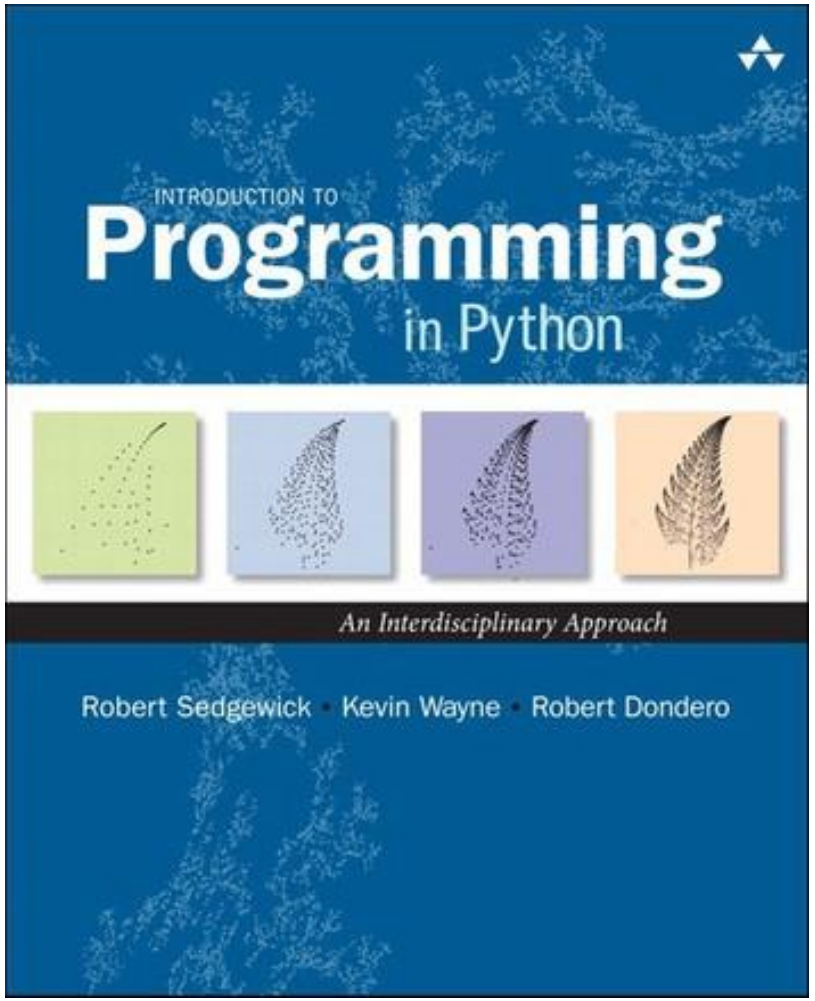
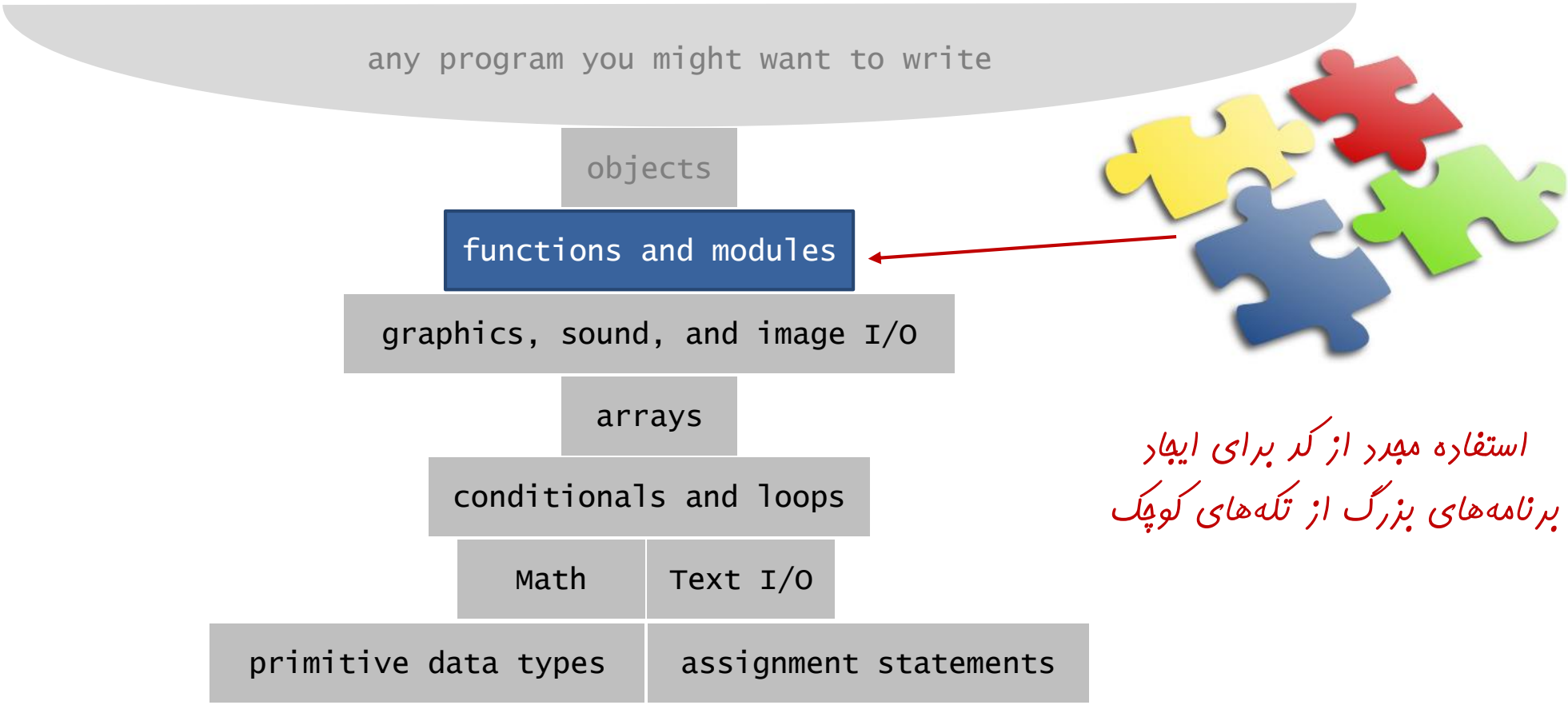
توابع و کتابخانه‌ها: توابع

سید ناصر رضوی **www.snrazavi.ir**
۱۳۹۸

۲-۱ توابع


INTRODUCTION TO
Programming
in Python
An Interdisciplinary Approach
Robert Sedgewick · Kevin Wayne · Robert Dondero

عناصر برنامه‌نویسی

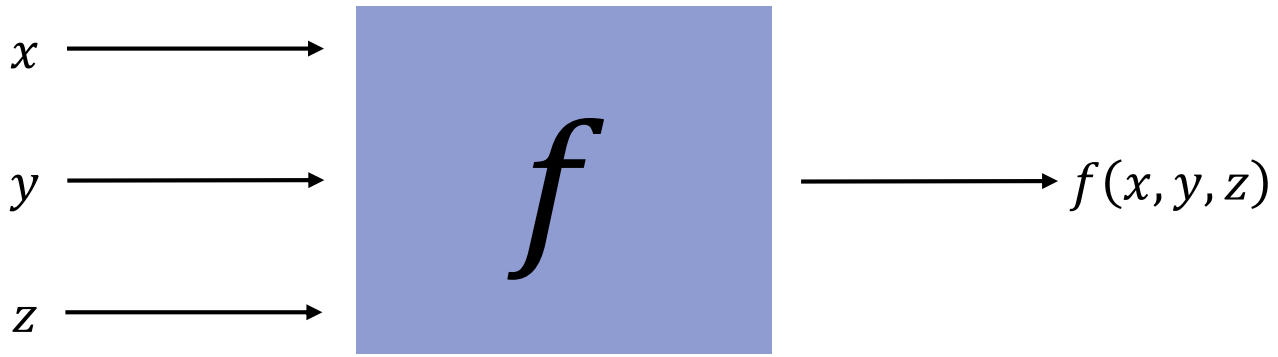


استفاده مجدد از کد برای ایجاد
برنامه‌های بزرگ از تکه‌های کوچک

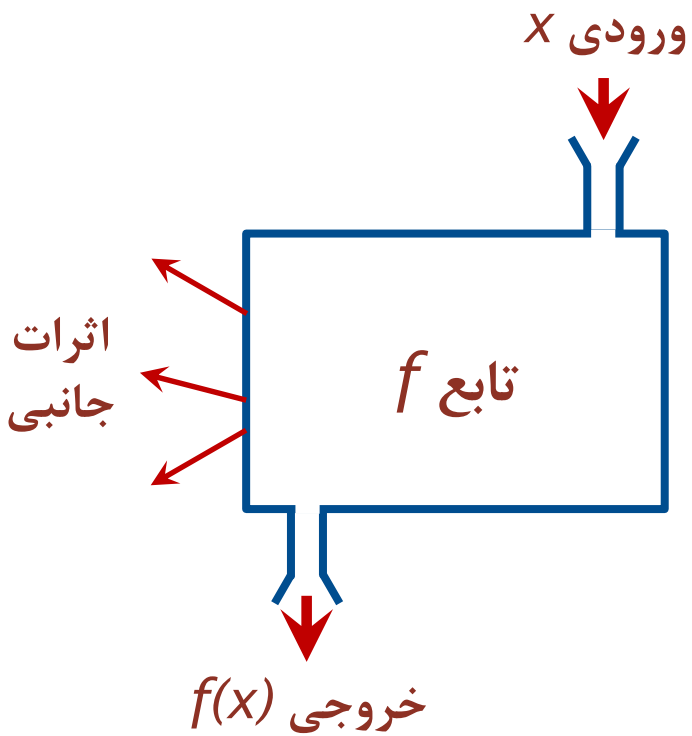
توابع، کتابخانه‌ها و ماجول‌ها

- برنامه‌نویسی ماجولار (پیمانه‌ای).
  - سازمان‌دهی برنامه به صورت مجموعه‌ای از ماجول‌های مستقل که در کنار یکدیگر کاری را انجام می‌دهند.
  - چرا؟ سهولت اشتراک‌گذاری و استفاده مجدد از کد برای ایجاد برنامه‌های بزرگ.

۲-۱ توابع

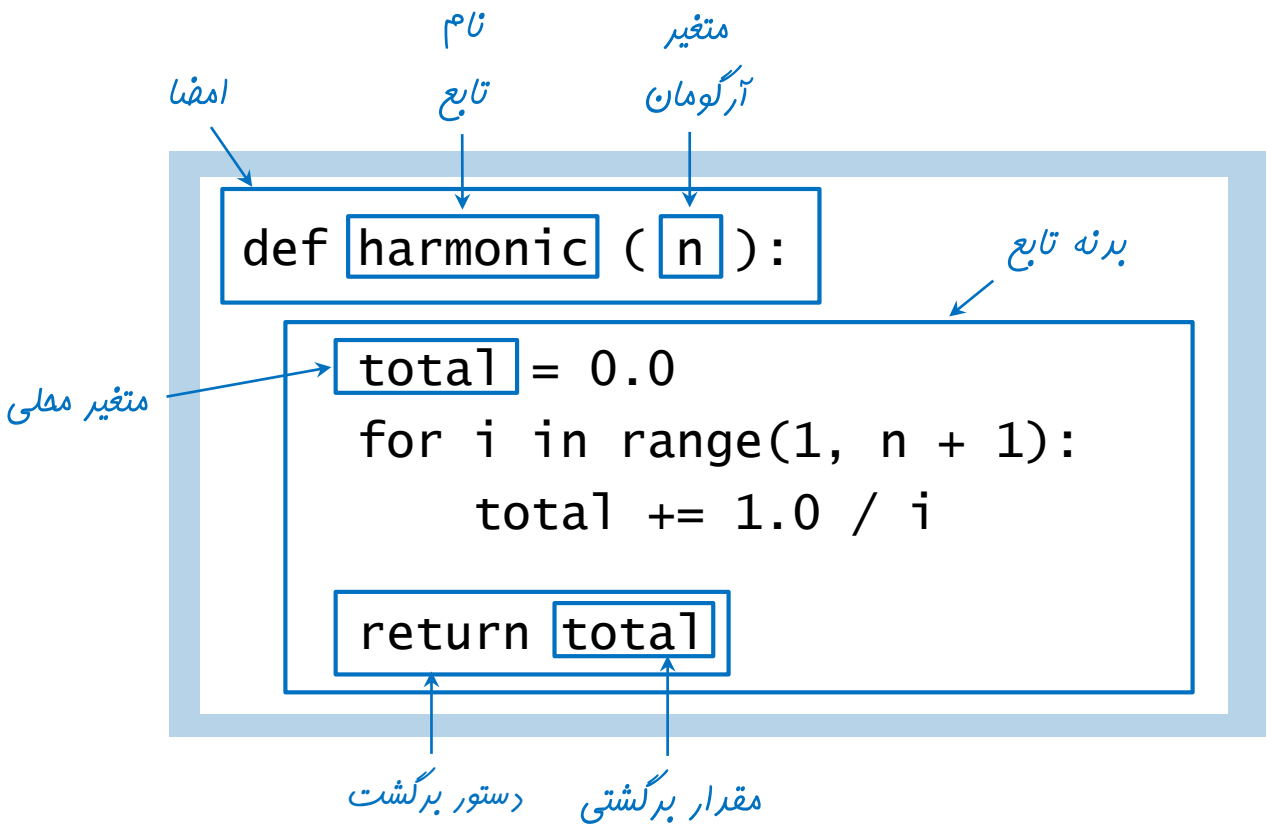


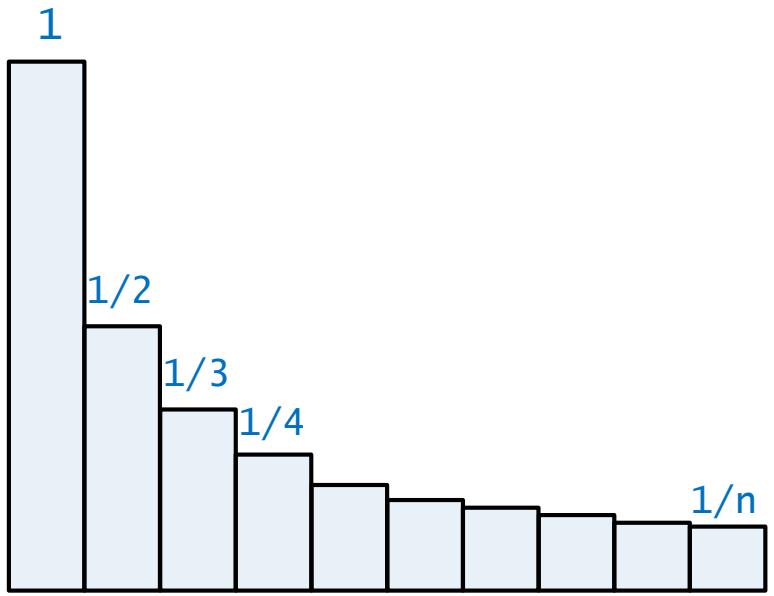
توابع



- تابع پایتون.
  - دریافت صفر یا چندکمیت به عنوان ورودی.
  - برگرداندن صفر یا یک کمیت به عنوان خروجی.
  - اثرات جانبی (مانند نوشتن در خروجی یا ترسیم نمودار).
- کاربردها.
  - دانشمندان از توابع ریاضی برای محاسبه فرمول‌ها استفاده می‌کنند.
  - برنامه‌نویس‌ها از توابع برای ایجاد برنامه‌های پیمانه‌ای استفاده می‌کنند.
  - شما از توابع برای هر دو منظور استفاده می‌کنید.
- مثال‌هایی که تا کنون دیده‌اید
  - توابع پیش‌ساخته: `print()`، `str()`، `int()`، `random.random()` و `math.sqrt()`
  - کتابخانه‌های ورودی/خروجی: `stdio.readInt()`، `stddraw.line()` و `stdaudio.play()`

ساختار یک تابع پایتون





ساختار یک برنامه پایتون

- ساختار یک برنامه پایتون.
  - دنباله‌ای از دستورات import
  - دنباله‌ای از تعریف توابع
  - بخش سراسری یا بدنه برنامه (اختیاری)

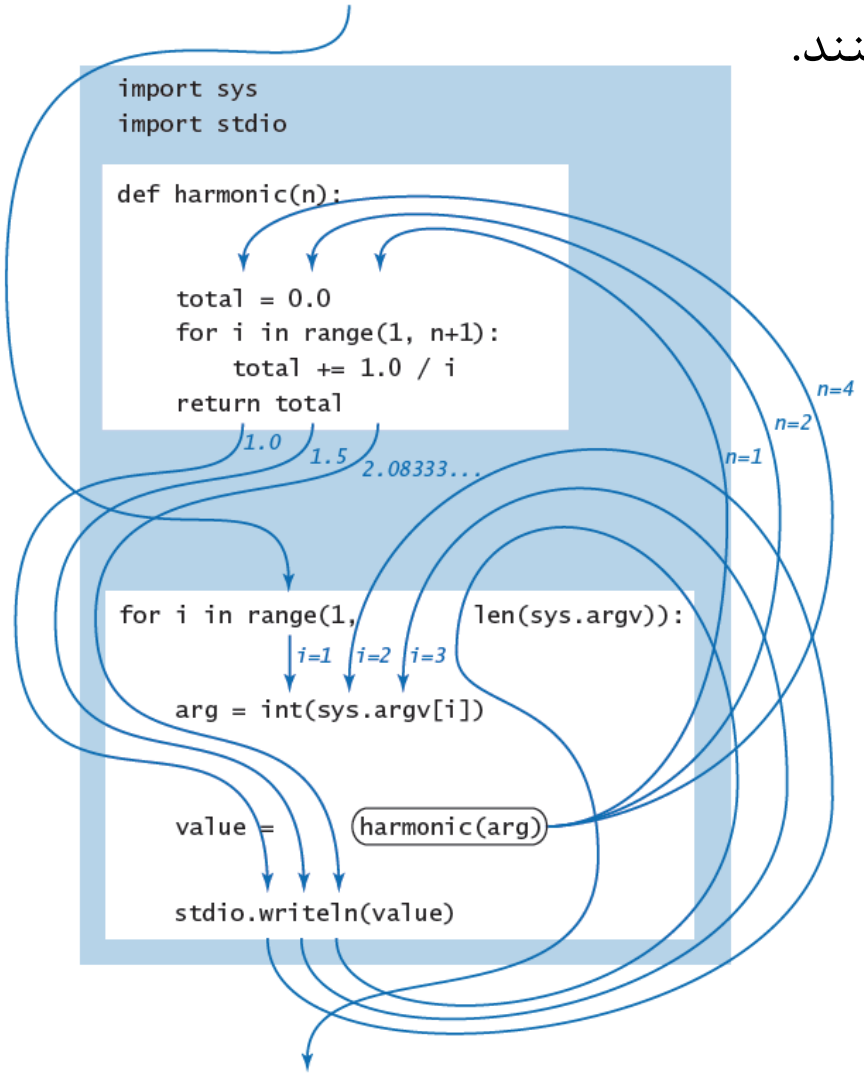
```
import sys

def harmonic(n):
    total = 0.0
    for i in range(1, n + 1):
        total += 1.0 / i
    return total

for i in range(1, len(sys.argv)):
    arg = int(sys.argv[i])
    value = harmonic(arg)
    print(value)
```

```
% python harmonicf.py 1 2 4
1.0
1.5
2.083333333333333
```

جریان کنترل

- نکته کلیدی. توابع یک روش جدید به منظور کنترل جریان اجرا فراهم می‌کنند.

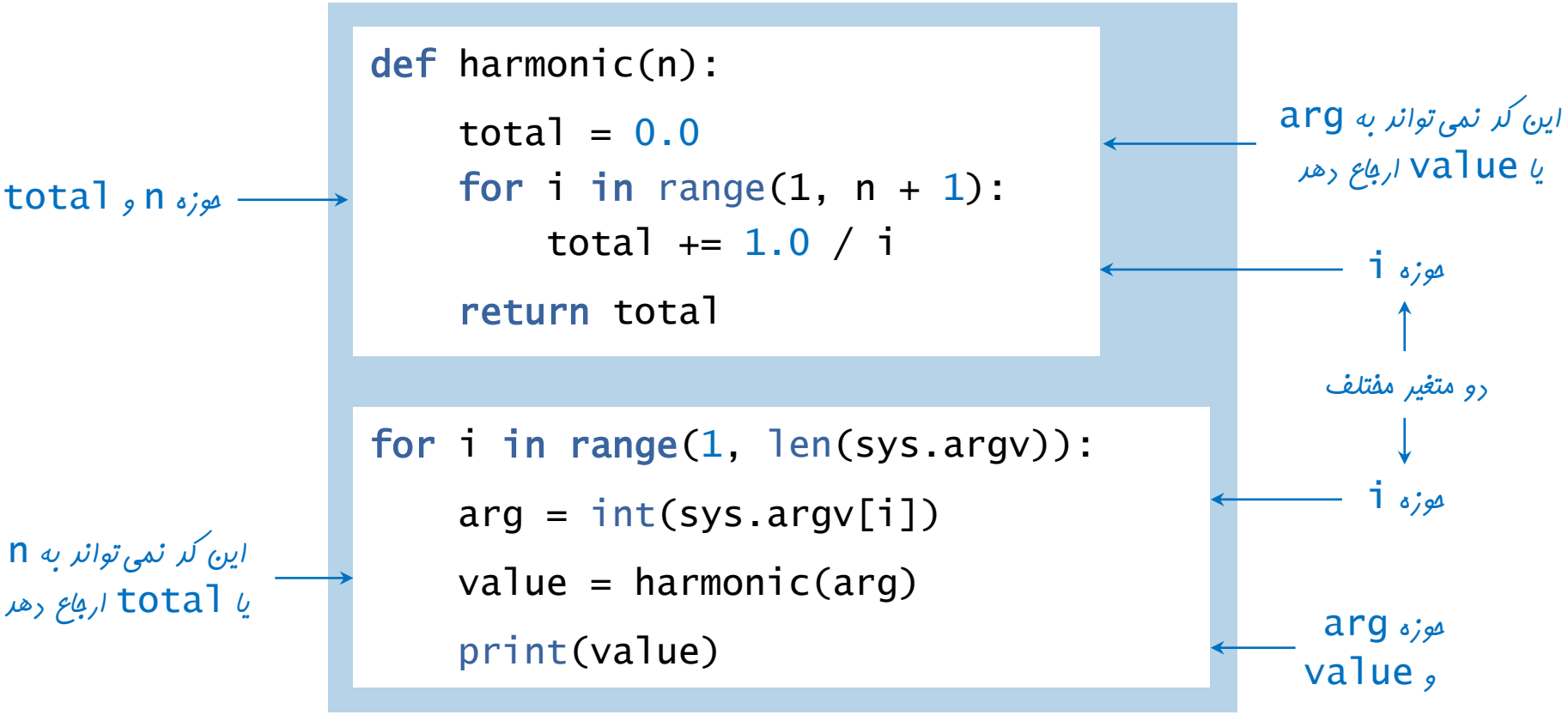
- مراحل فراخوانی یک تابع.
  - کنترل به ابتدای کد تابع فراخوانی شده منتقل می‌شود.
  - آرگومان‌ها با استفاده از مقادیر موجود در فراخوانی مقداردهی می‌شوند.
  - کد تابع اجرا می‌شود.
  - به جای نام تابع در کد فراخوان مقدار برگشتی تابع فراخوانی شده قرار داده می‌شود.
  - کنترل دوباره به کد فراخوان منتقل می‌شود.




```
% python harmonicf.py 1 2 4
1.0
1.5
2.083333333333333
```

حوزه

- حوزه (متغیر). مجموعه دستوراتی که می‌توانند به طور مستقیم به آن متغیر ارجاع دهند.
  - مثال: حوزه یک متغیر برابر است با کد پس از اعلان آن متغیر در درون بلوک کد.



مقادیر پیش‌فرض برای آرگومان‌ها

- آرگومان‌ها با مقادیر پیش‌فرض.

آرگومان با مقدار پیش‌فرض

```
def harmonic(n, r=1):
    total = 0.0
    for i in range(1, n + 1):
        total += 1.0 / (i ** r)
    return total
```

```
print(harmonic(4))
```

$$\frac{1}{1^1}+\frac{1}{2^1}+\frac{1}{3^1}+\frac{1}{4^1}$$

```
print(harmonic(4, r=2))
```

$$\frac{1}{1^2}+\frac{1}{2^2}+\frac{1}{3^2}+\frac{1}{4^2}$$

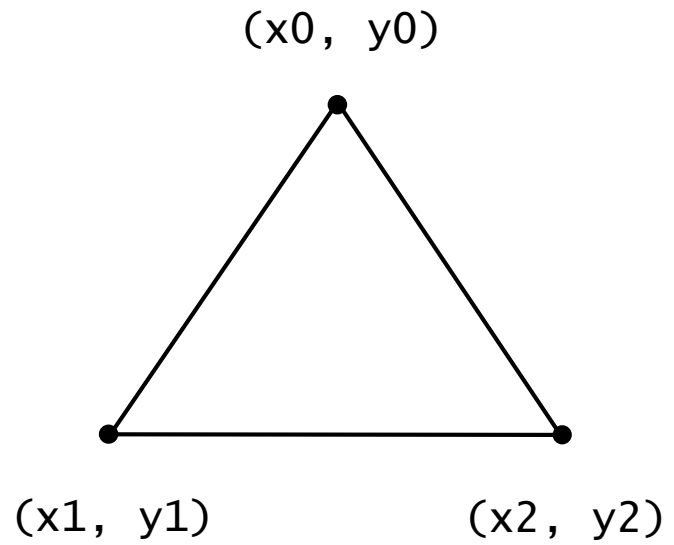
```
print(harmonic(4, r=3))
```

$$\frac{1}{1^3}+\frac{1}{2^3}+\frac{1}{3^3}+\frac{1}{4^3}$$

اثرات جانبی

- اثرات جانبی.

```
def draw_triangle(x0, y0, x1, y1, x2, y2):
    stddraw.line(x0, y0, x1, y1)
    stddraw.line(x1, y1, x2, y2)
    stddraw.line(x2, y2, x0, y0)
```



بررسی نوع و چندریختی

- چندریختی.

```
def f(a, b):
    return a + b
```

```
x = 2
y = 3
z = f(x, y)    # z is 5
```

```
x = 'Hello, '
y = 'world'
z = f(x, y)    # z is 'Hello, world'
```

توابع: چند مثال

یک تابع با چندین دستور برگشت

```
def is_prime(n):
    if n < 2: return False
    i = 2
    while i*i <= n:
        if n % i == 0: return False
        i += 1
    return True
```

بررسی اول بودن

یک تابع با چندین آرگومان ورودی

```
def hypot(a, b):
    return math.sqrt(a*a + b*b)
```

وتر یک مثلث قائم‌الزاویه

یک تابع با آرگومان دارای مقدار پیش‌فرض

```
def harmonic(n, r=1):
    total = 0.0
    for i in range(1, n + 1):
        total += 1.0 / (i ** r)
    return total
```

اعداد هارمونیک تعمیم یافته

یک تابع بدون دستور برگشت

```
def draw_triangle(x0, y0, x1, y1, x2, y2):
    stddraw.line(x0, y0, x1, y1)
    stddraw.line(x1, y1, x2, y2)
    stddraw.line(x2, y2, x0, y0)
```

ترسیم یک مثلث

توابع: چالش ۱-الف

- پرسش. پس از کامپایل و اجرای کد زیر چه اتفاقی می‌افتد؟

```
import sys

def cube(i):
    j = i * i * i
    return j

n = int(sys.argv[1])
for i in range(1, n + 1):
    print('%s %s' % (i, cube(i)))
```

```
% python cubes1.py 6
1 1
2 8
3 27
4 64
5 125
6 216
```

توابع: چالش ۱-ب

- پرسش. پس از کامپایل و اجرای کد زیر چه اتفاقی می‌افتد؟

```
import sys

def cube(i):
    i = i * i * i
    return i

n = int(sys.argv[1])
for i in range(1, n + 1):
    print('%s %s' % (i, cube(i)))
```

```
% python cubes2.py 6
1 1
2 8
3 27
4 64
5 125
6 216
```

توابع: چالش ۱-پ

- پرسش. پس از کامپایل و اجرای کد زیر چه اتفاقی می‌افتد؟

```
import sys

def cube(i):
    i = i * i * i



n = int(sys.argv[1])
for i in range(1, n + 1):
    print('%s %s' % (i, cube(i)))
```

```
% python cubes3.py 6
1 None
2 None
3 None
4 None
5 None
6 None
```

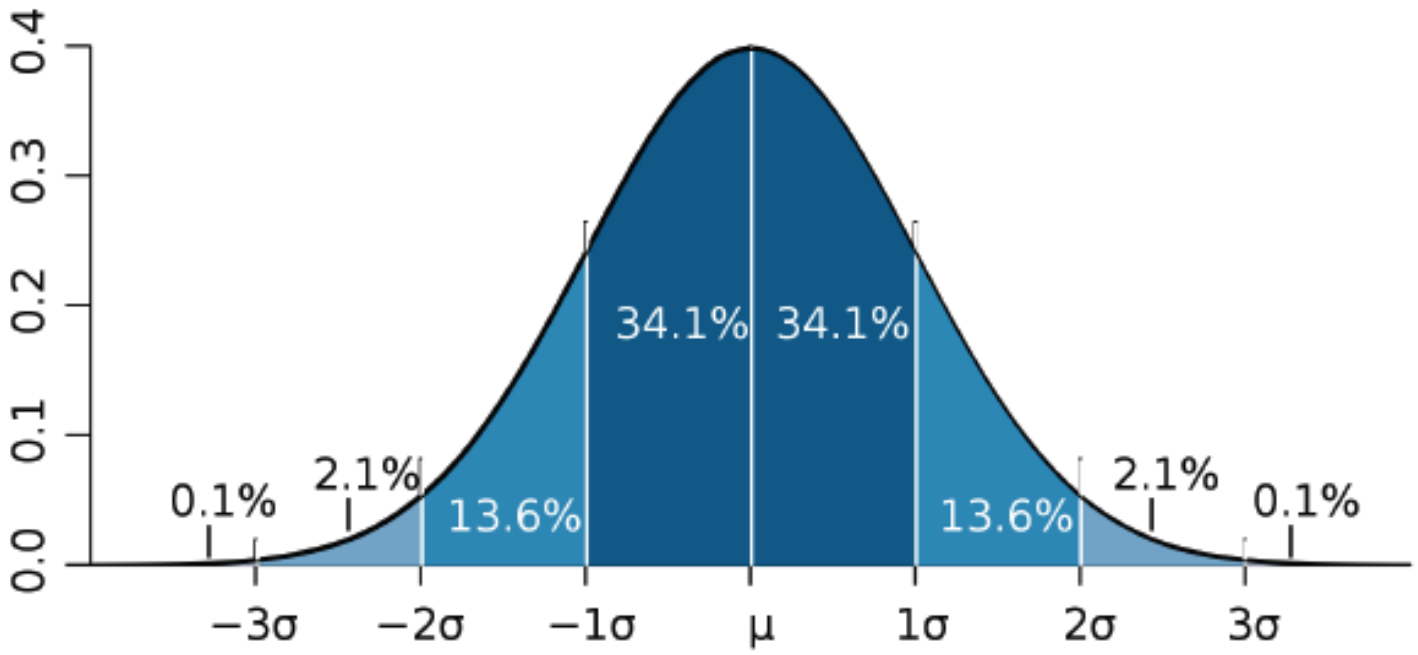
توابع: چالش ۱-ت

- پرسش. پس از کامپایل و اجرای کد زیر چه اتفاقی می‌افتد؟

```
import sys

def cube(i):
    return i * i * i



n = int(sys.argv[1])
for i in range(1, n + 1):
    print('%s %s' % (i, cube(i)))
```

```
% python cubes4.py 6
1 1
2 8
3 27
4 64
5 125
6 216
```

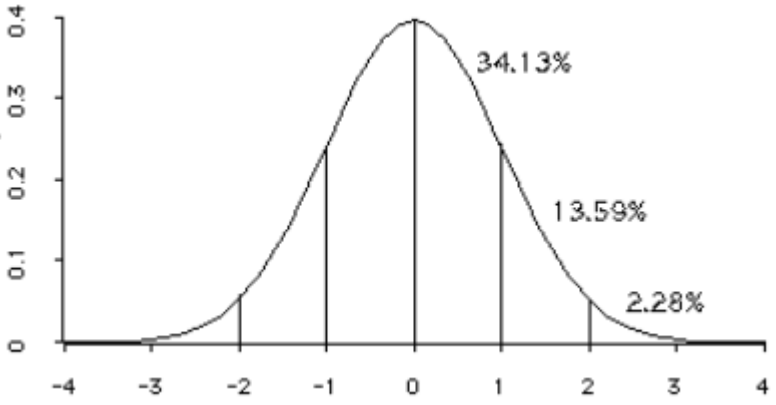
توزیع گاوسی



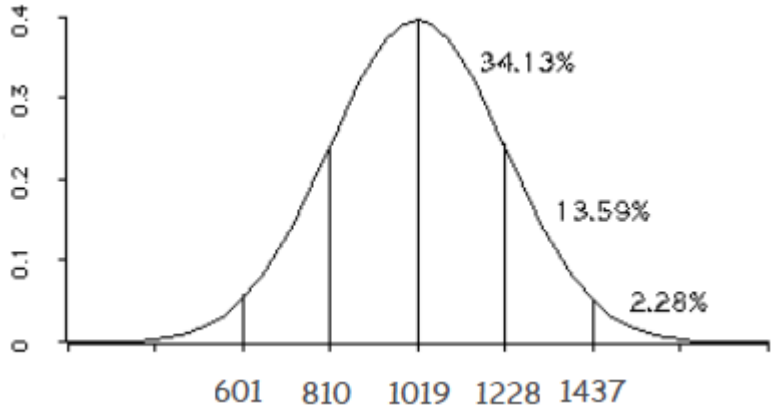
توزیع گاوسی

- توزیع گاوسی استاندارد.
  - «توزیع زنگوله شکل»
  - پایه اغلب تحلیل‌های آماری در علوم اجتماعی و فیزیکی.

- مثال. نمرات آزمون SAT در سال ۲۰۰۰ از یک توزیع گاوسی با میانگین $\mu = 1019$ و انحراف معیار $\sigma = 209$ پیروی می‌کند.



$$\phi(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2}$$



$$\phi(x, \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/2\sigma^2}$$

پیاده‌سازی در پایتون

- توابع ریاضی. هر زمان ممکن است از توابع پیش ساخته استفاده کنید. در غیر این صورت خودتان توابع مورد نیاز را پیاده‌سازی کنید.

```
def phi(x):
    return math.exp(-x * x / 2.0) / math.sqrt(2.0 * math.pi)

def pdf(x, mu=0.0, sigma=1.0):
    return phi((x - mu) / sigma) / sigma
```

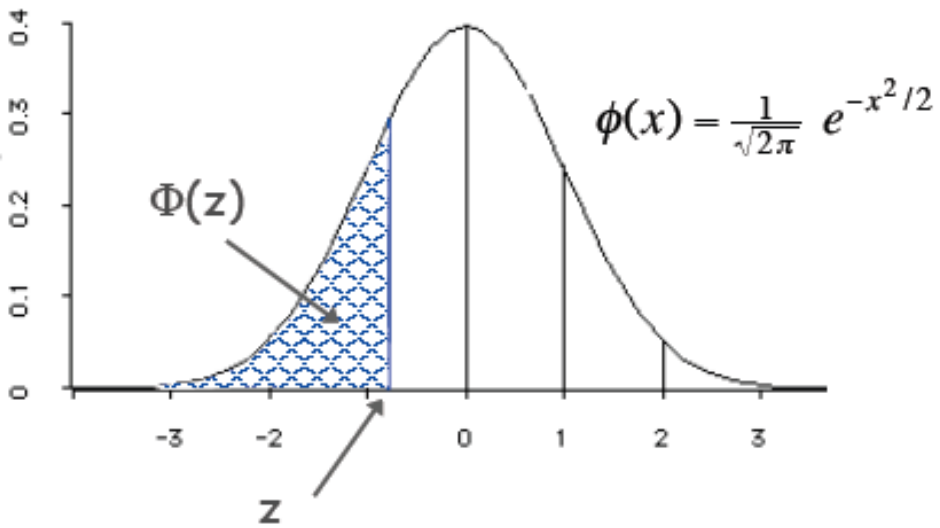
$$\phi(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2}$$

$$\phi(x, \mu, \sigma) = \phi\left(\frac{x-\mu}{\sigma}\right)/\sigma$$

- چندین آرگومان. یک تابع می‌تواند هر تعداد آرگومان ورودی داشته باشد.
- فراخوانی توابع دیگر. یک تابع می‌تواند توابع دیگر را فراخوانی کند.

تابع توزیع تجمعی گاوسی

- هدف. محاسبه تابع توزیع تجمعی گاوسی $\Phi(z)$.
- چالش. هیچ عبارت «شکل بسته» وجود ندارد!

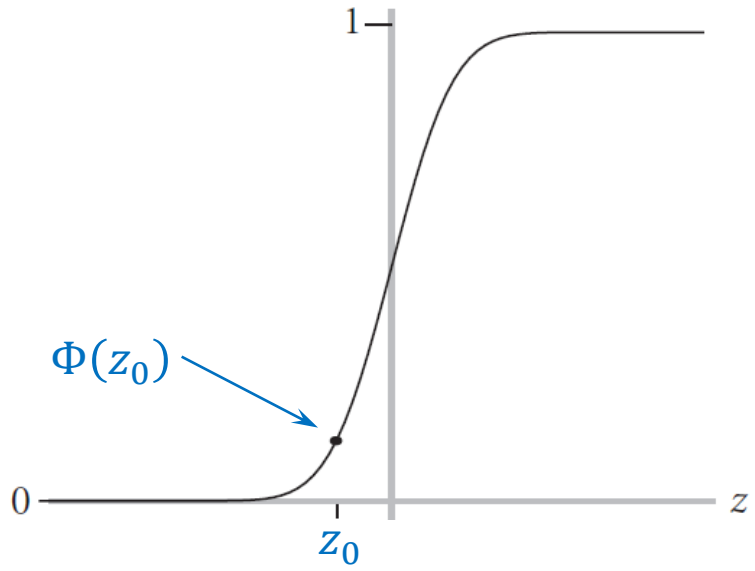


بسط تیلور

$$\Phi(z) = \int_{-\infty}^{z} \phi(x)dx$$
$$= \frac{1}{2} + \phi(z)\left(z + \frac{z^3}{3} + \frac{z^5}{3 \cdot 5} + \frac{z^7}{3 \cdot 5 \cdot 7} + \cdots\right)$$

پیاده‌سازی پایتون


```
def Phi(z):
    if z < -8.0: return 0.0
    if z >  8.0: return 1.0
    total = 0.0
    term = z
    i = 3
    while total != total + term:
        total += term
        term *= z * z / float(i)
        i += 2
    return 0.5 + phi(z) * total

def cdf(z, mu=0.0, sigma=1.0):
    return Phi((z - mu) / sigma)
```



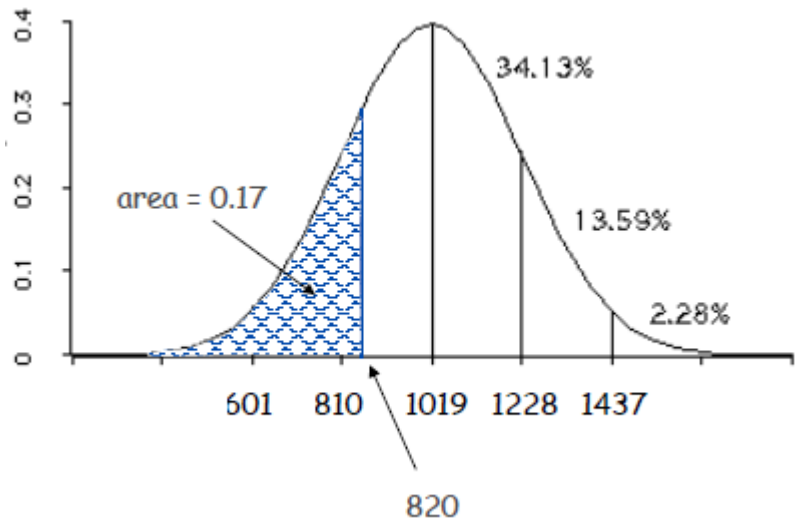
بسط تیلور

$$\Phi(z) = \int_{-\infty}^{z} \phi(x)dx = \frac{1}{2} + \phi(z)\left(z + \frac{z^3}{3} + \frac{z^5}{3\cdot 5} + \frac{z^7}{3\cdot 5\cdot 7} + \cdots\right)$$

نمرات آزمون SAT

- پرسش. چند درصد از شرکت کنندگان در آزمون SAT واجد شرایط نیستند. برای واجد شرایط بودن، نمره شرکت کننده باید حداقل برابر با ۸۲۰ باشد.

- پاسخ.



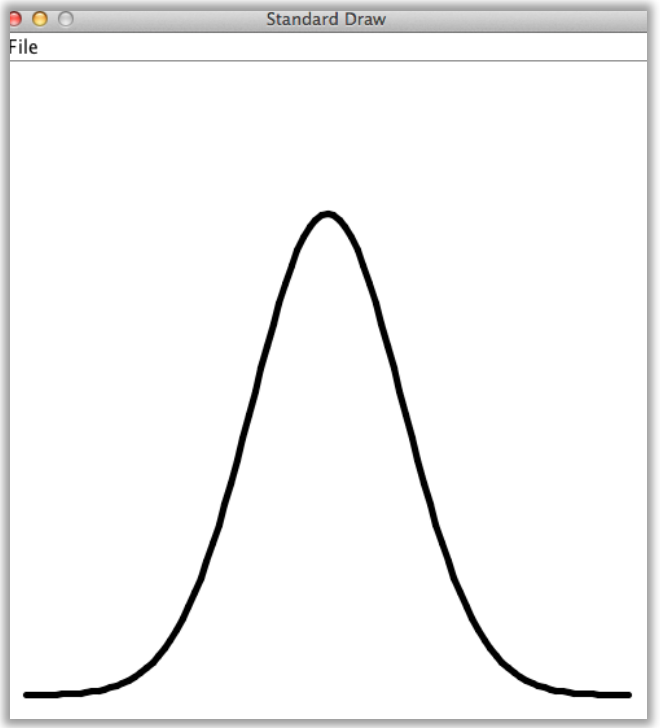
$$\Phi(820,1019,209) \approx 0.17051$$

```
fraction = cdf(820, 1019, 209)
```

ترسیم توزیع گوسی

- مثال. ترسیم $\phi(x,0,1)$ در بازه $(-4,4)$.

```
import sys, stddraw, gauss

def main():
    n = int(sys.argv[1])

    stddraw.setXscale(-4.0, 4.0)
    stddraw.setYscale(0.0, 0.5)
    stddraw.setPenRadius(0.01)

    x = [0.0] * (n + 1)
    y = [0.0] * (n + 1)

    for i in range(n + 1):
        x[i] = -4.0 + 8.0 * i / n
        y[i] = gaussian.pdf(x[i])

    for i in range(n):
        stddraw.line(x[i], y[i], x[i+1], y[i+1])

    stddraw.show()

if __name__ == '__main__':
    main()
```



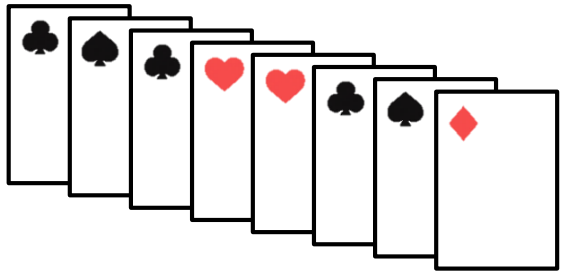

```
% python gaussian_plot.py 200
```

ایجاد توابع

- توابع به شما امکان می‌دهند یک سطح جدید از انتزاع ایجاد کنید:
  - فراتر از آن چیزی که کتابخانه‌های از پیش بسته‌بندی شده در اختیار شما قرار می‌دهند.
  - هر زمان بخواهید ابزار مورد نیازتان را ایجاد می‌کنید: `gaussian.Phi()` و ...
- فرآیند.
  - گام ۱: شناسایی یک ویژگی مفید
  - گام ۲: پیاده‌سازی
  - گام ۳: استفاده
  - گام ۳+: استفاده مجدد در هر برنامه‌ای که بخواهید.

مسئله جمع‌کننده کالابرگ‌ها

- مسئله جمع‌کننده کالابرگ‌ها. با داشتن n نوع کارت مختلف، برای این که از هر نوع کارت (حداقل) یکی داشته باشیم؛ چند کارت باید انتخاب شود؟



فرض می‌کنیم در هر انتخاب، احتمال انتخاب کارت‌ها با یکدیگر برابر است.

- الگوریتم شبیه‌سازی. به صورت تکراری هر بار یک عدد صحیح بین 0 و n - 1 انتخاب کن، تا زمانی که از هر کارت حداقل یکی داشته باشیم.

- پرسش. چگونه می‌توانیم بررسی کنیم که از هر نوع کارت حداقل یکی داریم؟

- پاسخ. با استفاده از یک آرایه بولی به گونه‌ای که is_collected[i] تنها وقتی درست است که از کارت نوع i حداقل یکی داشته باشیم.

is_collected

0	1	2	...	n - 1
F	F	F	...	F

مسئله جمع‌کننده کالابرگ‌ها: قبلاً


```
import sys
import random

n = int(sys.argv[2])

count = 0
collected_count = 0
is_collected = [False] * n

while collected_count < n:
    count += 1
    value = random.randrange(0, n)
    if not is_collected[value]:
        collected_count += 1
        is_collected[value] = True

print(count)
```

مسئله جمع‌کننده کالابرگ‌ها: با استفاده از توابع


```
import sys, random

def get_coupon(n):
    return random.randrange(0, n)

def collect(n):
    count, collected_count = 0, 0
    is_collected = [False] * n

    while collected_count < n:
        count += 1
        value = get_coupon(n)
        if not is_collected[value]:
            collected_count += 1
            is_collected[value] = True

    return count

n = int(sys.argv[1])
print(collect(n))
```

`is_collected[]`	*cards collected*
`count`	*number collected*
`collected_count`	*number that differ*
`value`	*current value*

```
% python coupon.py 1000
6552

% python coupon.py 1000
6481

% python coupon.py 1000000
12783771
```

آرایه‌ها به عنوان آرگومان

find the maximum of the arrays value

```
def mean(a):
    total = 0.0
    for v in a:
        total += v
    return total / len(a)
```

dot product

```
def dot(a, b):
    total = 0.0
    for i in range(len(a)):
        total += a[i] * b[i]
    return total
```

exchange two elements in the array

```
def exchange(a, i, j):
    temp = a[i]
    a[i] = a[j]
    a[j] = temp
```

shuffle the array

```
def shuffle(a):
    n = len(a)
    for i in range(n):
        r = random.randrange(i, n)
        exchange(a, i, r)
```