = np.arange(10,101,10)
>>> np.log10(a)
array([ 1.        , 1.30103   , 1.47712125,
    ↪ 1.60205999, 1.69897,
        1.77815125, 1.84509804, 1.90308999,
            ↪ 1.95424251, 2.        ])
>>> np.log10(10.0**a)
array([  10., 20., 30., 40., 50., 60., 70., 80.,
    ↪ 90., 100.])
```

indeksowanie

- `a[i,...]` – pojedynczy element lub podtablica jeżeli liczba indeksów mniejsza niż wymiar
- `a[start:stop:krok,...]` – wycinek
  - `start`,`stop`,`krok` opcjonalne
  - ujemne indeksy od końca
- `a[tablica_indeksów]` – wybrane elementy; kształt wyniku taki jak kształt tablicy indeksów
- `a[wektor_indeksów_1,wektor_indeksów_2,...]` – wybrane elementy (lub podtablice) na węzłach siatki
- `a[maska]` – wybrane elementy
  - możliwość konstrukcji indeksów o dynamicznie ustalanej liczbie wymiarów
  - `slice()` umożliwia specyfikację wycinków
- możliwość łączenia różnych sposobów indeksowania

indeksowanie

```
>>> import numpy as np
>>> x = np.arange(10)
>>> x[2]
2
>>> x[-2]
8
>>> x.shape = (2,5)
>>> x[0]
array([0, 1, 2, 3, 4])
>>> x[1,3]
8
>>> x[1,-1]
9
>>> x[0][2]
2
```

indeksowanie

```
>>> x[1,1:2]
array([6, 8])
>>> x[[1,1],[2,3]]
array([7, 8])
```

iloczyny

- `dot(a, b)`
- `inner(a, b)`

Python w obliczeniach

SciPy

- Stałe
- Algebra liniowa
- Całkowanie
- Szybka transformata Fouriera

stałe matematyczne

- `Inf`
- pakiet `scipy.constants`
  - `pi`
  - `golden`

stałe fizyczne

- pakiet `scipy.constants`
  - `c`
  - `G`
  - `g`
  - `e`
  - `sigma`
  - `Wien`
  - `Rydberg`
  - itd...
- wszystkie wartości w układzie SI.

SciPy

- Stałe
- Algebra liniowa
- Całkowanie
- Szybka transformata Fouriera

podstawowe operacje na macierzach

- iloczyny, dodawanie, indeksowanie itd z `numpy`
- `linalg.inv(a)`
- `linalg.det(a)`

podstawowe operacje na macierzach

```
>>> import numpy as np
>>> from scipy import linalg
>>>
>>> A = np.array([[1,2,3],[4,5,6],[7,8,9]])
>>> linalg.det(A)
0.0
>>> B = np.dot(A,A)
>>> B
array([[ 30,  36,  42],
       [ 66,  81,  96],
       [102, 126, 150]])
```

podstawowe operacje na macierzach

```
>>> linalg.inv(B)
array([[  1.40737488e+14,  -2.81474977e+14,
    ↪ 1.40737488e+14],
       [ -2.81474977e+14,   5.62949953e+14,
          ↪ -2.81474977e+14],
       [  1.40737488e+14,  -2.81474977e+14,
          ↪ 1.40737488e+14]])
```

układy równań

- `linalg.solve(a, b)`
- `linalg.solve_triangular(a, b)`

układy równań

```
>>> import numpy as np
>>> from scipy import linalg
>>> A = np.array([[3,2,0],[1,-1,0],[0,5,1]])
>>> b = np.array([2,4,-1])
>>> linalg.solve(A,b)
array([ 2., -2.,  9.])
```

dekompozycje

- `linalg.eig(a, b=None)`
- `linalg.svd(a)`
- `linalg.lu(a)`
- `linalg.cholesky(a)`
- `linalg.qr(a)`
- `linalg.rq(a)`
- `...`

diagonalizacja

```
>>> from scipy.linalg import eig
>>> A = np.array([[1,2],[2,3]])
>>> eig(A)
(array([-0.23606798+0.j,  4.23606798+0.j]),
 array([[-0.85065081, -0.52573111],
        [ 0.52573111, -0.85065081]]))
```

SciPy

- Stałe
- Algebra liniowa
- Całkowanie
- Szybka transformata Fouriera

całkowanie numeryczne

- podstawowe, adaptacyjne procedury całkowania numerycznego
  - `integrate.quad(func,a,b)`
  - `integrate.dblquad(func,a,b)`
  - `integrate.tplquad(func,a,b)`
  - `integrate.nquad(func, ranges, args=None, opts=None)`
- ponadto dostępne bardziej wyspecjalizowane funkcje

całkowanie numeryczne

```
>>> from scipy import integrate, constants, Inf
>>> import math
>>>
>>> integrate.quad(math.sin, 0, constants.pi)
(2.0, 2.220446049250313e-14)
>>>
>>> integrate.quad(lambda x: math.exp(-x), 0, Inf)
(1.0000000000000002, 5.842606742906004e-11)
>>> integrate.dblquad(lambda
    ↪ x,y:math.exp(-x**2-y**2),
...                    -Inf, Inf,
...                    lambda x:-Inf, lambda x:Inf)
(3.141592653589777, 2.5173086737433208e-08)
```

SciPy

- Stałe
- Algebra liniowa
- Całkowanie
- Szybka transformata Fouriera

FFT

- `fftpack.fft(x)`
- `fftpack.ifft(x)`
- `fftpack.fft2(x)`
- `fftpack.ifft2(x)`
- `fftpack.fftn(x)`
- `fftpack.ifftn(x)`

Python w obliczeniach

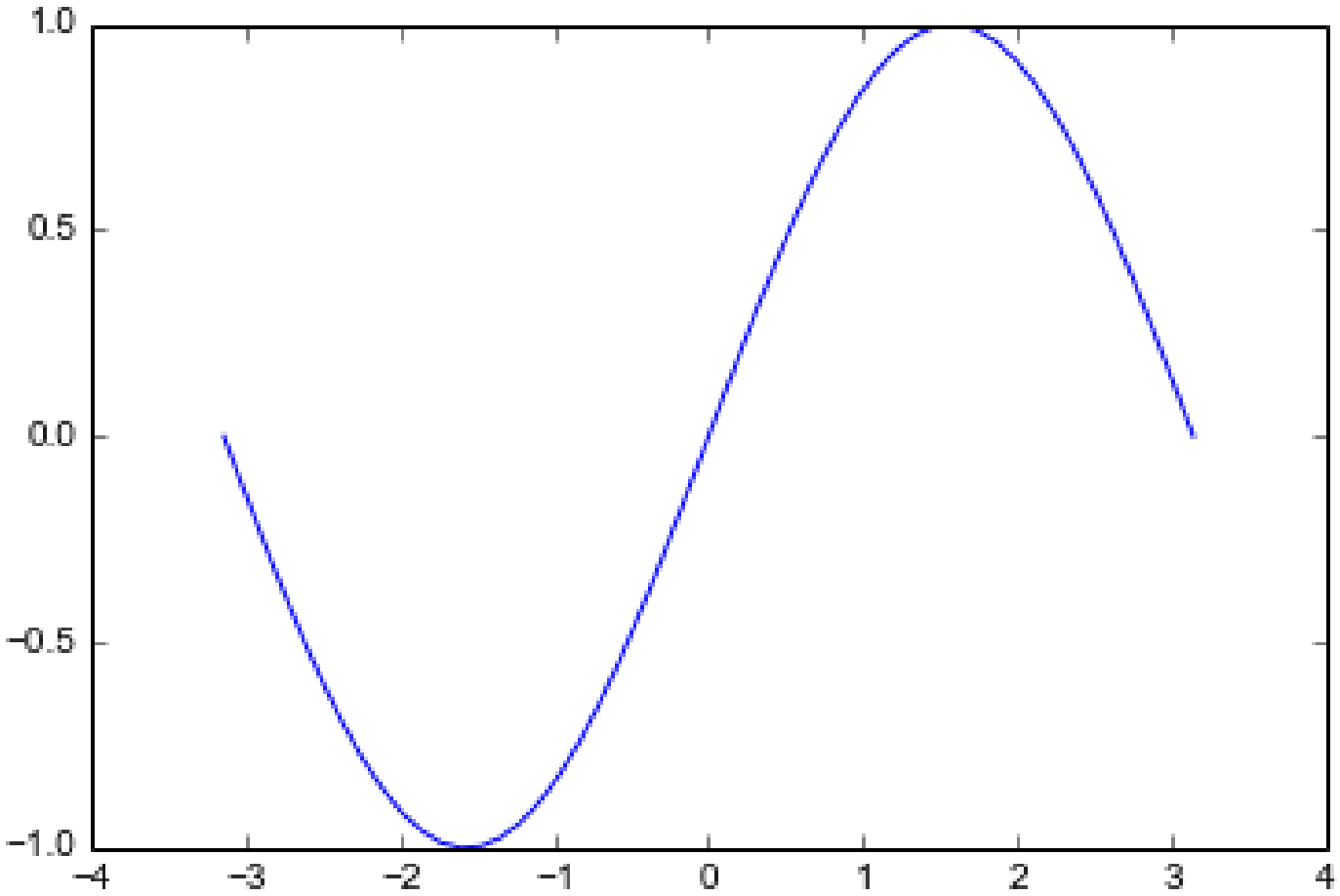
matplotlib

- zaawansowana biblioteka do wykresów dwuwymiarowych
- prosty interfejs `matplotlib.pyplot`
- generacja
  - PS
  - PDF
  - PNG
  - SVG
  - ...

pyplot

```
>>> import matplotlib.pyplot as plt
>>> import math
>>> import numpy as np
>>> from scipy.constants import pi
>>> x = np.linspace(-pi, pi)
>>> plt.plot(x, np.sin(x))
>>> plt.show()
```

pyplot



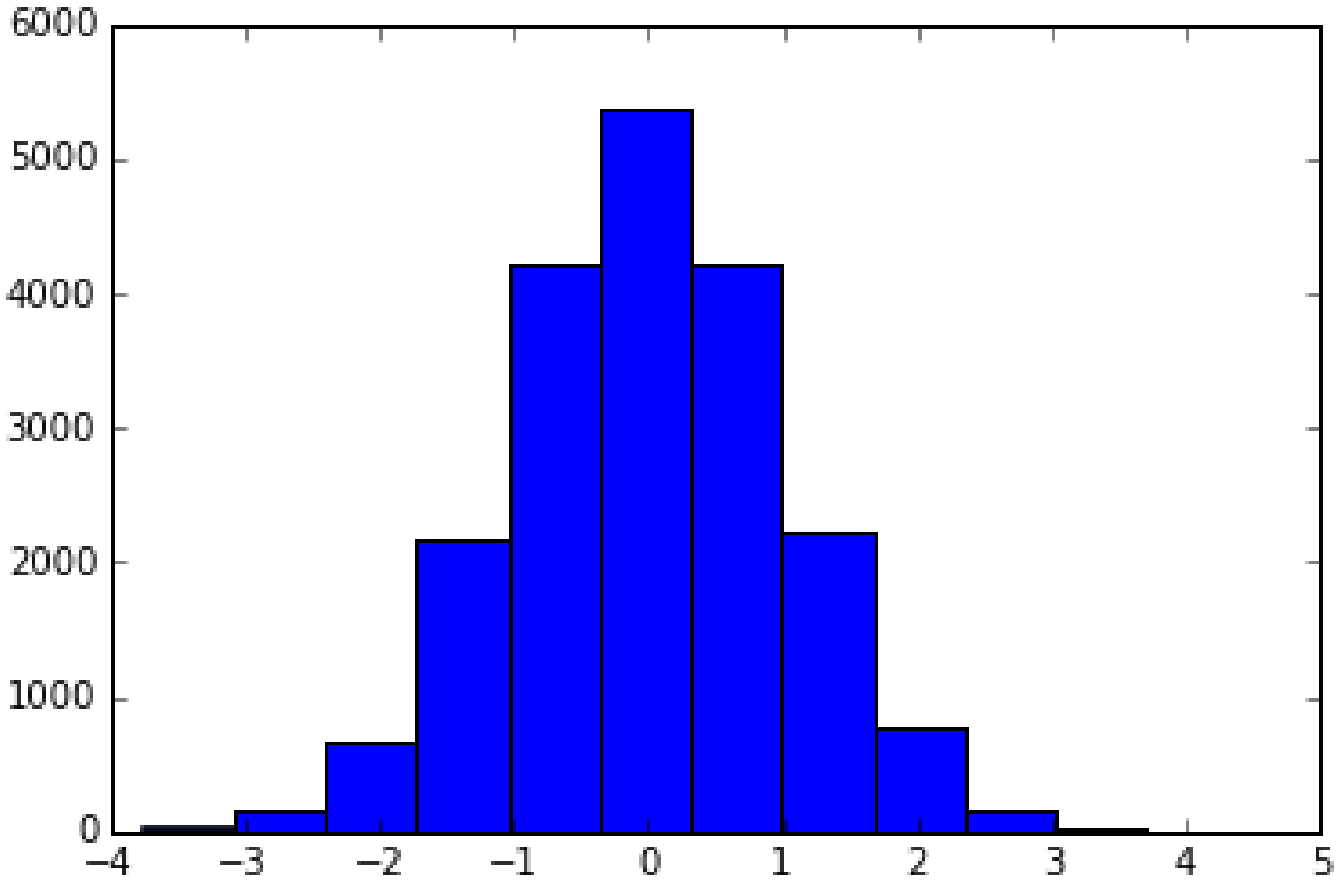
rysowanie wykresów w minutę

- `matplotlib.pyplot.plot(*args, **kwargs)`
  - zestawy danych
  - własności linii
    - kolory: 'b', 'g', 'r', 'c', 'm', 'y', 'k', 'w'
    - linia: '-', '–', '-.', ':', '.', ',', 'o', 'v', '^', '<', '>', '1', '2', '3', '4', 's', 'p', '*', 'h', 'H', '+', 'x', 'D', 'd', '|', '_'
    - wiele bardziej zaawansowanych możliwości
- operacje modyfikujące wykres
  - `matplotlib.pyplot.xlabel(s, *args, **kwargs)`
  - `matplotlib.pyplot.ylabel(s, *args, **kwargs)`
  - ...

histogramy

```
>>> import matplotlib.pyplot as plt
>>> from numpy.random import normal
>>> gaussian_numbers = normal(size=20000)
>>> plt.hist(gaussian_numbers, bins=12)
>>> plt.show()
```

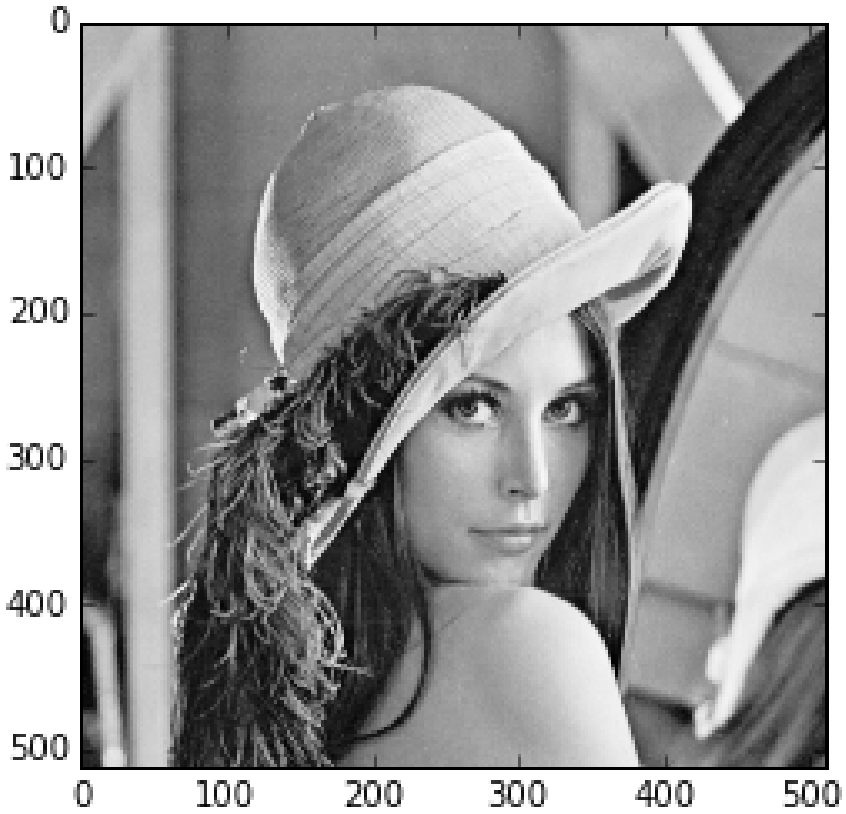
histogramy



obrazki

```
>>> import scipy
>>> import matplotlib
>>> img=scipy.misc.lena()
>>> print(img)
[[162 162 162 ..., 170 155 128]
 [162 162 162 ..., 170 155 128]
 [162 162 162 ..., 170 155 128]
 ...,
 [ 43  43  50 ..., 104 100  98]
 [ 44  44  55 ..., 104 105 108]
 [ 44  44  55 ..., 104 105 108]]
>>> matplotlib.pyplot.imshow(img, cmap =
    ↪ matplotlib.cm.Greys_r)
```

obrazki



Python w obliczeniach

macierz odwrotna

Oblicz macierz odwrotną do macierzy

$$A = \begin{pmatrix} 1 & 3 & 5 \\ 2 & 5 & 1 \\ 2 & 3 & 8 \end{pmatrix}$$

macierz odwrotna – rozwiązanie

$$A^{-1} \approx \begin{pmatrix} -1.48 & 0.36 & 0.88 \\ 0.56 & 0.08 & -0.36 \\ 0.16 & -0.12 & 0.04 \end{pmatrix}$$

```
1 import numpy as np
2 from scipy import linalg
3 A = np.array([[1,3,5],[2,5,1],[2,3,8]])
4 linalg.inv(A)
```

układu równań

Znajdź rozwiązanie układu równań

$$\begin{aligned} x + 3y + 5z &= 10 \quad (1) \\ 2x + 5y + z &= 8 \quad (2) \\ 2x + 3y + 8z &= 3 \quad (3) \end{aligned}$$

Sprawdź poprawność obliczeń wykorzystując $A^{-1}b - x = 0$

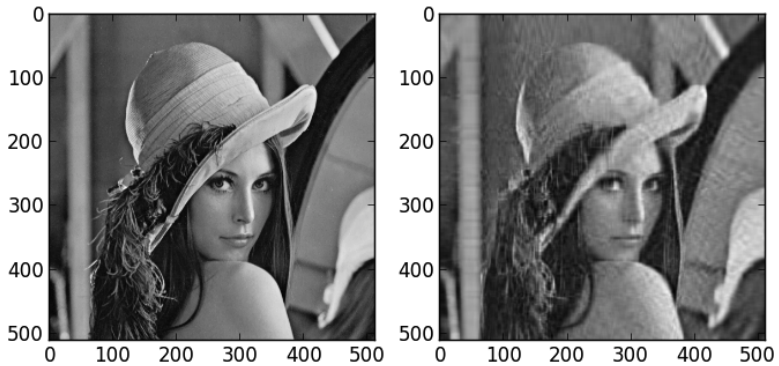
układ równań – rozwiązanie

$$x \approx -9.28,\ y \approx 5.16,\ z \approx 0.76$$

```
1 import numpy as np
2 from scipy import linalg
3 A = np.array([[1,3,5],[2,5,1],[2,3,8]])
4 b = np.array([[10],[8],[3]])
5 xyz = linalg.solve(A,b)
6 linalg.inv(A).dot(b) - xyz
```

kompresja obrazu

Dysponując obrazem 512x512 pikseli w skali szarości dokonaj kompresji tak, by zredukować przestrzeń zajmowaną przez obraz z 262144 jednostek pamięci do 32800 ((2*32*512) + 32). W celu uzyskania takiego stopnia kompresji należy wykorzystać 32 największe wartości osobliwe.

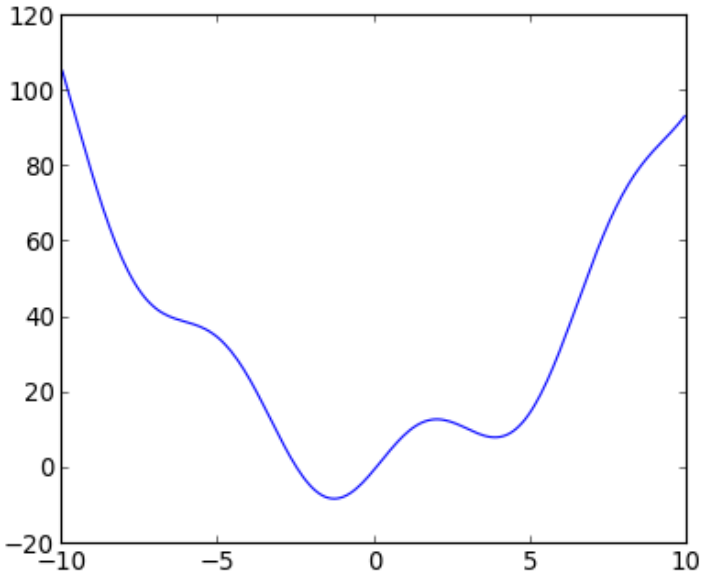


kompresja obrazu – rozwiązanie

```
1  import numpy as np
2  from scipy.linalg import svd
3  import matplotlib.pyplot as plt
4  from scipy import misc
5  img=misc.lena()
6  U,s,Vh=svd(img)
7  s_dot = np.dot(np.diag(s[0:32]), Vh[0:32,:])
8  A = np.dot( U[:,0:32], s_dot)
9  plt.subplot(121,aspect='equal')
10 plt.imshow(img, cmap=matplotlib.cm.Greys_r)
11 plt.subplot(122,aspect='equal')
12 plt.imshow(A, cmap=matplotlib.cm.Greys_r)
13 plt.show()
```

optymalizacja

Znajdź minimum funkcji $y = x^2 + 10\sin(x)$. Sprawdź, czy jest to minimum globalne.



optymalizacja – rozwiązanie

Minima:

- $x \approx 3.8$
- $x \approx -1.3$ (globalne)

optymalizacja – rozwiązanie

```
1  from scipy import optimize, arange, sin
2  import matplotlib.pyplot as plt
3
4  def f(x):
5      return x**2 + 10*sin(x)
6
7  x = arange(-10, 10, 0.1)
8  plt.plot(x, f(x))
9  plt.show()
10
11 xmin_local_0 = optimize.minimize(f, 0)
12 xmin_local_1 = optimize.minimize(f, 3)
13 grid = (-10, 10, 0.1)
14 xmin_global = optimize.brute(f, (grid,))
```

całkowanie

Oblicz wartość funkcji:

$$f(x) = \int_1^{\infty} \frac{e^{-xt}}{t^3} dt$$

w punktach 1, 1.5, 2, 2.5, 3, 3.5.

całkowanie – rozwiązanie

```
1  from scipy.integrate import quad
2  from numpy import vectorize, arange, Inf, exp
3
4  def integrand(t, n, x):
5      return exp(-x*t) / t**n
6
7  def expint(n, x):
8      return quad(integrand, 1, Inf, args=(n, x))[0]
9
10 vec_expint = vectorize(expint)
11 vec_expint(3,arange(1.0,4.0,0.5))
12
13 # alternatywa
14 #from scipy import special
15 #*special.expn(3,arange(1.0,4.0,0.5))
```

miejsca zerowe funkcji

Znajdź miejsce zerowe funkcji

$$f(x) = 4x^3 + 3x^2 + 2x + 1$$

miejsca zerowe funkcji – rozwiązanie

```
1  from scipy.optimize import fsolve
2  from numpy import linspace
3  import matplotlib.pyplot as plt
4
5  f = lambda x: 4*x**3+3*x**2+2*x+1
6  x0 = fsolve(f,10)
7  print (x0)
8  x = linspace(-1,1,50)
9  plt.plot(x,f(x),x0,f(x0),'ro')
10 plt.grid(b=1)
11 plt.show()
```