

3 Objektorientiertes Programmieren – erste Schritte

Wir haben jetzt gesehen, wie man Programme durch Funktionen und Module strukturieren kann. Beim Programmieren ist es wichtig, herauszufinden, welche Programmteile Einheiten bilden, die man als Funktion/Modul zusammenfassen und mit einem Namen versehen kann.

Im Beispiel mit dem Kugelnrollen haben wir auch zusammengehörige *Daten* „strukturiert“:
jede Kugel wurde durch ein Tupel aus Position und Richtung beschrieben.

Beim objektorientierten Programmieren kann man Daten, die zusammengehören, als Datentyp zusammenfassen.

Zum Datentyp gehören auch Operationen zum Manipulieren dieser Daten.

Den Datentyp nennt man dann *Klasse* und die Operationen dazu *Methoden*.

In dieser Vorlesung wollen wir (nochmal) ansehen, wie die Benutzung von Datentypen/Klassen geht.

Im nächsten Semester geht es auch darum, wie man eigene Datentypen implementiert.

3 Objekt-orientierte Programmierung – erste Schritte

1. Elemente des Programmierens

2. Funktionen und Module

3. Objekt-orientierte Programmierung (erste Schritte)

3.1 Benutzung von Daten-Typen

3.2 Dictionaries

3. Objekt-orientierte Programmierung (erste Schritte)

3.1 Benutzung von Daten-Typen

- Datentyp für Farben: `Color`

- Wetterdaten als buntes Streifendiagramm darstellen

- Datentyp für Bilder: `Picture`

3.2 Dictionaries

3.1 Benutzung von Datentypen

Datentypen/Klassen dienen der Speicherung bestimmter Daten

- ▶ `int`: ganze Zahlen
- ▶ `str`: Folge von Zeichen
- ▶ `Array list`: Folge beliebiger Daten
- ▶ Dateien (`_io.TextIOWrapper`): Daten auf der Festplatte
- ▶ Leinwand (`std::draw`): Platz zum Bemalen und zum Anzeigen auf dem Bildschirm

und der Bereitstellung von Funktionen/Methoden zu deren Manipulation. Beispiele:

- ▶ `int`: Addition `+`, Multiplikation `*`
- ▶ `str`: Konkatenation `+`, indizierter Zugriff `[i]`
- ▶ `Array list`: Erzeugen, Durchlaufen, Sortieren
- ▶ Dateien (`_io.TextIOWrapper`): eine Zeile lesen, eine Zeile schreiben
- ▶ `std::draw`: male einen Strich auf die Leinwand, stelle die Farbe des Stiftes ein

In dieser Vorlesung wollen wir nochmal ansehen, wie die Benutzung von Datentypen geht.
Im nächsten Semester geht es auch darum, wie man eigene Datentypen implementiert.

Programmierauftrag: erstelle eine Graphik für eine Titelseite



Auf der Titelseite der FAZ vom 26.6.2019 stand eine Graphik über die Entwicklung der Jahresdurchschnittstemperatur an einem nicht genannten Ort seit 1850. Für jedes Jahr steht von links nach rechts ein vertikaler Balken.

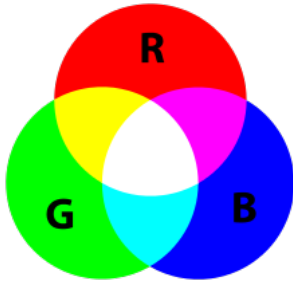
Ein weißer Balken steht dafür, dass die Durchschnittstemperatur des Jahres (etwa) das Mittel der Temperaturen seit 1850 ist. Je weiter die Jahresdurchschnittstemperatur unter dem Mittelwert liegt, desto dunkelblauer ist der Balken, und je weiter sie über dem Mittelwert liegt, desto dunkelroter ist er.

Wir wollen eine solche Graphik für Jena (oder andere Orte) malen.

1 Die Klasse Color – ein Datentyp für Farbe

Eine Vorstellung von Farbe

Jede Farbe setzt sich aus rot, grün und blau in unterschiedlichen Intensitäten zusammen.
Bei maximaler Intensität aller Farben zusammen erhalten wir weiß,
bei minimaler Intensität schwarz.



Farben werden durch Tripel (r, g, b) angegeben,
bei denen r die Intensität von rot,
 g die Intensität von grün und
 b die Intensität von blau angibt.

Jede Intensität ist eine Zahl aus dem Bereich $0 \dots 1$.

Das API der Klasse Color in color.py

Die Implementierung der Farbvorstellung

Operation	Bedeutung
<code>Color(r,g,b)</code>	eine neue Farbe mit Rot-, Grün- und Blau-Intensitäten r , g und b ; r , g und b sind <code>int</code> -Werte aus dem Bereich $0 \dots 255$
<code>c.getRed()</code>	die Rot-Intensität der Farbe c
<code>c.getGreen()</code>	die Grün-Intensität der Farbe c
<code>c.getBlue()</code>	die Blau-Intensität der Farbe c
<code>str(c)</code>	Darstellung der Farbe c als String (R, G, B)

`Color()` ist der *Konstruktor* und erzeugt ein neues Objekt der Klasse `Color`.

`getRed()`, `getGreen()` und `getBlue()` sind *Methoden* der Klasse `Color`.
Sie werden auf Objekte vom Typ `Color` angewendet.

`str()` gibt eine Text-Darstellung des Objektes zurück.

(Es gibt keine Methoden zum Ändern eines `Color`-Objektes.)

Das API der Klasse Color in color.py

Die Implementierung der Farbvorstellung

Operation	Bedeutung
<code>Color(r,g,b)</code>	eine neue Farbe mit Rot-, Grün- und Blau-Intensitäten r , g und b ; r , g und b sind <code>int</code> -Werte aus dem Bereich $0 \dots 255$
<code>c.getRed()</code>	die Rot-Intensität der Farbe c
<code>c.getGreen()</code>	die Grün-Intensität der Farbe c
<code>c.getBlue()</code>	die Blau-Intensität der Farbe c
<code>str(c)</code>	Darstellung der Farbe c als String (R, G, B)

`Color()` ist der *Konstruktor* und erzeugt ein neues Objekt der Klasse `Color`.

`getRed()`, `getGreen()` und `getBlue()` sind *Methoden* der Klasse `Color`.
Sie werden auf Objekte vom Typ `Color` angewendet.

`str()` gibt eine Text-Darstellung des Objektes zurück.

(Es gibt keine Methoden zum Ändern eines `Color`-Objektes.)

Das API des Moduls stddraw im Buch

Farb-Objekte der Klasse Color können als Farben bei stddraw verwendet werden.

APIs		761
function call	description	
<i>basic functions for drawing</i>		
stddraw.line(x0, y0, x1, y1)	draw a line from (x0, y0) to (x1, y1)	
stddraw.point(x, y)	draw a point at (x, y)	
stddraw.show()	show the drawing in the standard drawing window (and wait until it is closed by the user)	
<i>control functions for setting drawing parameters</i>		
stddraw.setCanvasSize(w, h)	set the size of the canvas to w-by-h pixels (w and h default to 512)	
stddraw.setXscale(x0, x1)	set the x-range of the canvas to (x0, x1) (x0 defaults to 0 and y0 defaults to 1)	
stddraw.setYscale(y0, y1)	set the y-range of the canvas to (y0, y1) (y0 defaults to 0 and y1 defaults to 1)	
stddraw.setPenRadius(r)	set the pen radius to r (defaults to 0.005)	
Note: If the pen radius is 0, then points and line widths will be the minimum possible size.		
<i>functions for drawing shapes</i>		
stddraw.circle(x, y, r)	draw a circle of radius r centered at (x, y)	
stddraw.square(x, y, r)	draw a 2r-by-2r square centered at (x, y)	
stddraw.rectangle(x, y, w, h)	draw a w-by-h rectangle with lower-left endpoint (x, y)	
stddraw.polygon(x, y)	draw a polygon that connects (x[i], y[i])	
Note: FilledCircle(), FilledSquare(), FilledRectangle(), and FilledPolygon() correspond to these and draw filled shapes, not just outlines.		
<i>functions for drawing shapes</i>		
stddraw.text(x, y, s)	draw string s, centered at (x, y)	
stddraw.setPenColor(color)	set the pen color to color (defaults to stddraw.BLACK)	
stddraw.setFontFamily(font)	set the font family to font (defaults to 'Helvetica')	
stddraw.setFontSize(size)	set the font size to size (defaults to 12)	
<i>functions for animation</i>		
stddraw.clear(color)	clear the background canvas (color every pixel color)	
stddraw.show(t)	show the drawing in the standard drawing window and wait for t milliseconds	

Programmier-Auftrag: male den Farbverlauf von Rot

Es sollen alle Rot-Töne vom dunkelsten bis zum hellsten Rot gemalt werden.

- ▶ `Color(0, 0, 0)` ist schwarz und `Color(255, 255, 255)` ist weiß.
- ▶ Ein „richtiges“ Rot ist `Color(255, 0, 0)`.
- ▶ Eine Farbe `Color(i, 0, 0)` mit $0 \leq i < 255$ ist ein „dunkleres“ Rot:
je kleiner i ist, desto dunkler ist das Rot.
- ▶ Eine Farbe `Color(255, i, i)` mit $0 < i \leq 255$ ist ein „helleres“ Rot:
je größer i ist, desto heller ist das Rot.



Insgesamt sind das $1 + 255 + 255 = 511$ Rot-Töne.

Unser Plan

- ▶ Wir schreiben eine Funktion `rot(i)`, die für $i=0..510$ einen der 511 Rot-Töne zurückgibt. Je größer i ist, umso heller ist das Rot.



- ▶ Wir schreiben eine Funktion `maleRotverlauf()`, die den Verlauf der Rot-Töne vom dunkelsten zum hellsten Rot als aufeinanderfolgende Balken auf die Leinwand von `std::draw` malt.
- ▶ Beide Funktionen zusammen bilden ein Modul.
`maleRotverlauf()` kann als Testfunktion des Moduls benutzt werden.

Unser Plan

- ▶ Wir schreiben eine Funktion `rot(i)`, die für $i=0..510$ einen der 511 Rot-Töne zurückgibt. Je größer i ist, umso heller ist das Rot.

Funktion	Bedeutung
----------	-----------

<code>rot(i)</code>	gibt einen Rot-Ton als <code>Color</code> -Objekt zurück (i ist ein <code>int</code>). Für $i=0..255$ wird <code>Color(i,0,0)</code> zurückgegeben, und für $i=256..510$ wird <code>Color(255,i-255,i-255)</code> zurückgegeben. Also ist <code>rot(0)</code> das dunkelste Rot, und <code>rot(510)</code> ist das hellste Rot.
---------------------	--

- ▶ Wir schreiben eine Funktion `maleRotverlauf()`, die den Verlauf der Rot-Töne vom dunkelsten zum hellsten Rot als aufeinanderfolgende Balken auf die Leinwand von `std::draw` malt.
- ▶ Beide Funktionen zusammen bilden ein Modul.
`maleRotverlauf()` kann als Testfunktion des Moduls benutzt werden.

Unser Plan

- ▶ Wir schreiben eine Funktion `rot(i)`, die für $i=0..510$ einen der 511 Rot-Töne zurückgibt. Je größer i ist, umso heller ist das Rot.

Funktion	Bedeutung
<code>rot(i)</code>	gibt einen Rot-Ton als <code>Color</code> -Objekt zurück (i ist ein <code>int</code>). Für $i=0..255$ wird <code>Color(i,0,0)</code> zurückgegeben, und für $i=256..510$ wird <code>Color(255,i-255,i-255)</code> zurückgegeben. Also ist <code>rot(0)</code> das dunkelste Rot, und <code>rot(510)</code> ist das hellste Rot.

- ▶ Wir schreiben eine Funktion `maleRotverlauf()`, die den Verlauf der Rot-Töne vom dunkelsten zum hellsten Rot als aufeinanderfolgende Balken auf die Leinwand von `std::draw` malt.

Funktion	Bedeutung
<code>maleRotverlauf()</code>	zeigt ein Bild mit dem Verlauf der Rot-Töne vom dunkelsten zum hellsten Rot.

- ▶ Beide Funktionen zusammen bilden ein Modul.
`maleRotverlauf()` kann als Testfunktion des Moduls benutzt werden.

```

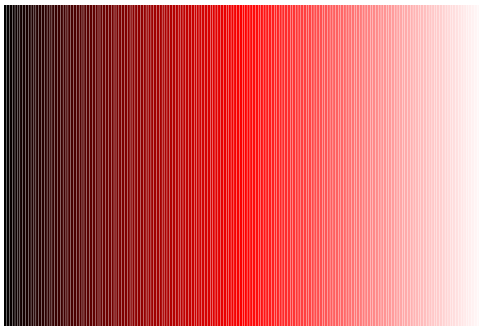
# rot_farbverlauf.py
#-----
# Malt den Farbverlauf von rot in Streifen auf die Leinwand von stddraw.
#   Color(0,0,0), Color(1,0,0),...,Color(255,0,0),Color(255,1,1),...,Color(255,255,255)
# Rot-Ton:  0           1     ...       255           256     ...       510
#-----
import stddraw, color
#-----
# rot(n) gibt ein Color-Objekt mit Rot-Ton n gemäß der obigen Darstellung zurück.
def rot(n):
    if n<=255:    c = color.Color( n, 0, 0 )
    else:        c = color.Color( 255, n-255, n-255)
    return c
#-----
# maleRotverlauf() richtet eine Leinwand mit x-Skala 0...511 ein
# und malt für jedes i in 0...510 einen Streifen mit Farbe rot(i) an x-Position i.
#-----
def maleRotverlauf():
    stddraw.setCanvasSize(1200,800)
    stddraw.setXscale(0,511)

    for i in range(0,511):
        c = rot(i)
        stddraw.setPenColor(c)
        stddraw.filledRectangle(i, 0, 1.4, 1)

    stddraw.show()
#-----
if __name__=='__main__': maleRotverlauf()

```

```
python3 rot_farbverlauf.py
```



Hier wurden die Streifen mit Breite 1 gemalt:
`stddraw.filledRectangle(i, 0, 1, 1)`
Durch Rundungsfehler von `stddraw` (das Bild ist 1200 Pixel breit und hat 512 ganzzahlige x-Positionen) werden dabei nicht alle Pixel der Leinwand bemalt und es entstehen dünne weiße Streifen, die man nicht haben will.

```
python3 rot_farbverlauf.py
```



Hier wurden die Streifen mit Breite 1.4 gemalt:
`stddraw.filledRectangle(i, 0, 1.4, 1)`
Dadurch werden alle Rundungsfehler übermalt ...

Verallgemeinerung: Farbverläufe für alle drei Grundfarben

Funktionen, die eine der drei Grundfarben rot, grün und blau im angegebenen Ton zurückgeben, können wir genauso implementieren. Das machen wir im Modul `farbverlaeufe.py`.

```
# farbverlaeufe.py
#-----
# Stellt Funktionen für Farbverläufe von rot, blau und grün zur Verfügung.
# Für rot(i) ist der Verlauf      rot(0)      rot(1)      ...      rot(255)      rot(256)      ...      rot(510)
#                               Color(0,0,0), Color(1,0,0),...,Color(255,0,0),Color(255,1,1),...,Color(255,255,255)
# Für gruen(i) ist der Verlauf gruen(0)      gruen(1)      ...      gruen(255)      gruen(256)      ...      gruen(510)
#                               Color(0,0,0), Color(0,1,0),...,Color(0,255,0),Color(1,255,1),...,Color(255,255,255)
# Für blau(i) ist der Verlauf  blau(0)      blau(1)      ...      blau(255)      blau(256)      ...      blau(510)
#                               Color(0,0,0), Color(0,0,1),...,Color(0,0,255),Color(1,1,255),...,Color(255,255,255)
#-----
import stddraw, color
#-----
# rot(n) gibt ein Color-Objekt mit Rot-Ton n gemäß der obigen Darstellung zurück.
def rot(n):      return color.Color( min(n,255), max(0,n-255), max(0,n-255) )
#-----
# gruen(n) gibt ein Color-Objekt mit Grün-Ton n gemäß der obigen Darstellung zurück.
def gruen(n):    return color.Color( max(0,n-255), min(n,255), max(0,n-255) )
#-----
# blau(n) gibt ein Color-Objekt mit Blau-Ton n gemäß der obigen Darstellung zurück.
def blau(n):     return color.Color( max(0,n-255), max(0,n-255), min(n,255) )
```


Nun können wir Farbverläufe der drei Farben malen.

```
# farbverlaeufe.py (Fortsetzung)
#-----
# Das Testprogramm malt den Farbverlauf von blau beginnend mit schwarz
# und dahinter den Farbverlauf von rot beginnend mit weiß
# und dahinter den Farbverlauf von grün beginnend mit schwarz.
#-----
def testprogramm():
    stddraw.setCanvasSize(1200,800)
    stddraw.setXscale(0,1541)

    for i in range(0,511):
        stddraw.setPenColor( blau(i) )
        stddraw.filledRectangle(i, 0, 1.8, 1)

        stddraw.setPenColor( rot(510-i) )
        stddraw.filledRectangle(510+i, 0, 1.8, 1)

        stddraw.setPenColor( gruen(i) )
        stddraw.filledRectangle(1020+i, 0, 1.8, 1)

    stddraw.show()
#-----
if __name__=='__main__': testprogramm()
```

Nun können wir Farbverläufe der drei Farben malen.

```
# farbverlaeufe.py (Fortsetzung)
```

```
#-----
```

```
# Das Testprogramm malt den Farbverlauf von blau beginnend mit schwarz
```

```
# und dahinter den Farbverlauf von rot beginnend mit weiß
```

```
# und dahinter den Farbverlauf von grün beginnend mit schwarz.
```

```
#-----
```

```
def testprogramm():
```

```
    stddraw.setCanvasSize(1200,800)
```

```
    stddraw.setXscale(0,1541)
```

```
    for i in range(0,511):
```

```
        stddraw.setPenColor( blau(i) )
```

```
        stddraw.filledRectangle(i, 0, 1.8, 1)
```

```
        stddraw.setPenColor( rot(510-i) )
```

```
        stddraw.filledRectangle(510+i, 0, 1.8, 1)
```

```
        stddraw.setPenColor( gruen(i) )
```

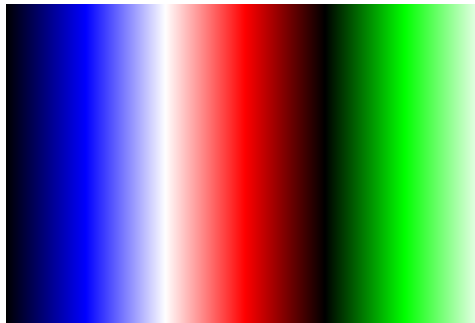
```
        stddraw.filledRectangle(1020+i, 0, 1.8, 1)
```

```
    stddraw.show()
```

```
#-----
```

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

python3 farbverlaeufe.py



Der Unterschied zwischen Methoden und Funktionen

```
a = 100
```

```
f = color.Color(a,a/2,a/3)
```

```
g = f.getGreen()
```

```
c = farbverlaeufe.gruen(a)
```

Der Unterschied zwischen Methoden und Funktionen

`a = 100` ← Objekt vom Typ `int`

`f = color.Color(a,a/2,a/3)`

`g = f.getGreen()`

`c = farbverlaeufe.gruen(a)`

Der Unterschied zwischen Methoden und Funktionen

`a = 100` ← Objekt vom Typ `int`

`f = color.Color(a,a/2,a/3)`

`g = f.getGreen()`

`c = farbverlaeufe.gruen(a)`

Modulname

Der Unterschied zwischen Methoden und Funktionen

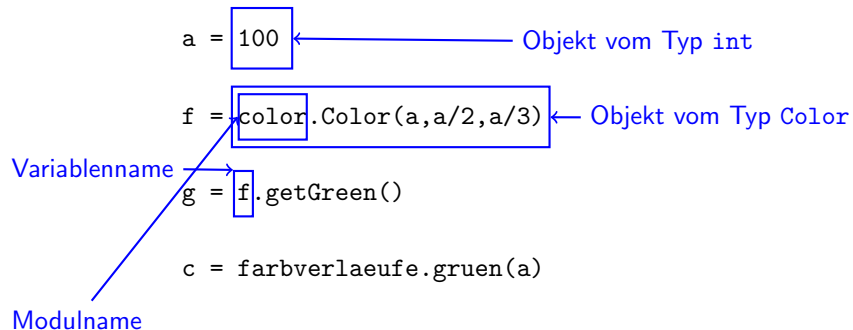
```
a = 100  
f = color.Color(a,a/2,a/3)  
g = f.getGreen()  
c = farbverlaeufe.gruen(a)
```

← Objekt vom Typ int

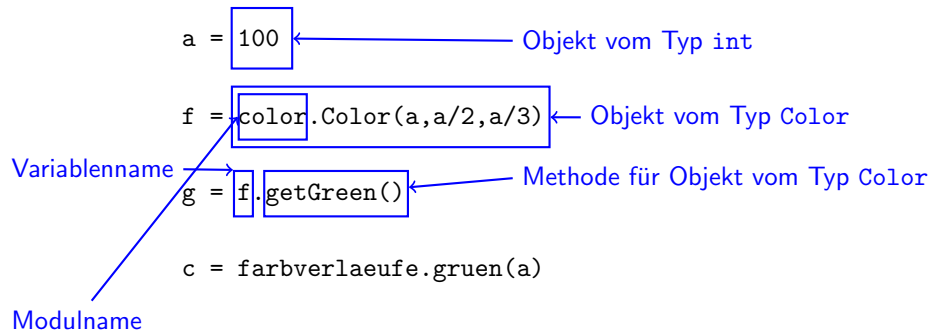
← Objekt vom Typ Color

Modulname

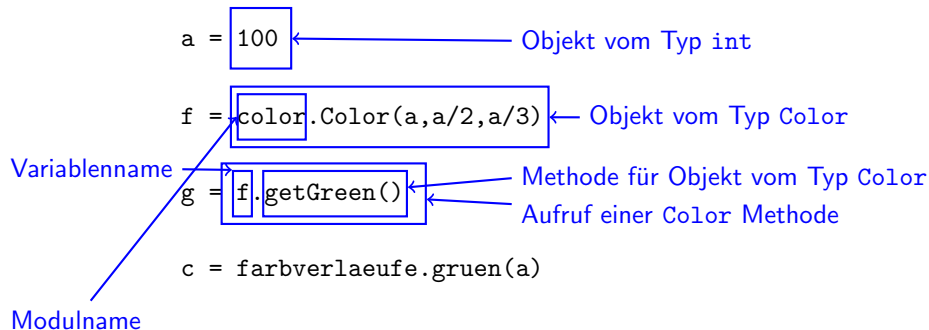
Der Unterschied zwischen Methoden und Funktionen



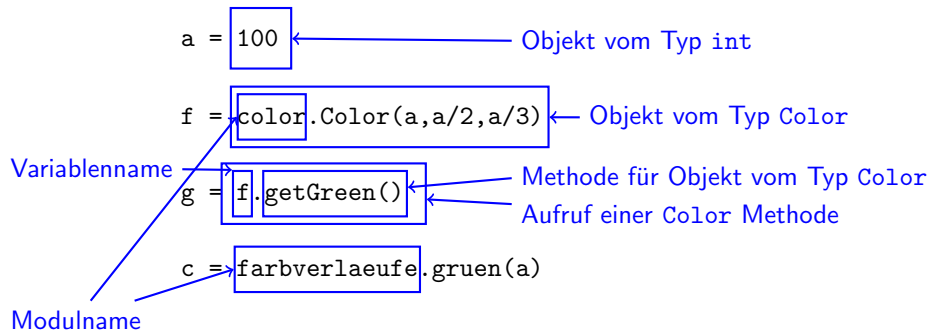
Der Unterschied zwischen Methoden und Funktionen



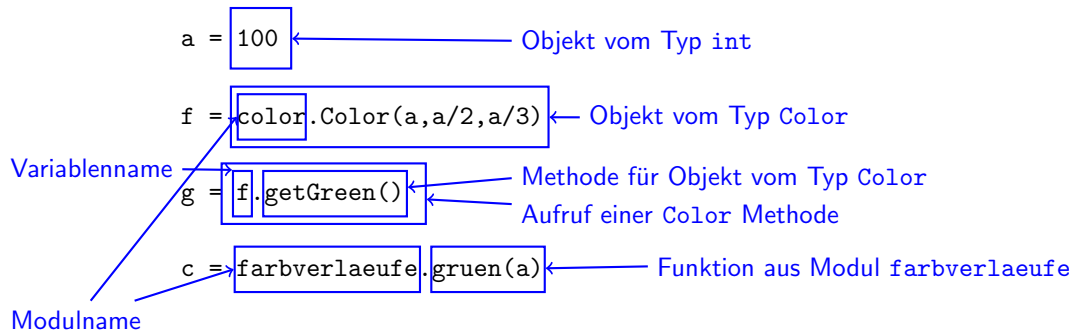
Der Unterschied zwischen Methoden und Funktionen



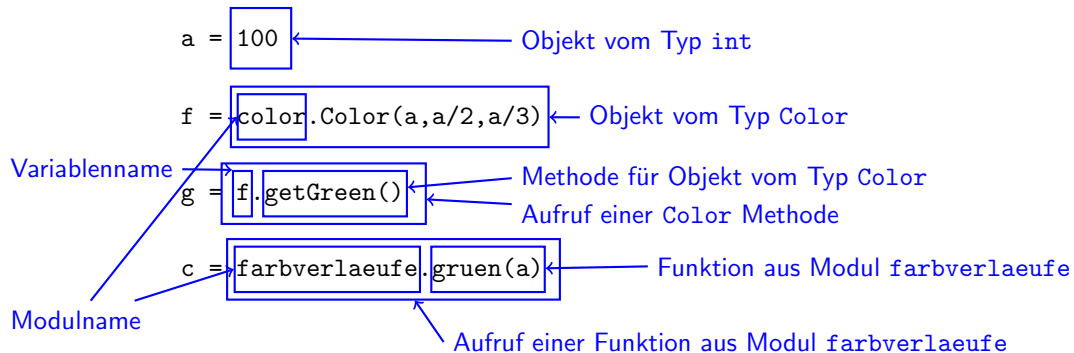
Der Unterschied zwischen Methoden und Funktionen



Der Unterschied zwischen Methoden und Funktionen



Der Unterschied zwischen Methoden und Funktionen



2 Zurück zum Programmierauftrag: Graphische Darstellung der Entwicklung der Jahresdurchschnittstemperaturen

Stelle die Entwicklung der durchschnittlichen Jahrestemperaturen aus Daten vom DWD graphisch durch ein Streifendiagramm dar.

Für jedes Jahr wird ein Streifen gemalt.

Seine Farbe stellt die Abweichung von der Durchschnittstemperatur aller Jahre dar.

Liegt die Jahrestemperatur unter dem Durchschnitt, wird sie durch einen Blau-Ton dargestellt. Je weiter sie unter der Durchschnittstemperatur liegt, desto dunkler ist das Blau.

Liegt die Jahrestemperatur über dem Durchschnitt, wird sie durch einen Rot-Ton dargestellt. Je weiter sie über der Durchschnittstemperatur liegt, desto dunkler ist das Rot.

Grobe Struktur des Programmablaufs und der Daten

Eingabedatei: 1824 13.50
1825 13.38
...
2017 15.85
2018 16.94

filtere die Temperaturen aus der Datei

[13.50, 13.38, ..., 15.85, 16.94]

wandle die Temperaturen in Farben um

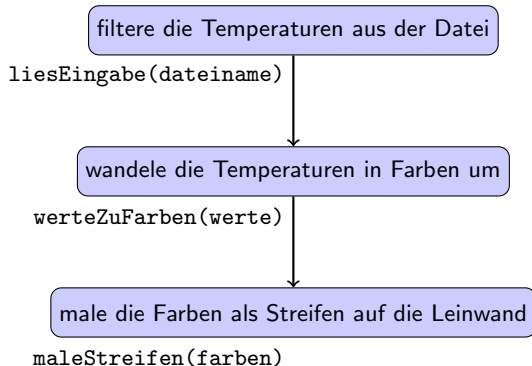
[Color(50,50,255), ..., Color(0,0,0)]

male die Farben als Streifen auf die Leinwand



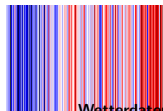
Grobe Struktur des Programmablaufs und der Daten

Eingabedatei: 1824 13.50
1825 13.38
...
2017 15.85
2018 16.94



[13.50, 13.38, ..., 15.85, 16.94]

[Color(50,50,255), ..., Color(0,0,0)]



APIs der Funktionen

Funktion	Bedeutung
<code>liesEingabe(dateiname)</code>	gibt die Folge von 2.Spalten (Trennzeichen ' ') aller Zeilen der Datei <code>dateiname</code> (str) als Array zurück.
<code>maleStreifen(f)</code>	<code>f</code> ist ein Array aus Farben (Color). Die Farben werden der Reihe nach von links nach rechts auf die Leinwand von <code>stddraw</code> gemalt.
<code>werteZuFarben(w)</code>	<code>w</code> ist ein Array aus Zahlen. Jede Zahl wird in eine Farbe (Color) umgewandelt, so dass Zahlen unter dem Durchschnitt von <code>w</code> blau bzw. über dem Durchschnitt von <code>w</code> rot ergeben. Je weiter der Wert vom Durchschnitt entfernt ist, desto dunkler ist die Farbe. Das Array aus Farben wird zurückgegeben.
<code>skalriere(wert, unten, oben)</code>	gibt einen ganzzahligen Wert aus dem Bereich 0...510 zurück, der <code>wert</code> (float) auf der Skala <code>unten...oben</code> (floats) nach dem Umskalieren auf die Skala 0...510 entspricht.

Die grobe Struktur von `werteZuFarben(werte)`

Umrechnung eines Wertes w in einen Farbton:

1. Der Wert w liegt unter der Durchschnittstemperatur aller Jahre:
 - 1.1 Skaliere den Bereich *minimale Temperatur*...*Durchschnittstemperatur* um zu 0...510.
Das Ergebnis ist der Blauton mit dem umskalierten Wert von w .
2. Der Wert w liegt über der Durchschnittstemperatur aller Jahre:
 - 2.1 Skaliere den Bereich *Durchschnittstemperatur*...*maximale Temperatur* um zu 510...0.
Das Ergebnis ist der Rotton mit dem umskalierten Wert von w .

```

# temperaturStreifen.py
#-----
# Liest den Namen einer Datei von der Kommandozeile, deren Zeilen aus einer Jahreszahl und
# einem Zahlenwert (float) bestehen (z.B. Jahresdurchschnittstemperaturen).
# Die Zahlenwerte werden als Streifendiagramm graphisch dargestellt.
# Die Werte unter dem Durchschnitt werden durch blaue Streifen dargestellt (je kleiner, umso dunkler),
# die Werte über dem Durchschnitt werden durch rote Streifen dargestellt (je größer, umso dunkler).
#-----
import sys, stddraw
from farbverlaeufer import blau, rot
#-----
# liesEingabe(dateiname) liest die Eingabedatei aus der Datei mit Namen dateiname (string).
# Die Datei wird als Tabelle aus zwei Spalten (getrennt durch ' ') aufgefasst.
# Die zweite Spalte wird in einem Array zurückgegeben.
def liesEingabe(dateiname):
    # Öffne die Datei und lies die erste Zeile.
    datei = open(dateiname, 'r')
    # werte speichert die Werte der zweiten Spalte jeder Zeile.
    werte = []
    # Solange aus der Datei noch eine Zeile gelesen werden konnte:
    #   speichere den Wert aus der 2.Spalte in werte und lies die nächste Zeile.
    zeile = datei.readline()
    while zeile != '':
        wert = float( str.split(zeile)[1] )
        werte += [ wert ]
        zeile = datei.readline()
    # Gib alle eingelesenen Werte zurück.
    return werte

```

```

# Fortsetzung von temperaturStreifen.py
#-----
# Gibt einen ganzzahligen Wert aus dem Bereich 0..510 zurück, der wert auf der
# Skala von unten..oben nach dem Umskalieren auf 0..510 entspricht.
def skaliere(wert, unten, oben):
    m = 510*(wert-unten)/(oben-unten)
    return round(m)
#-----
# Erhält ein Array aus Zahlenwerten und wandelt sie in Farben um.
# Werte unter dem Mittelwert werden in Blautöne umgewandelt. Kleine Werte haben dunklere Blautöne als größere Werte.
# Werte über dem Mittelwert werden in Rottöne umgewandelt. Kleine Werte haben hellere Rottöne als größere Werte.
# Das Array mit den Farben wird zurückgegeben.
def werteZuFarben(werte):
    # Bestimme den Bereich der Werte und seinen Durchschnittswert.
    durchschnittswert = sum(werte)/len(werte)
    minWert = min(werte)
    maxWert = max(werte)
    # Das Array farben wird am Ende in farbe[i] die Farbe zu werte[i] enthalten.
    farben = []
    # Wandle die Werte der Reihe nach in Farben um.
    for w in werte:
        # Falls der Wert kleiner als der Durchschnitt ist, wird er umskaliert und in einen Blauton umgewandelt.
        # Sonst wird er umskaliert und in einen Rotton umgewandelt.
        if w<=durchschnittswert:
            farben += [ blau( skaliere(w, minWert, durchschnittswert) ) ] # Je kleiner w, umso dunkler das Blau.
        else:
            farben += [ rot( 510 - skaliere(w, durchschnittswert, maxWert) ) ] # Je größer w, umso dunkler das Rot.
    return farben

```

```

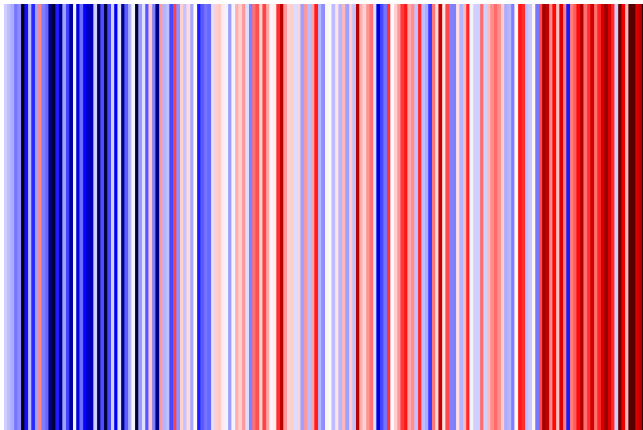
# Fortsetzung von temperaturStreifen.py
#-----
# Male die Farben als Streifen auf die Leinwand.
def maleStreifen(farben):
    # Male die Streifen in den angegebenen Farben der Reihe nach von links nach rechts auf die Leinwand.
    for i in range(0,len(farben)):
        stddraw.setPenColor(farben[i])
        stddraw.filledRectangle(i, 0, 1.1, 1)

#-----
# Lies die darzustellenden Werte aus einer Datei,
# wandele die Werte in Farben um und male sie als Streifen auf die Leinwand.
def hauptprogramm(dateiname):
    # Lies die darzustellenden Werte aus der Datei, deren Name von der Kommandozeile gelesen wird.
    werte = liesEingabe(dateiname)
    # Wandele die Werte in Farben um.
    farben = werteZuFarben(werte)
    # Richte die Leinwand zum Malen ein.
    stddraw.setCanvasSize(1200,800)
    stddraw.setXscale(0, len(werte))
    # Male die Farben als Streifen auf die Leinwand und zeige das Bild an.
    maleStreifen(farben)
    stddraw.show()

#-----
if __name__=='__main__': hauptprogramm(sys.argv[1])
#-----
# python3 temperaturStreifen.py Jena-JahresMittel-Temp.txt
#-----

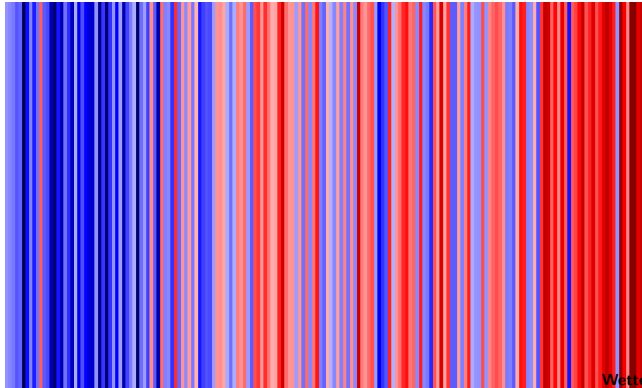
```

```
python3 temperaturStreifen.py Jena-JahresMittel-Temp.txt
```



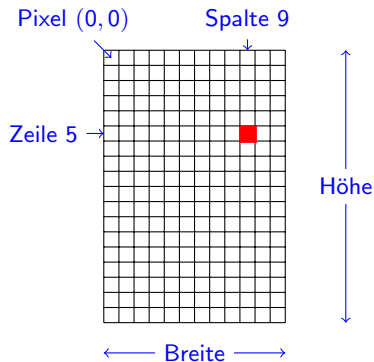
Um die fast schwarzen Streifen und die weißen Streifen zu vermeiden, kann man in einen kleineren Bereich umskalieren (im Beispiel 0...360), den man dann in die Mitte des ganzen Bereichs 0...510 verschiebt (also um 75). Dadurch gibt es keine Werte in den Randbereichen 0...75 und 510-75...510.

```
def skaliere(wert, unten, oben):  
    m = 360*(wert-unten)/(oben-unten) + 75  
    return round(m)
```



3 Ein Datentyp für digitale Bilder

Die Klasse Picture



Ein digitales Bild stellt man sich als ein Gitter aus Pixeln vor. Jedes Pixel hat eine Position (Spalte,Zeile) und eine Farbe. Die Zählung beginnt oben links mit Spalte 0 und Zeile 0. Im Beispiel links ist das Pixel (9,5) rot.

Die Klasse `Picture` ist im Modul `picture.py` definiert.

Sie stellt Methoden zum Lesen und Schreiben von Bildern und deren Pixeln zur Verfügung.

API für den Datentyp Picture in picture.py

<i>operation</i>	<i>description</i>
<code>Picture(w, h)</code>	<i>a new w-by-h array of pixels, initially all black</i>
<code>Picture(filename)</code>	<i>a new picture, initialized from filename</i>
<code>pic.save(filename)</code>	<i>save pic to filename</i>
<code>pic.width()</code>	<i>the width of the pic</i>
<code>pic.height()</code>	<i>the height of the pic</i>
<code>pic.get(col, row)</code>	<i>the Color of pixel (col, row) in pic</i>
<code>pic.set(col, row, c)</code>	<i>set the Color of pixel (col, row) in pic to c</i>

Note: Filename must end in .png or .jpg, which denotes the file format.

Our Picture data type (picture.py)

<i>operation</i>	<i>description</i>
<code>std draw.picture(pic, x, y)</code>	<i>display pic in std draw, centered at (x, y)</i>

Note: Both x and y default to the center of the standard drawing canvas.

Displaying a Picture object

Anzeigen eines Bildes

```
# bildAnzeigen.py
# Liest den Dateinamen eines Bildes von der Komandozeile und zeigt das Bild an.
#-----
import sys, stddraw
from picture import Picture
#-----
# Richte die Leinwand von stddraw so ein, dass ihre Breite bzw. Höhe
# die Breite bzw. Höhe des anzuzeigenden Bildes ist.
def richteLeinwandEin(bild):
    breite = bild.width()
    hoehe = bild.height()
    stddraw.setCanvasSize(breite, hoehe)
#-----
# Zeige das Bild bild (Picture) auf der Leinwand von stddraw an.
def bildAnzeigen(bild):
    richteLeinwandEin(bild)
    stddraw.picture(bild)
    stddraw.show()
#-----
# Lies den Dateinamen eines Bildes (.jpg oder .png) von der Kommandozeile und zeige es an.
def modultest():
    bild = Picture(sys.argv[1])
    bildAnzeigen(bild)
#-----
if __name__ == '__main__': modultest()
#-----
```

```
#-----
# python3 bildAnzeigen.py Spektraltorte.jpg
```



Die Leuchtdichte einer Farbe

Die *Leuchtdichte* (Luminanz) einer RGB-Farbe (r, g, b) ist

$$E(r, g, b) = 0.299 \cdot r + 0.587 \cdot g + 0.114 \cdot b \quad .$$

Man benutzt sie, um Farbfotos schwarz-weiß zu drucken.

Eine RGB-Farbe (x, x, x) mit gleichem Anteil aller Farben erscheint uns grau.

Ersetzt man in einem Farbfoto jede Farbe (r, g, b) durch $(E(r, g, b), E(r, g, b), E(r, g, b))$, können wir das entstandene Schwarz-Weiß-Bild gut erkennen.

Die Leuchtdichte ist auch wichtig beim farbigen Schreiben auf farbigem Hintergrund.

Die Differenz der Leuchtdichten der beiden Farben sollte mindestens 128 sein.

Anderenfalls kann man die Schrift nicht deutlich erkennen.

```
# luminanz.py
```

```
#-----
```

```
import color
```

```
#-----
```

```
# luminanz(farbe) gibt die Luminanz von farbe (Typ Color) zurück.
```

```
def luminanz(farbe):
```

```
    l = 0.299*farbe.getRed() + 0.587*farbe.getGreen() + 0.114*farbe.getBlue()
```

```
    return round(l)
```

Programmier-Auftrag: mache aus einem Farbbild ein Schwarzweißbild

Idee:

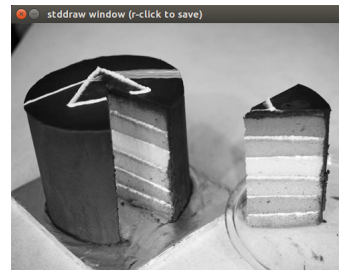
Ändere die Farbe jedes Pixels in seine Graustufe.

Die Graustufe einer Farbe f ist die Farbe,
die für jeden der drei RGB-Werte die Luminanz von f hat.

```

# farbeNachSW.py
#-----
# Liest den Dateinamen eines Bildes von der Kommandozeile und zeigt es in schwarz-weiß an.
#-----
import sys, stddraw, color, luminanz, picture, bildAnzeigen
#-----
# graustufe(f) gibt die Graustufe (Typ Color) der Farbe f (Typ Color) zurück.
def graustufe(f):
    l = luminanz.luminanz(f)
    return color.Color(l,l,l)
#-----  Hauptprogramm -----
# Lies den Dateinamen des Bildes von der Kommandozeile und erzeuge das Picture-Objekt foto daraus.
foto = picture.Picture(sys.argv[1])
# Zeige das Bild an.
bildAnzeigen.richteLeinwandEin(foto)
stddraw.picture(foto)
stddraw.show(1)
# Gehe zeilen- und spaltenweise alle Pixel des Bildes durch
# und ersetze dabei jede Pixel-Farbe durch ihre Luminanz.
# Wenn eine Zeile ersetzt wurde, dann zeige das Bild an.
for z in range(foto.height()):
    for s in range(foto.width()):
        pixel_farbe = foto.get(s,z)
        g = graustufe(pixel_farbe)
        foto.set(s,z,g)
    stddraw.picture(foto)
    stddraw.show(1)
stddraw.show()

```



Farbverläufe

Wir haben Farbverläufe bereits mit `std::draw` gemalt.

Es geht auch mit `Picture`. Dabei müssen wir selber jedes Pixel des Bildes bemalen.

Das Problem der „weißen Streifen“ durch Rundungsfehler bei der Pixel-Berechnung entfällt dabei.

```
# farbverlaeufePicture.py
```

```
#-----
```

```
from picture import Picture
from bildAnzeigen import bildAnzeigen
from farbverlaeufe import rot, blau
```

```
#-----
```

```
# Das Programm malt den Farbverlauf von blau beginnend mit schwarz
# und dahinter den Farbverlauf von rot beginnend mit weiß.
```

```
#-----
```

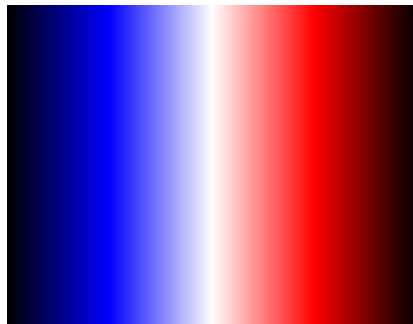
```
p = Picture(1021, 800)
```

```
for i in range(0,511):
    c = blau(i)
    for j in range(800): p.set(i,j, c)
```

```
for i in range(0,511):
    c = rot(510-i)
    for j in range(800): p.set(510+i,j, c)
```

```
bildAnzeigen(p)
```

python3 farbverlaeufePicture.py



- ▶ Mit Datentypen/Klassen werden verschiedenste Daten zum Rechnen nutzbar gemacht. In einem API liest man, welche Nutzung möglich ist.
- ▶ Methoden für Klassen ermöglichen eine abstrakte Handhabung der Daten.
- ▶ Wir kennen die Schreibweise für die Anwendung von Methoden auf Daten-Objekte und den Unterschied zur Anwendung von Funktionen.