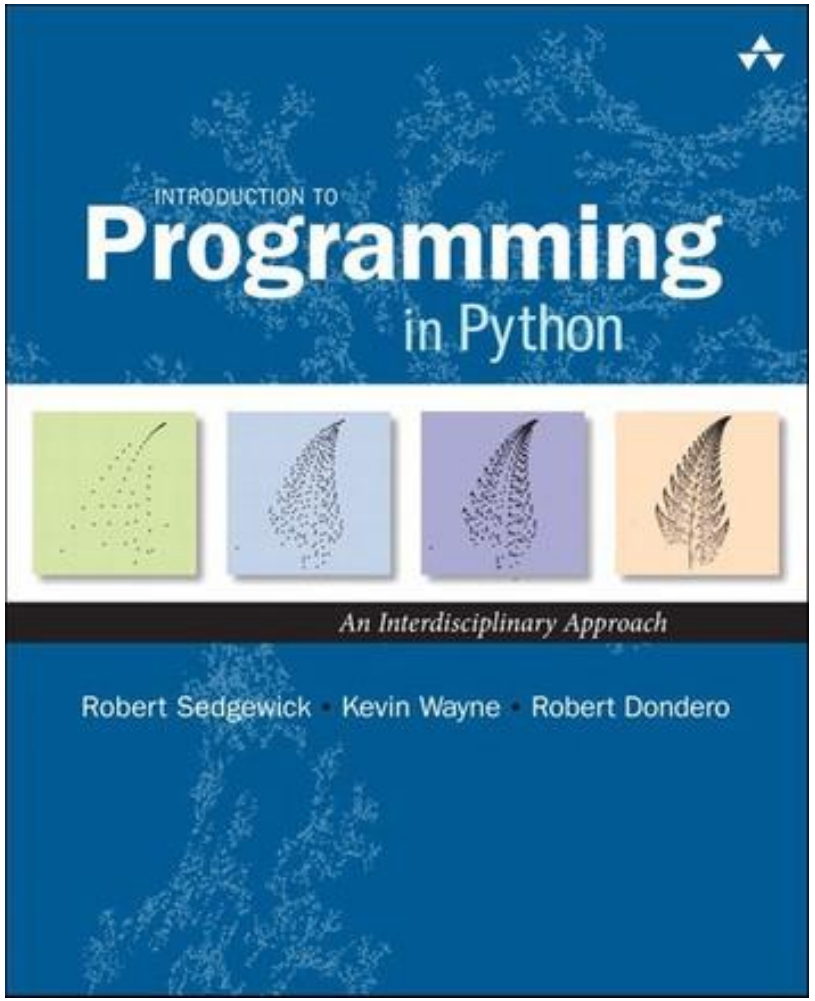
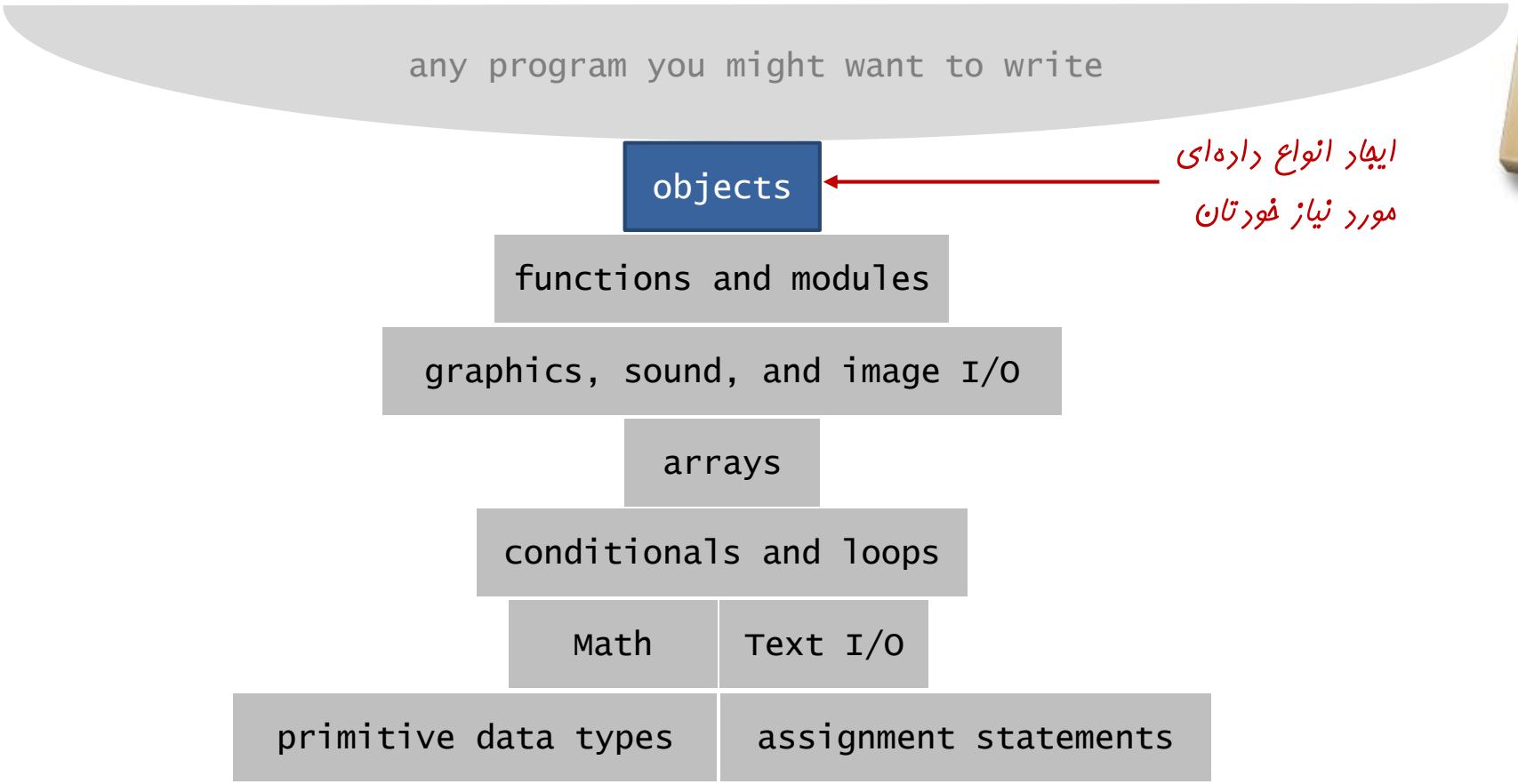
برنامه‌نویسی شی‌گرا:
طراحی انواع داده‌ای جدید

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

۳-۳ طراحی انواع داده‌ای



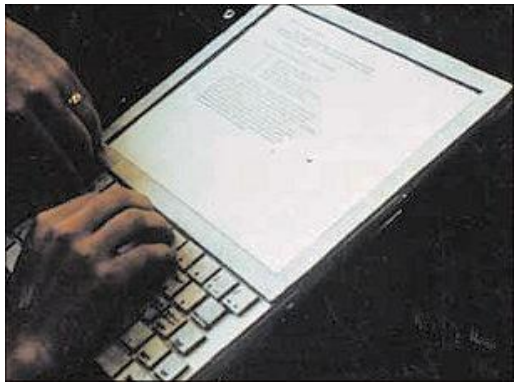
اجزای برنامه‌نویسی



برنامه‌نویسی شی‌گرا

- برنامه‌نویسی رویه‌ای. [فعل‌گرا]
  - به کامپیوتر بگو این کار را انجام دهد.
  - به کامپیوتر بگو آن کار را انجام دهد.
- فلسفه الن کِی. نرم‌افزار یک شبیه‌سازی از دنیای واقعی است.
  - ما (تقریباً) می‌دانیم دنیای واقعی چگونه کار می‌کند.
  - طراحی نرم‌افزار برای مدل‌سازی دنیای واقعی.
- برنامه‌نویسی شی‌گرا. [اسم‌گرا]
  - یک الگوی برنامه‌نویسی بر محور انواع داده‌ای.
  - مشخص کردن چیزهایی که بخشی از دامنه مسئله یا راه‌حل هستند.
  - اشیا در دنیا چیزهایی را می‌دانند: متغیرهای نمونه.
  - اشیا در دنیا کارهایی را انجام می‌دهند: متدها.

الن کِی



- الن کِی. [زیراکس، دهه ۷۰]
  - پدید آورنده زبان برنامه‌نویسی Smalltalk.
  - ایده‌ها: لپ‌تاپ، واسط‌های گرافیکی کاربر امروزی، برنامه‌نویسی شی‌گرا.

الن کِی
جایزه تورینگ ۲۰۰۳

« انقلاب کامپیوتری هنوز شروع نشده است. »

« بهترین روش برای پیش‌بینی آینده ابداع آن است. »

« اگر حداقل ۹۰ درصد از مواقع شکست نمی‌خوری، به اندازه کافی هدف‌دار نیستی. »

-- آلن کِی

پنهان‌سازی

- نوع داده‌ای. مجموعه‌ای از مقادیر و عملیات قابل انجام بر روی آن‌ها.
- مثال. رشته، عدد مختلط، بردار، ماتریس، ...

- نوع داده‌ای پنهان‌سازی شده. پنهان‌سازی بازنمایی درونی نوع داده‌ای.

- جداسازی پیاده‌سازی از مشخصات طراحی.
  - یک کلاس فراهم کننده بازنمایی داده‌ها و کدهایی برای عمل کردن بر روی آن داده‌ها است.
  - یک مشتری از نوع داده‌ای به صورت یک «جعبه سیاه» استفاده می‌کند.
  - یک API قرارداد میان مشتری و کلاس را مشخص می‌کند.

- سخن آخر. برای استفاده از یک نوع داده‌ای لازم نیست بدانید چگونه پیاده‌سازی شده است.

معنای شهودی

مشتری

مشتری باید براند چگونه
از واسط استفاده کند

واسط برنامه‌نویسی

- تنظیم صدا
- تغییر کانال
- تنظیم تصویر

مشتری و پیاده‌سازی باید از قبل بر روی واسط توافق کنند.

پیاده‌سازی

- لامپ پرتوی کاتدی
- تفنگ الکترونی
- وزن ۲۴۱ پوند

پیاده‌سازی باید براند چه واسطی را
پیاده‌سازی کند

معنای شهودی

مشتری

مشتری باید بداند چگونه از واسط استفاده کند

واسط برنامه‌نویسی

- تنظیم صدا
- تغییر کانال
- تنظیم تصویر

پیاده‌سازی

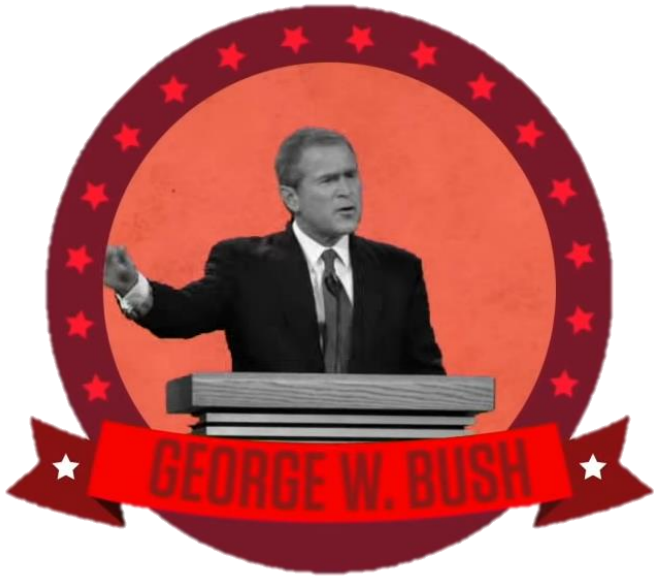
- صفحه نمایش پلاسما
- قابل نصب روی دیوار
- عمق ۴ اینچ

پیاده‌سازی باید بداند چه واسطی را پیاده‌سازی کند

باید بتوان یک پیاده‌سازی بهتر را بدون نیاز به تغییر مشتری جایگزین کرد.

نوع داده‌ای شمارنده

در انتخابات ریاست جمهوری سال ۲۰۰۰ ایالات متحده، تعداد آرای ال گور در یک ماشین الکترونیکی رأی‌گیری در ایالت فلوریدا در شهرستان ولوسیا برابر با ۱۶۰۲۲- بود.



https://www.youtube.com/watch?v=iZLWPleeCHE

نوع داده‌ای شمارنده: واسط برنامه‌نویسی

- شمارنده. یک نوع داده‌ای برای شمارش آرای الکترونیکی.

operation	*description*
`Counter(id, maxCount)`	*ایجاد یک شی شمارنده جدید با نام* `id`*، مقدار اولیه* ۰ *و حداکثر مقدار* `maxCount`
`c.increment()`	*افزایش مقدار شمارنده* `c` *به اندازه یک واحد، مگر این که مقدارش برابر با* `maxCount` *باشد*
`c.value()`	*برگرداندن مقدار شمارنده* `c`
`str(c)`	*نمایش رشته‌ای از شمارنده* `c` *به صورت* `'id: value'`

نوع داده‌ای شمارنده: پیاده‌سازی

- شمارنده. یک نوع داده‌ای برای شمارش آرای الکترونیکی.

```
class Counter:
    def __init__(self, id, maxCount):
        self._name = id
        self._maxCount = maxCount
        self._count = 0

    def increment(self):
        if self._count < self._maxCount:
            self._count += 1

    def value(self):
        return self._count

    def __eq__(self, other):
        return self._count == other._count

    def __ge__(self, other):
        return self._count >= other._count
```

بخواه ولی دست نزن!

- انواع داده‌ای پنهان‌سازی شده.
  - بدون دست زدن به داده‌ها هر کاری می‌خواهی انجام بده. [برنامه مشتری]
  - در عوض، از شی مربوطه بخواه داده‌هایش را دست‌کاری کند.

« بخواه ولی دست نزن. »

ادل گلدبرگ
رئیس پیشین ACM
از توسعه‌دهندگان Smalltalk

- درس. محدود کردن حوزه، نگهداری و درک برنامه‌ها را ساده‌تر می‌کند.

تغییرناپذیری

- داده‌های تغییرناپذیر. مقدار شی پس از ایجاد شدن دیگر نمی‌تواند تغییر کند.

mutable	*immutable*
Picture	Charge
Histogram	Color
Turtle	Stopwatch
StockAccount	Complex
Counter	String
Arrays	primitive types

تغییرناپذیری: مزایا و معایب

- داده‌های تغییرناپذیر. مقدار شی پس از ایجاد شدن دیگر نمی‌تواند تغییر کند.

```
def mandel(z0, limit=255):
    z = z0
    for t in range(limit):
        if abs(z) > 2.0: return t
        z = z * z + z0
    return limit
```

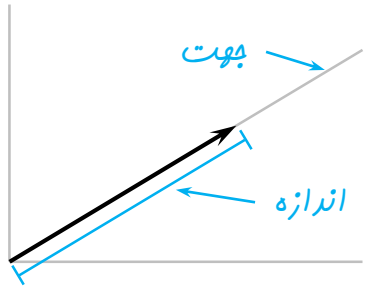
- مزایا.
  - اجتناب از خطاهای مربوط به نام مستعار.
  - ساده‌تر کردن اشکال‌زدایی از برنامه.
  - محدود کردن حوزه کد که می‌تواند مقادیر را تغییر دهد.
  - ارسال اشیا به متدها بدون نگرانی در مورد تغییر کردن آنها.

- ایراد. به ازای هر مقدار، باید یک شی جدید ایجاد شود.

نوع داده‌ای بردار: واسط برنامه‌نویسی

- مجموعه مقادیر. دنباله‌ای از اعداد حقیقی. [مختصات کارتزی]

operation	special method	description
`Vector(a)`	`__init__(self, a)`	یک بردار جدید با مختصات‌های کارتزی برگرفته از آرایه `a`
`x[i]`	`__getitem__(self, i)`	مختصات کارتزی `i` ام از آرایه `x`
`x + y`	`__add__(self, other)`	مجموع دو بردار `x` و `y`
`x - y`	`__sub__(self, other)`	تفاضل دو بردار `x` و `y`
`x.dot(y)`		ضرب نقطه‌ای دو بردار `x` و `y`
`x.scale(alpha)`		ضرب مقدار اعشاری `alpha` و بردار `x`
`x.direction()`		بردار واحد هم جهت با بردار `x`
`abs(x)`	`__abs__(self)`	اندازه (طول) بردار `x`
`len(x)`	`__len__(self)`	تعداد مؤلفه‌های بردار `x`
`str(a)`	`__str__(self)`	بازنمایی رشته‌ای از بردار `x`



$x = (0,\ 3,\ 4,\ 0)$
$y = (0,\ -3,\ 1,\ -4)$

نوع داده‌ای بردار: کاربردها

- اهمیت. یک مدل ریاضی بسیار پرکاربرد.

- کاربردها.
  - آمار.
  - جبر خطی.
  - خوشه‌بندی و جستجوی شباهت.
  - نمایش نیرو، سرعت، شتاب، اندازه حرکت، گشتاور.
  - ...

نوع داده‌ای بردار: برنامه مشتری آزمایشی


```
def main():

    xCoords = [1.0, 2.0, 3.0, 4.0]
    yCoords = [5.0, 2.0, 4.0, 1.0]

    x = Vector(xCoords)
    y = Vector(yCoords)

    print('x       = ' + str(x))
    print('y       = ' + str(y))
    print('x + y   = ' + str(x + y))
    print('10x     = ' + str(x.scale(10.0)))
    print('|x|     = ' + str(abs(x)))
    print('<x, y>  = ' + str(x.dot(y)))
    print('|x - y| = ' + str(abs(x - y)))

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

```
% python vector.py
x       = [1.0, 2.0, 3.0, 4.0]
y       = [5.0, 2.0, 4.0, 1.0]
x + y   = [6.0, 4.0, 7.0, 5.0]
10x     = [10.0, 20.0, 30.0, 40.0]
|x|     = 5.477225575051661
<x, y>  = 25.0
|x - y| = 5.099019513592784
```

نوع داده‌ای بردار: پیاده‌سازی با آرایه


```
class Vector:

    def __init__(self, a):
        self._coords = a[:]
        self._n = len(a)

    def __getitem__(self, i):
        return self._coords[i]

    def __add__(self, other):
        result = [0] * self._n
        for i in range(self._n):
            result[i] = self._coords[i] + other._coords[i]
        return Vector(result)

    def __sub__(self, other):
        result = [0] * self._n
        for i in range(self._n):
            result[i] = self._coords[i] - other._coords[i]
        return Vector(result)
```

ایجاد یک نسخه دفاعی به منظور تضمین تغییرناپذیری

نوع داده‌ای بردار: پیاده‌سازی با آرایه


```
def scale(self, alpha):
result = [0] * self._n
    for i in range(self._n):
        result[i] = alpha * self._coords[i]
    return Vector(result)
```

```
def __dot__(self, other):
    result = 0
    for i in range(self._n):
        result += self._coords[i] * other._coords[i]
    return result
```

```
def direction(self):
    return self.scale(1.0 / abs(self))
```

```
def __str__(self):
    return str(self._coords)
```

```
def __len__(self):
    return self._n
```

نوع داده‌ای بردار: پیاده‌سازی با تاپل


```
class Vector:

    def __init__(self, a):
        self._coords = tuple(a)
        self._n = len(a)

    def __getitem__(self, i):
        return self._coords[i]

    def __add__(self, other):
        result = [0] * self._n
        for i in range(self._n):
            result[i] = self._coords[i] + other._coords[i]
        return Vector(result)

    def __sub__(self, other):
        result = [0] * self._n
        for i in range(self._n):
            result[i] = self._coords[i] - other._coords[i]
        return Vector(result)
```

بیش‌بارگذاری عملگرها: عملگرهای ریاضی

operation	*special method*	*description*
`x + y`	`__add__(self, other)`	مجموع x و y
`x - y`	`__sub__(self, other)`	تفاضل x و y
`x * y`	`__mul__(self, other)`	حاصل‌ضرب x و y
`x ** y`	`__pow__(self, other)`	x به توان y
`x / y`	`__truediv__(self, other)`	خارج‌قسمت x و y
`x // y`	`__floordiv__(self, other)`	کف خارج‌قسمت x و y
`x % y`	`__mod__(self, other)`	باقیمانده تقسیم x بر y
`+x`	`__pos__(self)`	x
`-x`	`__neg__(self)`	منفی x

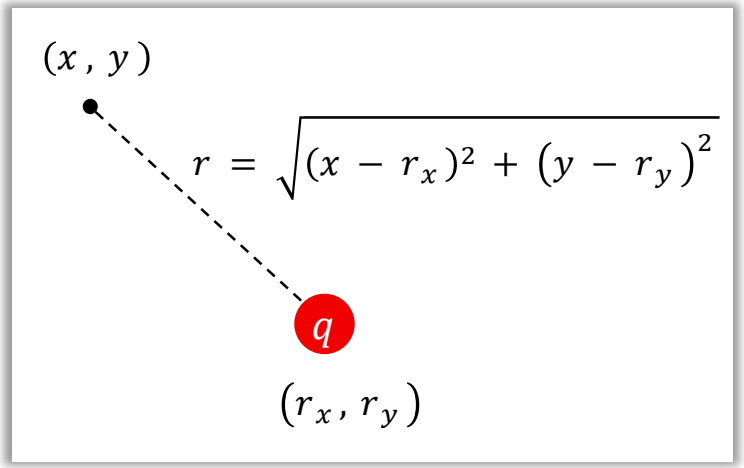
بیش‌بارگذاری عملگرها: تساوی

operation	*special method*	*description*
`x == y`	`__eq__(self, other)`	بررسی تساوی x و y
`x != y`	`__ne__(self, other)`	عدم تساوی x و y

- مثال. بررسی تساوی دو ذره بادار الکتریکی

```
class Charge:
    ...
    def __eq__(self, other):
        if self._rx != other._rx: return False
        if self._ry != other._ry: return False
        if self._q  != other._q : return False
        return True

    def __ne__(self, other):
        return not self.__eq__(other)
```



بیش‌بارگذاری عملگرها: درهم‌سازی

- درهم سازی. یک نگاشت از شی مورد نظر به یک عدد صحیح [کد درهم‌سازی].
  - یک عمل پایه‌ای در ارتباط با بررسی تساوی.

- می‌گوییم یک شی قابلیت درهم‌سازی دارد اگر سه ویژگی زیر را برآورده سازد:
  - این شی بتواند از طریق عملگر == با یک شی دیگر به منظور بررسی مساوی بودن مقایسه شود.
  - هرگاه دو شی مساوی باشند، کد درهم‌سازی یکسانی داشته باشند.
  - کد درهم‌سازی شی نتواند در طول زمان حیاتش تغییر کند.

- به منظور پیاده‌سازی قابلیت درهم‌سازی برای اشیای خودتان، متدهای `__eq__()` و `__hash__()` را پیاده‌سازی کنید.

```
class Charge:
    ...
    def __hash__(self):
        a = (self._rx, self._ry, self._q)
        return hash(a)
```

بیش‌بارگذاری عملگرها: عملگرهای مقایسه‌ای

operation	*special method*	*description*
`x < y`	`__lt__(self, other)`	آیا x کوچک‌تر از y است؟
`x <= y`	`__le__(self, other)`	آیا x کوچک‌تر یا مساوی y است؟
`x > y`	`__gt__(self, other)`	آیا x بزرگ‌تر از y است؟
`x >= y`	`__ge__(self, other)`	آیا x بزرگ‌تر یا مساوی y است؟

- مثال. شمارنده.

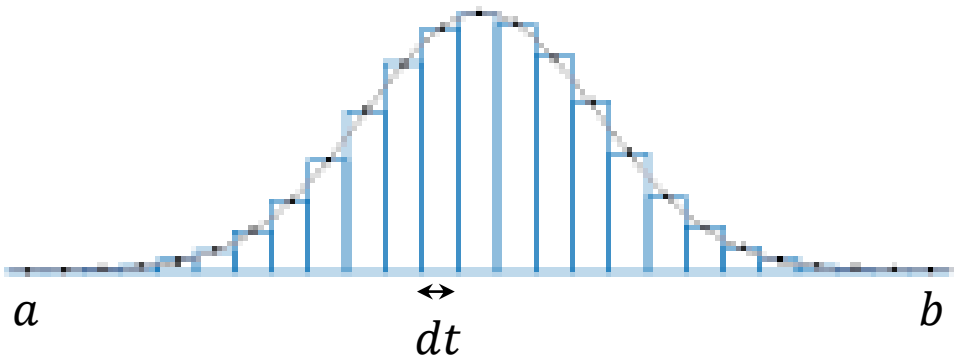
```
class Counter:
   ...
    def __lt__(self, other): return self._count <  other._count
    def __le__(self, other): return self._count <= other._count
    def __eq__(self, other): return self._count == other._count
    def __ne__(self, other): return self._count != other._count
    def __gt__(self, other): return self._count >  other._count
    def __ge__(self, other): return self._count >= other._count
```

توابع به عنوان اشیا

- در پایتون، همه چیز از جمله توابع، شی هستند.
- در نتیجه، می‌توان توابع را به عنوان آرگومان به توابع دیگر ارسال نمود و یا توابعی پیاده‌سازی کرد که خروجی آنها تابع باشد.

```
def square(x):
    return x * x
```

```
def integrate(f, a, b, n=1000):
    total = 0.0
    dt = 1.0 * (b - a) / n
    for i in range(n):
        total += dt * f(a + (i + 0.5) * dt)
    return total
```



وراثت

- وراثت. روشی به منظور استفاده مجدد از کد با ایجاد زیرکلاس از یک کلاس موجود.
- ایده. ایجاد یک کلاس جدید (زیرکلاس) که داده‌ها و متدهایش را از یک کلاس دیگر (سوپرکلاس) به ارث می‌برد.
  - معمولا زیرکلاس دارای متدهای بیشتری نسبت به سوپرکلاس خود است.

- وراثت برخلاف استفاده گسترده، دارای تأثیرات منفی بر روی برنامه‌نویسی ماجولار است:
  - هر تغییر در سوپرکلاس، بر روی تمام زیرکلاس‌ها تأثیر می‌گذارد.
  - زیرکلاس به داده‌های سوپرکلاس خود دسترسی دارد!

وراثت


```
class Counter:
    def __init__(self, id, maxCount):
        self._name = id
        self._maxCount = maxCount
        self._count = 0

    def increment(self):
        if self._count < self._maxCount:
            self._count += 1

    def value(self):
        return self._count

    def __eq__(self, other):
        return self._count == other._count

    def __ge__(self, other):
        return self._count >= other._count
```

```
from counter import Counter

class MyCounter(Counter):
    def __init__(self, id, maxCount):
        super().__init__(id, maxCount)

    def increment(self):
        self._count = (self._count + 1) %
                      self._maxCount

def main():
    c = MyCounter('counter1', 5)
    for i in range(13):
        print(c.value(), end=' ')
        c.increment()

if __name__ == '__
    main()
```

```
% python mycounter.py
0 1 2 3 4 0 1 2 3 4 0 1 2
```

استثناها و جملات ادعایی

- استثنا. یک رویداد که معمولا نشان دهنده بروز یک خطا در یک برنامه در حال اجرا است.

```
def __add__(self, other):
    if len(self) != len(other):
        raise Exception('vectors have different dimensions')
    result = [0] * self._n
    for i in range(self._n):
        result[i] = self._coords[i] + other._coords[i]
    return Vector(result)
```

- جمله ادعایی. یک عبارت بولی به منظور تشخیص خطاها و حصول اطمینان از درستی برنامه‌ها.

```
def increment(self):
    if self._count < self._maxCount:
        self._count += 1
    assert self._count >= 0, 'Negative count detected'
```