Objektorientierung mit Python

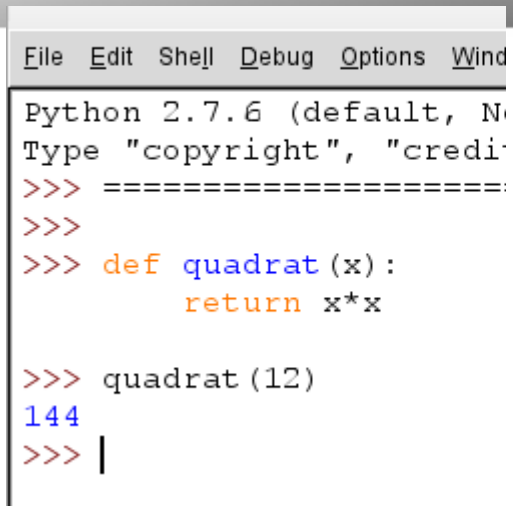
LIF-Qualifikationskurs
2020 / 2021

Objektorientierung mit Python

Die Idee zur Reduzierung der Probleme im zweistündigen Kurs:

- kein Wechsel der Programmiersprache in der Oberstufe
- Wahl einer Programmiersprache, mit der die Schülerinnen und Schüler ggf. schon in der Mittelstufe arbeiten können
- GUI-Programmierung, bei der die Schülerinnen und Schüler weitgehend selbstständig arbeiten können

Objektorientierung mit Python



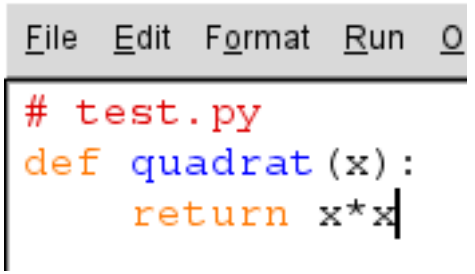
Python ist interaktiv

- Anweisungen kann man direkt auswerten und das Ergebnis direkt angezeigt bekommen
- Ein Python-Programm kann sofort ausgetestet werden
- Der Editor [z.B. IDLE] ist ein struktureller Editor, berücksichtigt die Python-Regeln

Objektorientierung mit Python

Python ist interaktiv

- Anweisungen kann man direkt auswerten und das Ergebnis direkt angezeigt bekommen
- Ein Python-Programm kann sofort ausgetestet werden



- Der Editor [z.B. IDLE] ist ein struktureller Editor, berücksichtigt die Python-Regeln

Objektorientierung mit Python

Python ist typgebunden

- Variable kennen ihren Typ
- keine Deklaration von Typen
- a=10
  definiert eine Variable a und weist ihr den Wert 10 zu, der damit Zahltyp ist
- a="10" [oder a='10' mit derselben Wirkung]
  definiert den String

Objektorientierung mit Python

Python verwendet Referenzen und Referenzzähler

```
>>> a=[1,2,3]
>>> b=a
>>> b
[1, 2, 3]
>>> b[1]=12
>>> a
[1, 12, 3]
>>> a=[1,2,3]
>>> b=a
>>> b
[1, 2, 3]
>>> b=[10,11,12]
>>> a
[1, 2, 3]
```

- kein neues Objekt,
  nur eine zweite Referenz
- Veränderungen am Objekt ändern beide
- Neue Zuweisung für die zweite Referenz erzeugt ein neues Objekt, die erste bleibt unverändert
- Achtung auch bei Prozeduren/Funktionen

Objektorientierung mit Python

Python ist prozedural

- Anweisungen werden sequenziell ausgeführt
- Mit der Anweisung
  ```
  def <Name>(<par>):
      <Anweisung 1>
      <Anweisung 2>
  ```
  können Prozeduren definiert werden

Objektorientierung mit Python

Python ist funktional

- Mit der Definition
  ```
  def <Name>(<par>):
      <...>
      return <Rückgabewert>
  ```
  können Funktionen definiert werden
- Funktionsschachtelungen sind möglich
- Rekursion ist möglich

Objektorientierung mit Python

Python ist funktional

- return → liefert Wert zurück
- u.a. anonyme Funktion: *(leider etwas kompliziert)*
  `lambda x,y: sqrt((x-zX)*(x-zX)+(y-zY)*(y-zY))`
  ist die Abstandsfunktion

```
def Abstand_vom_Zentrum(zX,zY):
    return lambda x,y: sqrt((x-zX)*(x-zX)+(y-zY)*(y-zY))
```

Für ein Zentrum Z(3,4)

```
abstand_von_Z=Abstand_vom_Zentrum(3,4)

abstand_von_Z(19,16)

→ 20.0
```

- Funktionen können Parameter sein

Objektorientierung mit Python

Python ist voll funktional mit einem **Mangel**

- Python erkennt keine Endrekursion!
- Das führt leider unnötigerweise oft zu Rekursionsabbruch mit Fehler

Objektorientierung mit Python

Python ist objektorientiert

- Ein großer Teil der Sprache ist objektorientiert definiert
- Das hat auch Probleme zur Folge (s.o.):
  - Listen sind Objekte
  - call-by-reference
  - Methoden verändern das Objekt dauerhaft
  - Zuweisungen (a=b) erzeugen keine neuen Objekte

Objektorientierung mit Python

Objektorientierung mit Python

Syntax am Beispiel

- Klassenkopf
- Konstruktor

```
class Zaehler:
    def __init__(self):
        self.stand = 0
```

Objektorientierung mit Python

Syntax am Beispiel

- Klassenkopf

```
class Zaehler:
    def __init__(self):
        self.stand = 0
```

Objektorientierung mit Python

Syntax am Beispiel

- Klassenkopf mit nachfolgendem Doppelpunkt
- und Einrückung

```
class Zaehler:
    def __init__(self):
        self.stand = 0
```

Syntax am Beispiel

- Konstruktor ist Methode mit besonderer Kennzeichnung

```
class Zaehler:
    def __init__(self):
        self.stand = 0
```

Syntax am Beispiel

- Das Objekt selbst wird mit self gekennzeichnet und muss dem Konstruktor als Parameter übergeben werden.

```
class Zaehler:
    def __init__(self):
        self.stand = 0
```

Syntax am Beispiel

- Auf eine Instanzvariable wird mit self vor dem Namen zugegriffen

```
class Zaehler:
    def __init__(self):
        self.stand = 0
```

Objektorientierung mit Python

Zugriffe auf Attribute und Methoden von Objekten

- immer mit Punktnotation

```
class Zaehler:
    def __init__(self):
        self.stand = 0
```

Eine prozedurale Methode (der Instanzen)

- Einbau der Zaehle()-Methode und ...

```
class Zaehler:
    def __init__(self):
        self.stand = 0
    def Zaehle(self):
        self.stand += 1
    def ZeigeStand(self):
        return self.stand
```

Eine funktionale Methode (der Instanzen)

- … Einbau der ZeigeStand()-Methode

```
class Zaehler:
    def __init__(self):
        self.stand = 0
    def Zaehle(self):
        self.stand += 1
    def ZeigeStand(self):
        return self.stand
```

Vererbung

- (nicht überraschend) durch Angabe der vererbenden Klasse hinter dem Klassennamen in Klammern

```
class ZyklischerZaehler(Zaehler):
    def __init__(self, zyklusLaenge):
```

Vererbung

- Der Aufruf des Konstruktors der vererbenden Klasse ist zwingend notwendig!

```
class ZyklischerZaehler(Zaehler):
    def __init__(self, zyklusLaenge):
        Zaehler.__init__(self)
```

Objektorientierung mit Python

ACHTUNG:
Python kennt Mehrfachvererbung!

Vererbung

- (nicht überraschend) durch Angabe der vererbenden Klasse hinter dem Klassennamen in Klammern

```
class ErbtVonZwei(Erste_Klasse , ZweiteKlasse):
    def __init__(self):
        Erste_Klasse.__init__(self)
        Zweite_Klasse.__init__(self)
```

OO Grafik mit wxPython

OO Grafik mit wxPython

- Zu Python gibt es mehrere Grafik-Toolkits
- Eine Erläuterung findet man im Python-Buch von Galileo-Computing, das man auch als e-book herunterladen konnte.
  [Galileo-Computing gibt es nicht mehr, ist heute Rheinwerk-Verlag]
- Dort sind etwas ausführlicher beschrieben
  - Tkinter
  - PyGtk
  - PyQt

OO Grafik mit wxPython

- Wir arbeiten mit wxPython
- Für Python 2 und Windows:
  - Herunterladen von http://wxpython.org/
  - Passende Version zur eigenen Python-Version verwenden
  - Installation erfolgt in den Ordner <Python>/Lib/sitepackages

OO Grafik mit wxPython

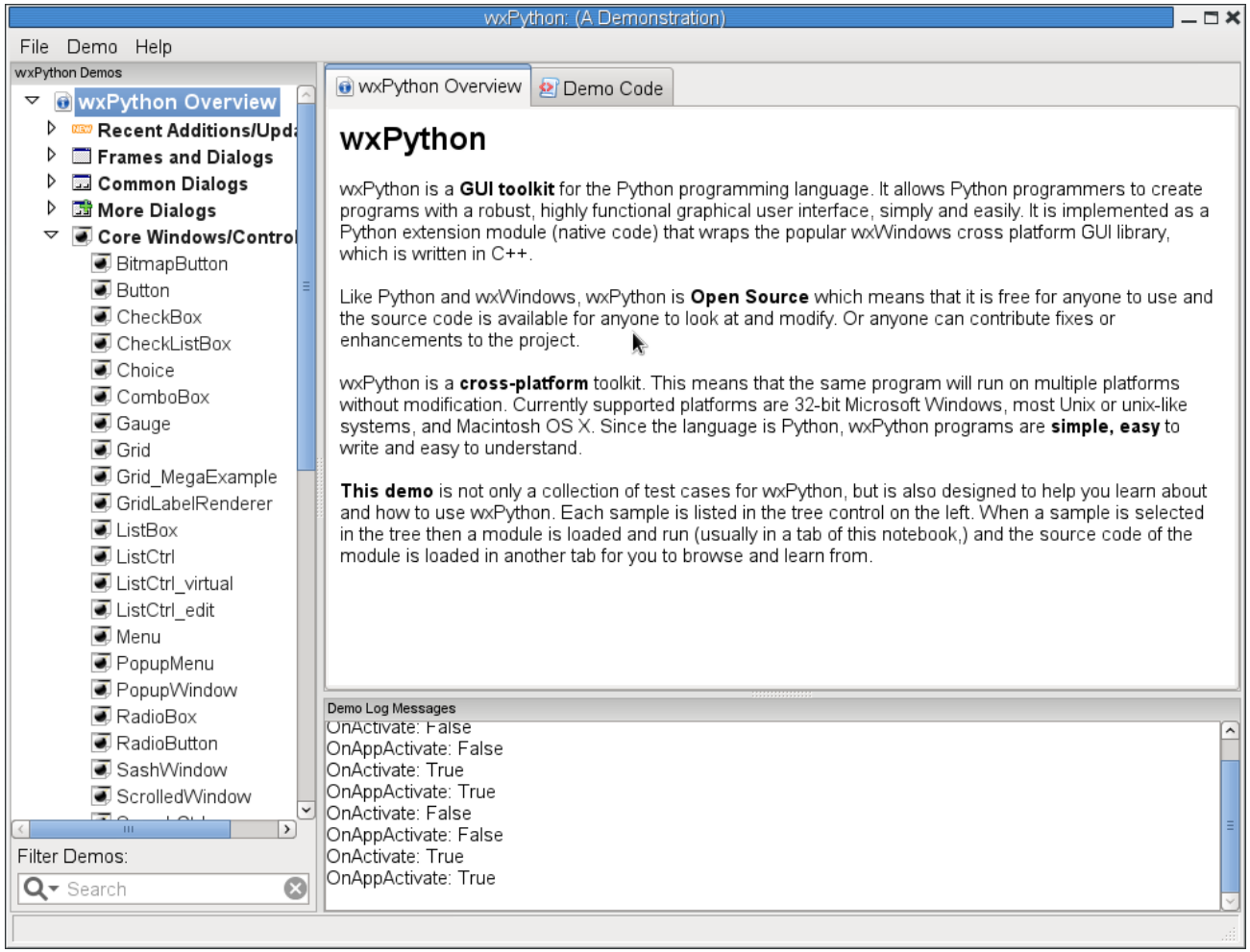
- Wir arbeiten mit wxPython
- Für Python 3 heißt es Phoenix
- installieren mit Hilfe von pip
  - Passende Version wird automatisch gewählt
  - Installation erfolgt in den Ordner <Python>/Lib/sitepackages

OO Grafik mit wxPython

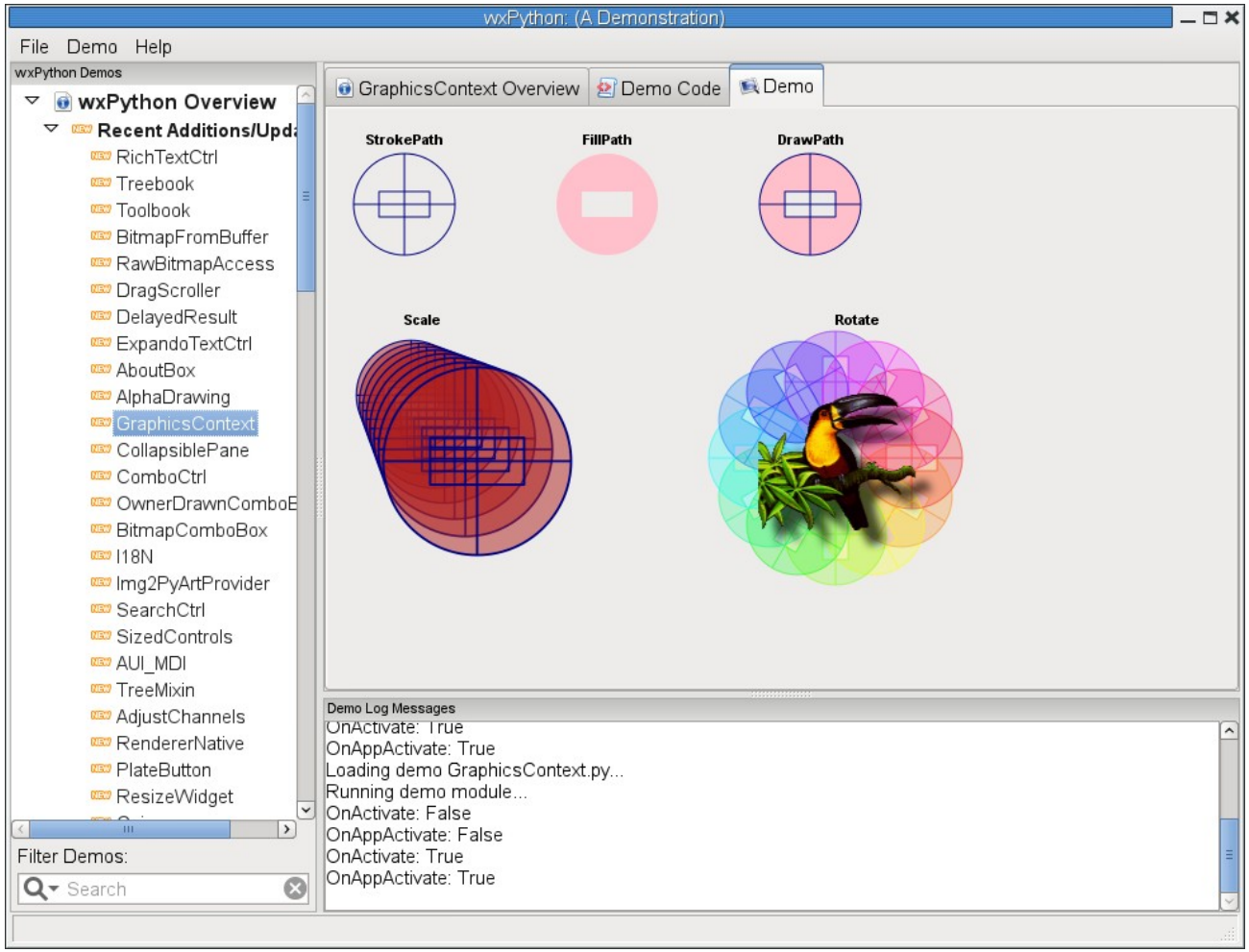
- Unbedingt auch die wxPython docs und demos herunterladen und installieren (noch unklar bei Phoenix)
- linux-Distributionen installieren beide in der Regel in die richtigen Verzeichnisse.

OO Grafik mit wxPython

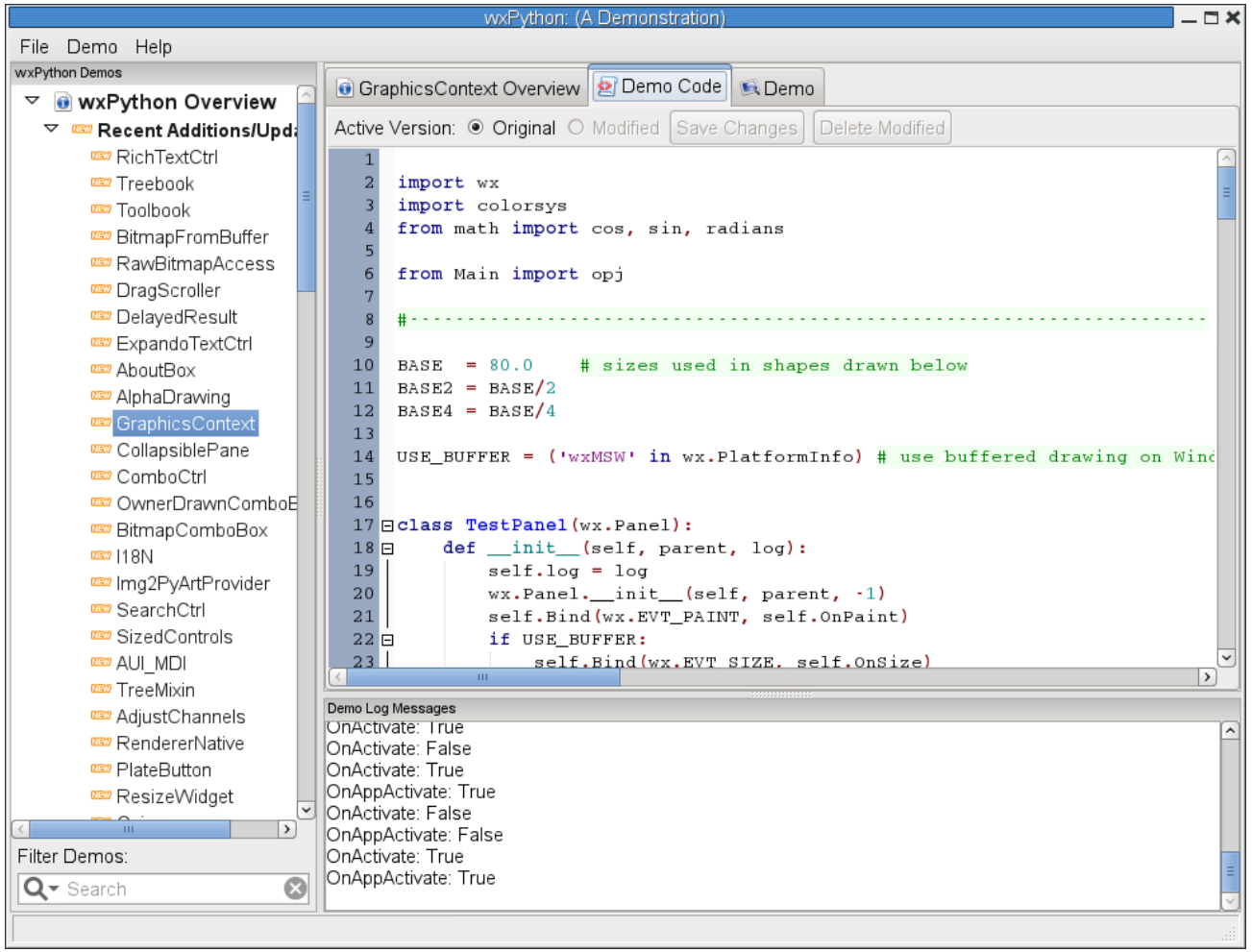
- Wir starten das Demotool und wählen ein Beispiel aus.



OO Grafik mit wxPython



OO Grafik mit wxPython



OO Grafik mit wxPython

- Die Lasche Demo Code zeigt den vollständigen Python-Code dieses Anwendungsfensters.
- Mit copy and paste kann man sich den Code in das Editorfenster von Python holen und weiter für eine eigenständige Anwendung bearbeiten.

OO Grafik mit wxPython

- In unserem Fortbildungskurs starten wir aber anders, nämlich mit einem schon "fertigen" Anwendungsprojekt.
- Es geht um die Entwicklung eines Raumplaners [Einrichtungsplaners].
- Öffnen Sie das Projekt Raumplaner-Ausgangsprojekt.py mit IDLE.