Simulation d'un oscilloscope

— projet algorithmique —

Alexis NEDELEC

LISYC EA 3883 UBO-ENIB-ENSIETA
Centre Européen de Réalité Virtuelle
Ecole Nationale d'Ingénieurs de Brest

Objectifs du semestre

Composer ses propres morceaux

- apprentissage de la conception d'algorithmes
- pédagogie par problèmes
- méthodologie de construction de programmes simples
- apprentissage par imitation

Objectifs du semestre

Composer ses propres morceaux

- apprentissage de la conception d'algorithmes
- pédagogie par problèmes
- méthodologie de construction de programmes simples
- apprentissage par imitation

Mini-projets

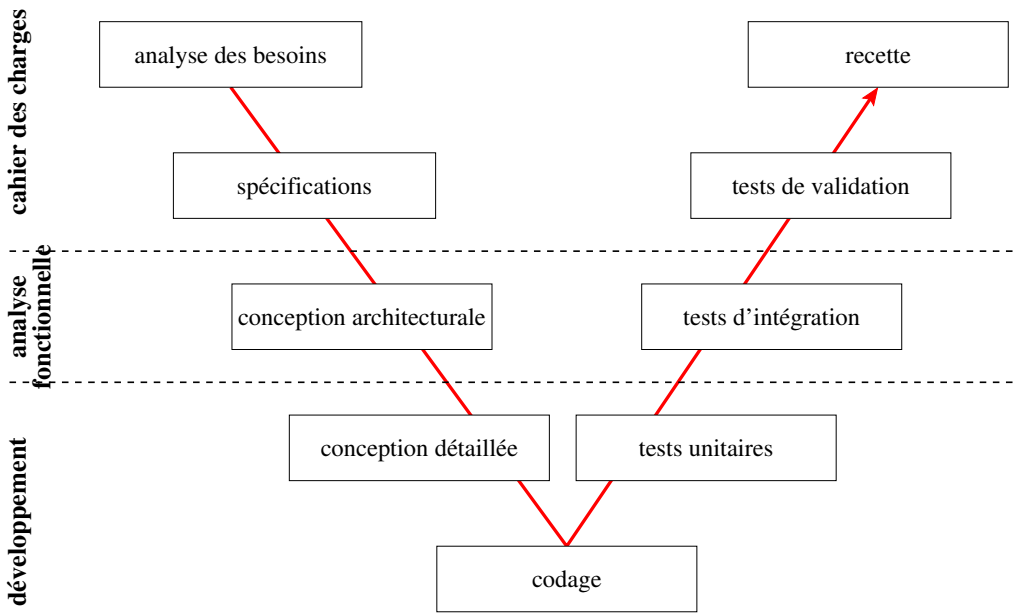
Cours : présentation de cas d'école

Exemples :

- mathématiques : résolution d'un système linéaire
- électronique : simulation d'un oscilloscope
- ...

TD : réalisation d'un mini-projet de type *jeu*

Mini-projets et cycle en V



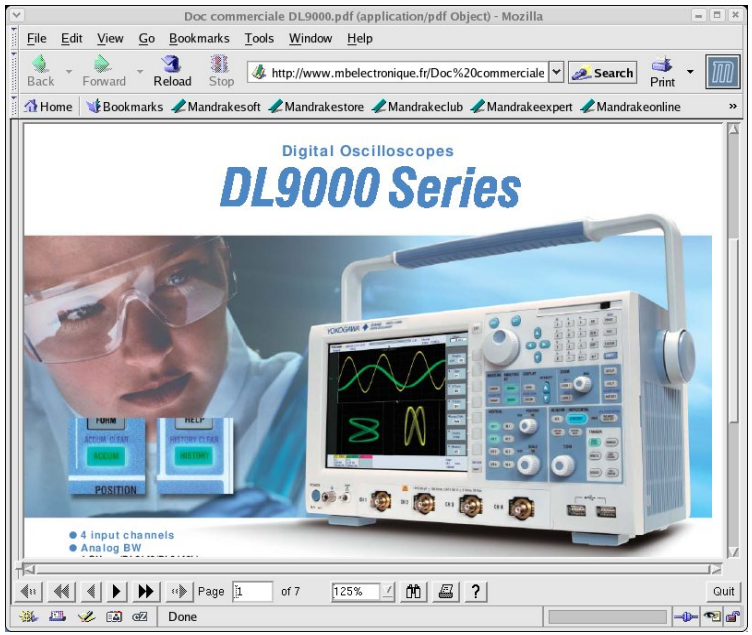
Trois phases principales :

1. cahier des charges : analyse et expression des besoins
2. analyse fonctionnelle : architecture modulaire et tests d'intégration
3. développement : codage et jeux de tests unitaires

Objectif

Simulation d'Oscilloscope

- visualisation de mouvement vibratoire harmonique
- mesure d'amplitude, fréquence et phase de signal
- mesure de différence de fréquence, phase entre 2 signaux
- modification d'amplitudes, fréquences et phases de signaux



Oscilloscope

Intérêt de l'oscilloscope

- visualiser les variations temporelles d'un signal électrique
- mesurer des grandeurs physiques à des instants très préçis,
- mesurer intervalles de temps, déphasages et périodes.

Oscilloscope

Intérêt de l'oscilloscope

- visualiser les variations temporelles d'un signal électrique
- mesurer des grandeurs physiques à des instants très préçis,
- mesurer intervalles de temps, déphasages et périodes.

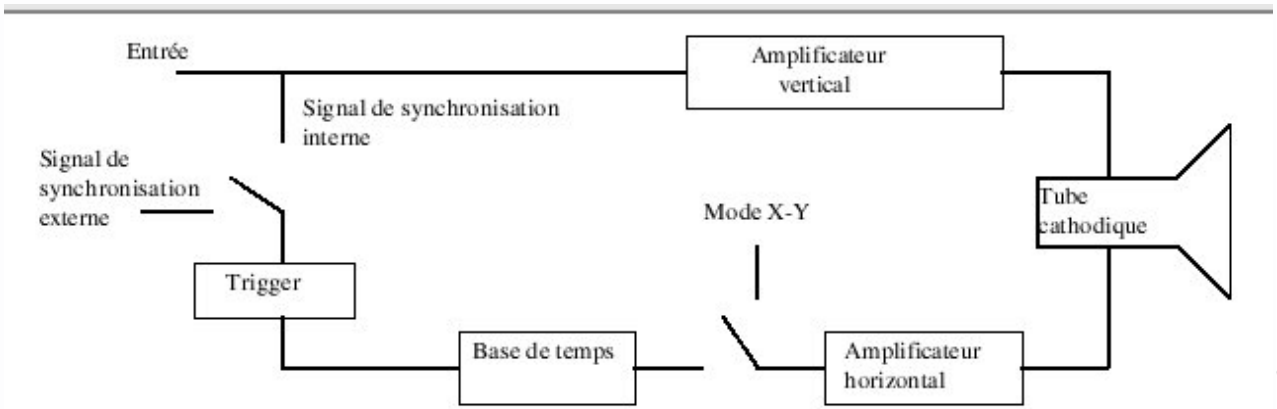
Intérêt de la simulation de l'oscilloscope

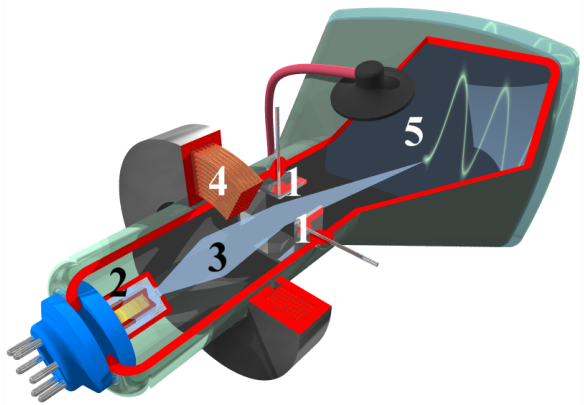
- fournir un outil pédagogique d'utilisation d'oscilloscope
- paramétrer simplement les signaux à visualiser
- associer directement la visualisation aux réglages effectués

sans avoir d'oscilloscope !

Oscilloscope

Principe de l'oscilloscope





- signal d'entrée (X) amplifié sur les plaques verticales (1)
- synchronisation base de temps sur les plaques horizontales (1)
- deuxième signal (X-Y), déconnexion base de temps

Simulation d'oscilloscope

Fonctions principales de la simulation

- Visualisation et animation de courbes de la forme : $e = A\ sin(2\ \pi\ f\ t\ +\ \varphi)$
- Réglages d'amplitude, fréquence et phase

Simulation d'oscilloscope

Fonctions principales de la simulation

- Visualisation et animation de courbes de la forme : $e = A\ sin(2\ \pi\ f\ t\ +\ \varphi)$
- Réglages d'amplitude, fréquence et phase

Fonctionnalités : visualisation et animation

- calcul des courbes correspondant aux signaux (X,Y) et (X-Y)
- création d'une grille d'écran (n carreaux)
- visualisation de courbe sur la grille
- représentation du spot de l'écran cathodique
- assocation du mouvement vibratoire au spot de l'écran

Simulation d'oscilloscope

Fonctionnalités : interfaces utilisateur

Gestion des signaux

- réglage de la base de temps
- paramétrage d'amplitude, fréquence et phase (X,Y)

Affichage de courbes

- en mode (X,Y) et (X-Y)

Entrée de signal

- choix du type de signal (X,Y)

Contraintes

Environnement informatique

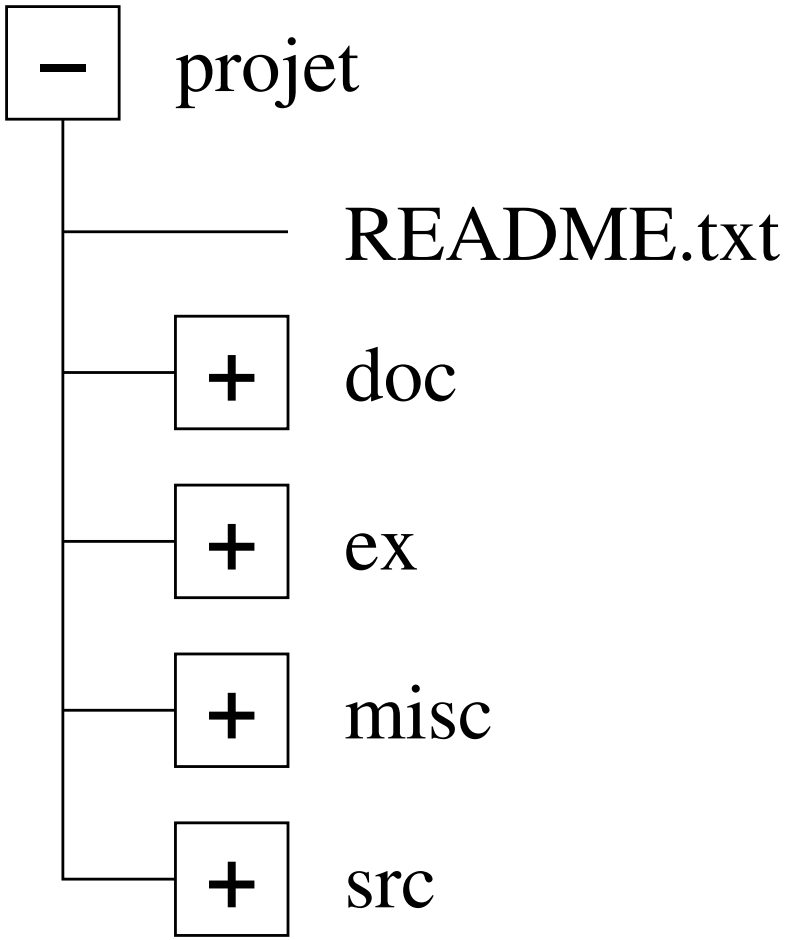
Matériel	:	ordinateur de bureau récent
Système d'exploitation	:	Linux, Windows XP, Mac OS
Langage	:	Python

Oscilloscope

Déplacement sur l'écran	:	$timebase + \delta t$ ($\delta t \leq 5.10^{-2}$)
Paramétrage de grille d'écran	:	largeur, hauteur, nb carreaux
Mesure de l'amplitude	:	fonction de la grille d'écran

Livrables

Archive `.tar` ou `.zip`

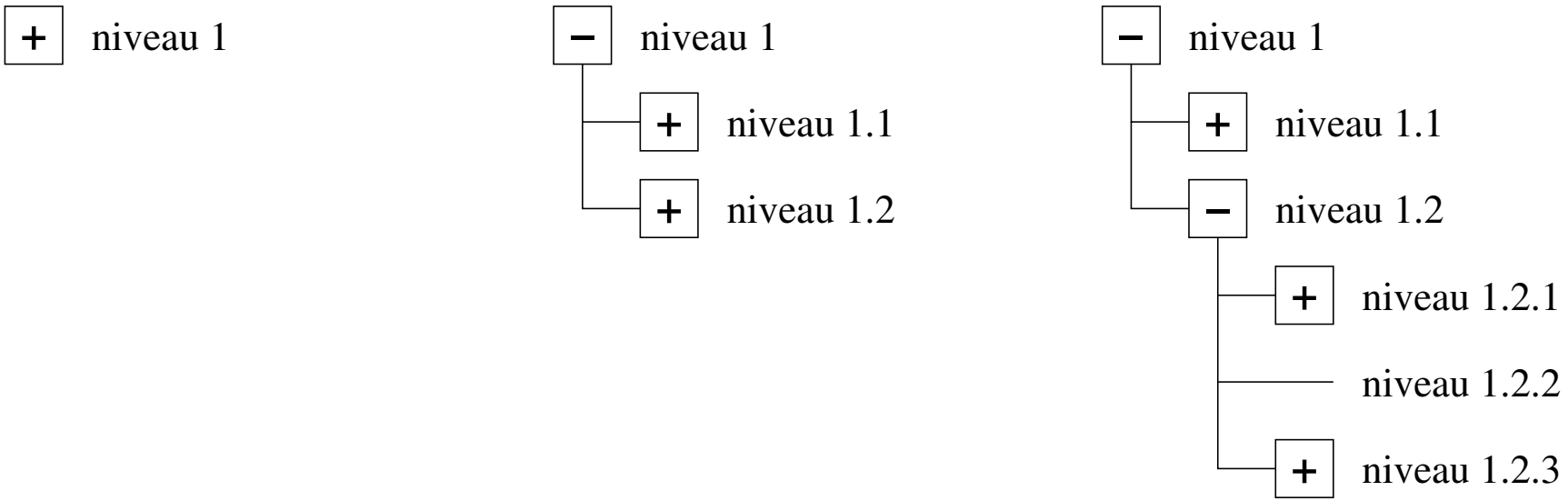


- `README.txt` : installation, configuration, lancement
- `doc` : documentations (cahier des charges, analyse fonctionnelle, guide de référence, guide de l'utilisateur, résultats des tests)
- `ex` : exemples d'utilisation (tests)
- `misc` : autres informations jugées utiles
- `src` : codes source

Analyse descendante

Définition

Méthode consistant à décomposer un problème complexe en sous-problèmes indépendants les uns des autres, et dont la complexité est moindre que le problème initial.



Arbres d'analyse fonctionnelle

Analyse descendante

Décomposition de l'application

- simulation de signaux : `signals.py`
- création d'une grille de visualisation : `oscilloGrid.py`
- visualisation de signaux (X,Y),(X-Y) : `oscilloCurve.py`
- création, animation du spot électronique : `oscilloSpot.py`
- réglage de la base de temps : `oscilloTimebase.py`
- affichage de courbe : `oscilloShowCurve.py`
- entrée de signal : `oscilloEvaluateCurve.py`
- paramétrage de courbe : `oscilloChangeCurveXY.py`
- réglage des signaux (X,Y),(X-Y) : `oscilloscope.py`

Analyse descendante

Décomposition de l'application

- signals.py
  - computeCurve(width,height,signal,timebase=0)
  - computeCurveXY(width,height,xsignal,ysignal)

Analyse descendante

Décomposition de l'application

- signals.py
  - computeCurve(width,height,signal,timebase=0)
  - computeCurveXY(width,height,xsignal,ysignal)
- oscilloGrid.py :
  - oscilloGrid(screen,step=8)

Analyse descendante

Décomposition de l'application

- signals.py
  - computeCurve(width,height,signal,timebase=0)
  - computeCurveXY(width,height,xsignal,ysignal)
- oscilloGrid.py :
  - oscilloGrid(screen,step=8)
- oscilloCurve.py :
  - createCurve(screen,curve,color)

Analyse descendante

Décomposition de l'application

- signals.py
  - computeCurve(width,height,signal,timebase=0)
  - computeCurveXY(width,height,xsignal,ysignal)
- oscilloGrid.py :
  - oscilloGrid(screen,step=8)
- oscilloCurve.py :
  - createCurve(screen,curve,color)
- oscilloSpot.py :
  - createSpot(screen,spot_w,color)
  - move(screen,t,xsignal,ysignal)

Analyse descendante

Décomposition de l'application

- signals.py
  - computeCurve(width,height,signal,timebase=0)
  - computeCurveXY(width,height,xsignal,ysignal)
- oscilloGrid.py :
  - oscilloGrid(screen,step=8)
- oscilloCurve.py :
  - createCurve(screen,curve,color)
- oscilloSpot.py :
  - createSpot(screen,spot_w,color)
  - move(screen,t,xsignal,ysignal)
- oscilloTimebase.py :
  - setTimeBase(t)

Analyse descendante

Décomposition de l'application

- `oscilloShowCurve.py` :
  - `showXCurve(event),showYCurve(event)`

Analyse descendante

Décomposition de l'application

- `oscilloShowCurve.py` :
  - `showXCurve(event),showYCurve(event)`
- `oscilloEvaluateCurve.py` :
  - `evaluateX(event),evaluateY(event)`

Analyse descendante

Décomposition de l'application

- oscilloShowCurve.py :
  - showXCurve(event),showYCurve(event)
- oscilloEvaluateCurve.py :
  - evaluateX(event),evaluateY(event)
- oscilloChangeCurveXY.py :
  - setXMagnitude(x),setYMagnitude(x)

Analyse descendante

Décomposition de l'application

- oscilloShowCurve.py :
  - showXCurve(event),showYCurve(event)
- oscilloEvaluateCurve.py :
  - evaluateX(event),evaluateY(event)
- oscilloChangeCurveXY.py :
  - setXMagnitude(x),setYMagnitude(x)
- oscilloscope.py : intégration finale
  - setXFrequency(x),setXPhase(x)
  - setYFrequency(x),setYPhase(x)

signals.py

signal en mode X ou Y (1/3)

```
1  def computeCurve(width,height,signal,\
2                   timebase=0) :
3      assert type(signal) is dict
4      t=0
5      x=0
6      y=0
7      curve_points=[]
8      curve=signal["curve"]
9      mag=signal["mag"]
10     freq=signal["freq"]
11     phase=signal["phase"]
```

`signal` : caractéristiques (`dict`) du mouvement vibratoire

signals.py

signal en mode X ou Y (2/3)

```
1        if timebase==0 :
2            x=width/2
3            e=mag
4            y=height/2-e*height/2
5            curve_points.append((x,y))
6            y=height/2+e*height/2
7            curve_points.append((x,y))
```

`timebase==0` : droite d'élongation de `e*height`

signals.py

signal en mode X ou Y (3/3)

```
1       else :
2           step=width/1000.0
3           while x<width :
4               x=(t*step)/timebase
5               e=mag*curve(2*pi*freq*t/1000
6                           +(pi*phase)/180)
7               y=height/2 - e*height/2
8               curve_points.append((x,y))
9               t=t+1
10      return curve_points
```

- calcul (x,y) sur l'écran (`x<width`)
- pour chaque `timebase` millième de seconde

signals.py

signal en mode X-Y

```
1  def computeCurveXY(width,height,\
2                     xsignal,ysignal) :
3      ...
4      curve_points=[]
5      while t<1000.0 :
6          x=xmag*(width/2)
7            *xcurve(2*pi*xfreq*t/1000.0
8                    +(pi*xphase)/180)+width/2
9          y=-ymag*(height/2)
10           *ycurve(...)+height/2
11         curve_points.append((x,y))
12         t=t+1
13     return curve_points
```

oscilloGrid.py

Grille de visualisation

```
1  def oscilloGrid(screen,step=8) :
2      screen_w=float(screen.cget("width"))
3      screen_h=float(screen.cget("height"))
4      stepX=screen_w/step
5      stepY=screen_h/step
6      for t in range(1,step) :
7          x=t*stepX
8          y=t*stepY
9          screen.create_line(x,0,x,screen_h,\
10                             fill="black")
11         screen.create_line(0,y,screen_w,y,\
12                             fill="black")
```

oscilloGrid.py

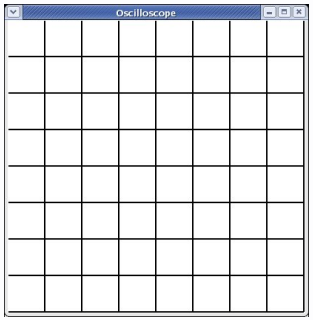
Structuration de l'application

```
1  ###### Globals                                  #####
2  ###### Functions                                #####
3  ###### GUI interactions                         #####
4  ###### GUI components                           #####
5  mw=Tk()
6  mw.title("Oscilloscope")
7  screen=Canvas(mw,width=400,height=400,\
8                bg="white")
9  ###### GUI layout management                    #####
10 screen.pack()
11 ###### Simulation init                          #####
12 oscilloGrid(screen)
13 mw.mainloop()
```

oscilloGrid.py

Tests de grille d'oscilloscope

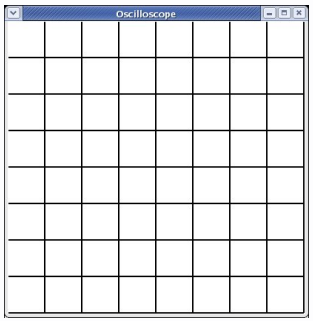
```
screen=Canvas(mw, width=400,height=400,bg="white")
oscilloGrid(screen)
```



oscilloGrid.py

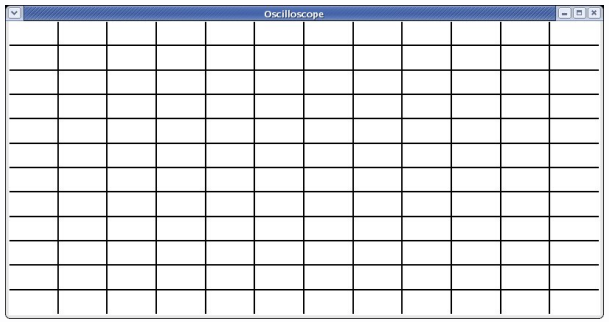
Tests de grille d'oscilloscope

```
screen=Canvas(mw, width=400,height=400,bg="white")
oscilloGrid(screen)
```



Tests de grille d'oscilloscope

```
screen=Canvas(mw, width=800,height=200,bg="white")
oscilloGrid(screen,12)
```



oscilloCurve.py

Création de courbes à l'écran

```
1 def createCurve(screen,curve,color) :
2     if curve :
3         return screen.create_line(curve,\
4                                   fill=color,\
5                                   smooth=1,\
6                                   width="1")
```

Importation et initialisation des signaux

```
1 from signals import *
2 xsignal={"curve":cos,"mag":1.0,\
3          "freq":1.0,"phase":0.0}
4 ysignal={"curve":sin,"mag":1.0,\
5          "freq":1.0,"phase":0.0}
```

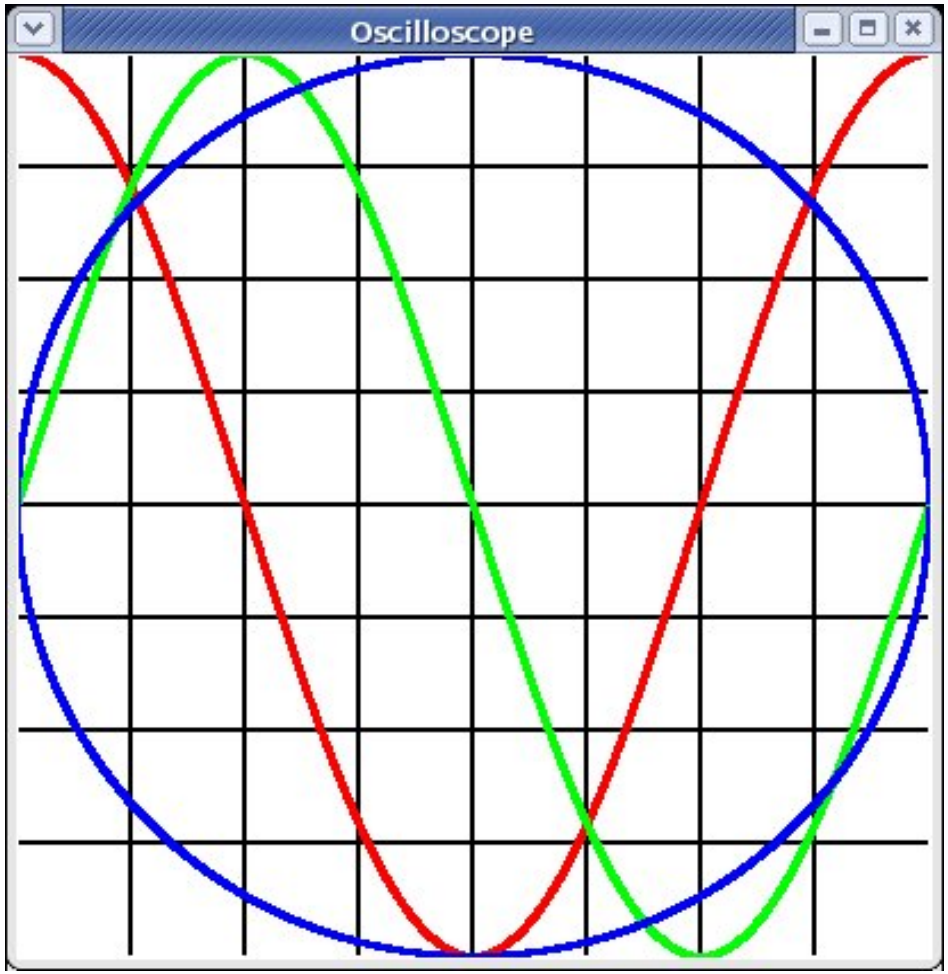
oscilloCurve.py

Calcul et affichage de signal à l'écran

```
1  curve=computeCurve(screen_w,screen_h,\
2                     xsignal,1)
3  color="red"
4  createCurve(screen,curve,color)
5  curve=computeCurve(screen_w,screen_h,\
6                     ysignal,1)
7  color="green"
8  createCurve(screen,curve,color)
9  curve=[]
10 curve=computeCurveXY(screen_w,screen_h,\
11                        xsignal,ysignal)
12 color="blue"
13 createCurve(screen,curve,color)
```

oscilloCurve.py

Tests d' affichage à l'écran



oscilloSpot.py

Création du spot à l'écran

```
1  def createSpot(screen, spot_w, color) :
2      screen_h=float(screen.cget("height"))
3      screen_w=float(screen.cget("width"))
4      spot=screen.create_oval(screen_w/2-5,\
5                              screen_h/2-5,\
6                              screen_w/2+5,\
7                              screen_h/2+5,\
8                              width=spot_w,\
9                              fill=color)
10     return spot
```

oscilloSpot.py

Déplacement du spot à l'écran (1/2)

```
1  def move(screen,t,xsignal,ysignal) :
2      global ...
3      ...
4      if ysignal["curve"]==-1 :
5          if timebase==0 :
6              x=screen_w/2
7          else :
8              x=(t*stepX)/timebase
9          e=xmag*xcurve(2*pi*xfreq*t/1000 \
10                       - (pi*xphase)/180)
11         y=screen_h/2-e*screen_h/2
```

`ysignal["curve"]==-1` : base de temps connectée

oscilloSpot.py

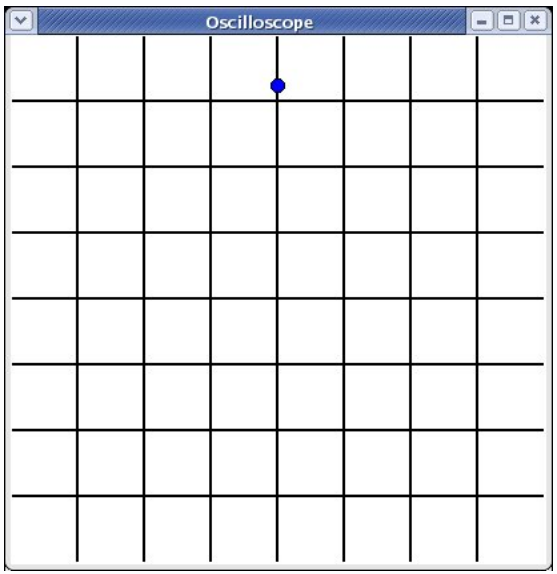
Déplacement du spot à l'écran (2/2)

```
1        else :
2            ycurve=ysignal["curve"]
3            ...
4            x=xmag*...*xcurve(...) + screen_w/2
5            y=-ymag*...*ycurve(...)+ screen_h/2
6        if x<=screen_w :
7            t=t+20
8        else :
9            x=0
10           t=0
11       screen.coords(spot,x-5,y-5,x+5,y+5)
12       id_after=screen.after(5,move,screen,\
13                                t,xsignal,ysignal)
```

oscilloSpot.py

Tests d'animation du spot électronique

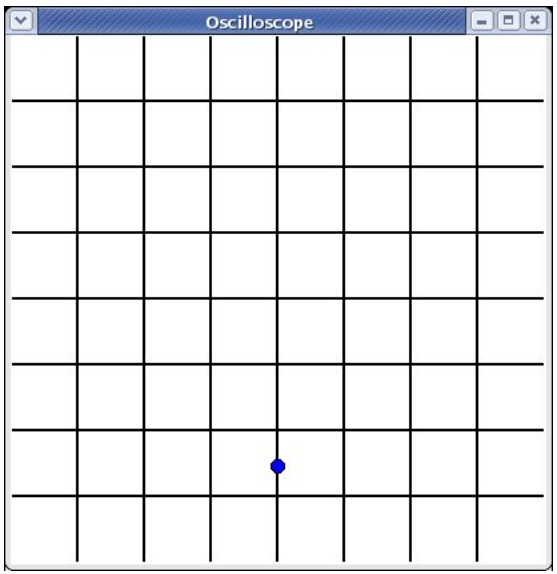
```
1 #### Globals
2 timebase=0
3 #### Simulation initialisation
4 oscilloGrid(screen)
5 spot=createSpot(screen,1,"blue")
6 move(screen,0,xsignal,ysignal)
```



oscilloSpot.py

Tests d'animation du spot électronique

```
1 #### Globals
2 timebase=0
3 #### Simulation initialisation
4 oscilloGrid(screen)
5 spot=createSpot(screen,1,"blue")
6 move(screen,0,xsignal,ysignal)
```



oscilloTimebase.py

Réglages de l'utilisateur

```
1  def setTimeBase(t):
2      global timebase
3      timebase=float(t)
4  ...
5  scaleTime = Scale(mw,length=screen_w,\
6                    orient=HORIZONTAL,\
7                    label='Time :',\
8                    troughcolor='dark grey',\
9                    sliderlength=40,\
10                   showvalue=1,\
11                   from_=0,to=10,\
12                   tickinterval=1,\
13                   command=setTimeBase)
```

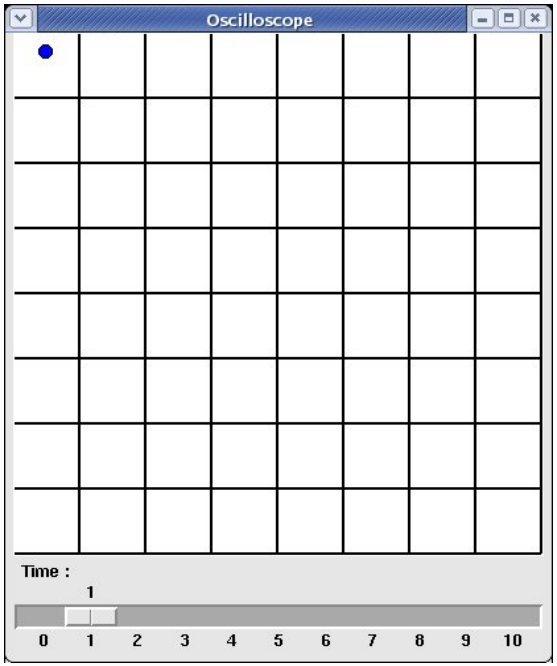
oscilloTimebase.py

Contrôle (time) de la base de temps

```
1  import time
2  def move(screen,t,xsignal,ysignal) :
3      global start_time,stop_time
4        ...
5      if x<=screen_w : t=t+20
6      else :
7          stop_time=time.time()
8          print "time" ,stop_time-start_time
9          start_time=time.time()
10         ...
11     screen.coords(beam,x-5,y-5,x+5,y+5)
12     id_after=screen.after(5,move,screen,\
13                           t,xsignal,ysignal)
```

oscilloTimebase.py

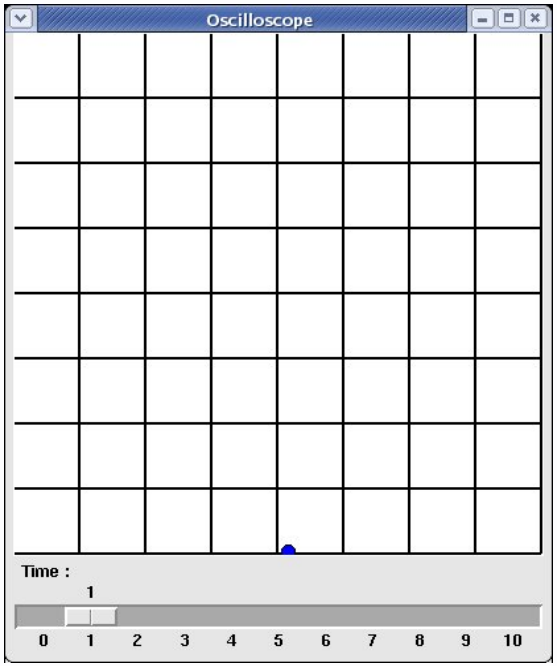
Gestion de la base de temps




```
1 ...
2 time 1.04870295525
```

oscilloTimebase.py

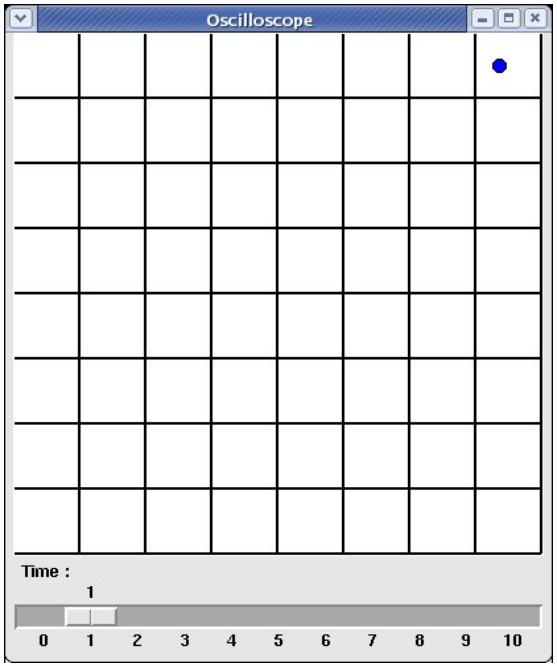
Gestion de la base de temps




```
1 ...
2 time 1.04870295525
```

oscilloTimebase.py

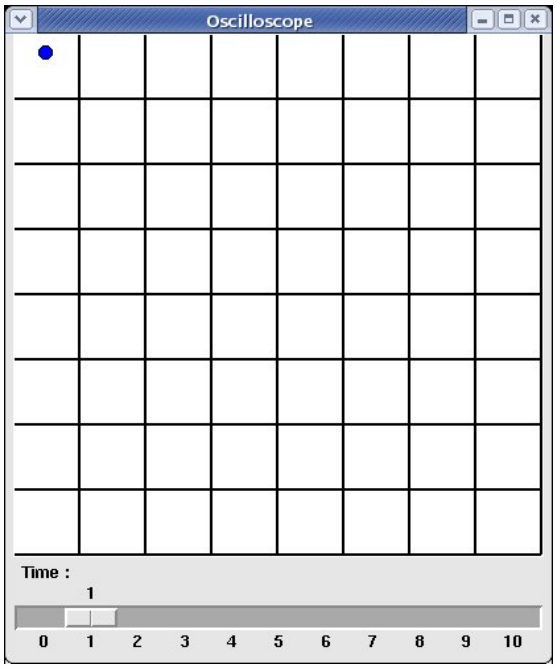
Gestion de la base de temps




```
1 ...
2 time 1.04870295525
```

oscilloTimebase.py

Gestion de la base de temps




```
1  ...
2  time  1.04870295525
3  time  1.04167103767
```

oscilloShowCurve.py

Calcul et création de courbes

```
1  def showXCurve(event) :
2      global ...
3      curve=[]
4      color=event.widget.cget("selectcolor")
5      chek=event.widget.cget("variable")
6      checked=event.widget.getvar(chek)
7      if checked=="0":
8          curve=computeCurve(screen_w,\
9                             screen_h,\
10                            xsignal,\
11                            timebase)
12         id_xcurve=createCurve(screen,\
13                               curve,color)
```

oscilloShowCurve.py

Calcul et création de courbes

```
1         else :
2             if id_xcurve!=-1 :
3                 screen.delete(id_xcurve)
4                 id_xcurve=-1
```

oscilloShowCurve.py

Calcul et création de courbes

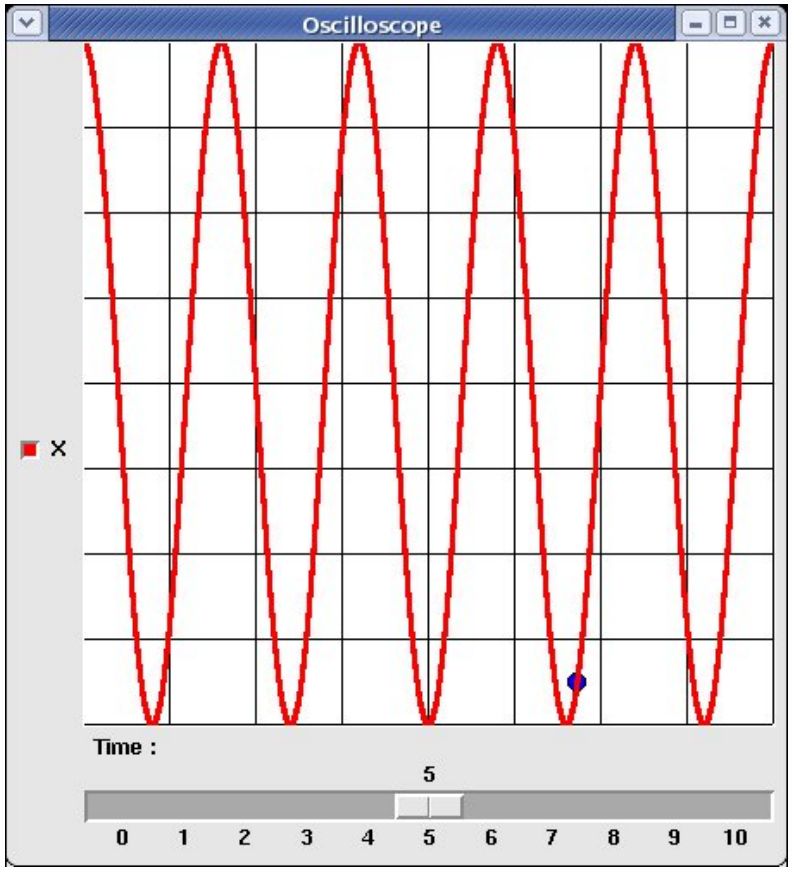
```
1       else :
2           if id_xcurve!=-1 :
3               screen.delete(id_xcurve)
4               id_xcurve=-1
```

Contrôle de l'affichage

```
1  checkX=IntVar()
2  checkX.set("0")
3  checkXCurve=Checkbutton(mw,\
4                          text='X Curve',\
5                          variable=checkX,
6                          selectcolor="red")
7  checkXCurve.bind("<Button-1>",showXCurve)
```

oscilloShowCurve.py

Contrôle de la visualisation



oscilloEvaluateCurve.py

Types de signaux

```
1  signalTypes={"cos":cos,"sin":sin,...}
2  def evaluateX(event):
3      global ...
4      name=event.widget.get()
5      if signalTypes.has_key(name) :
6          xsignal["curve"]=signalTypes[name]
7      else :
8          event.widget.delete(0,END)
9          event.widget.insert(0,"cos")
10         xsignal["curve"]=cos
11     screen.after_cancel(id_after)
12     move(screen,0,xsignal,ysignal)
```

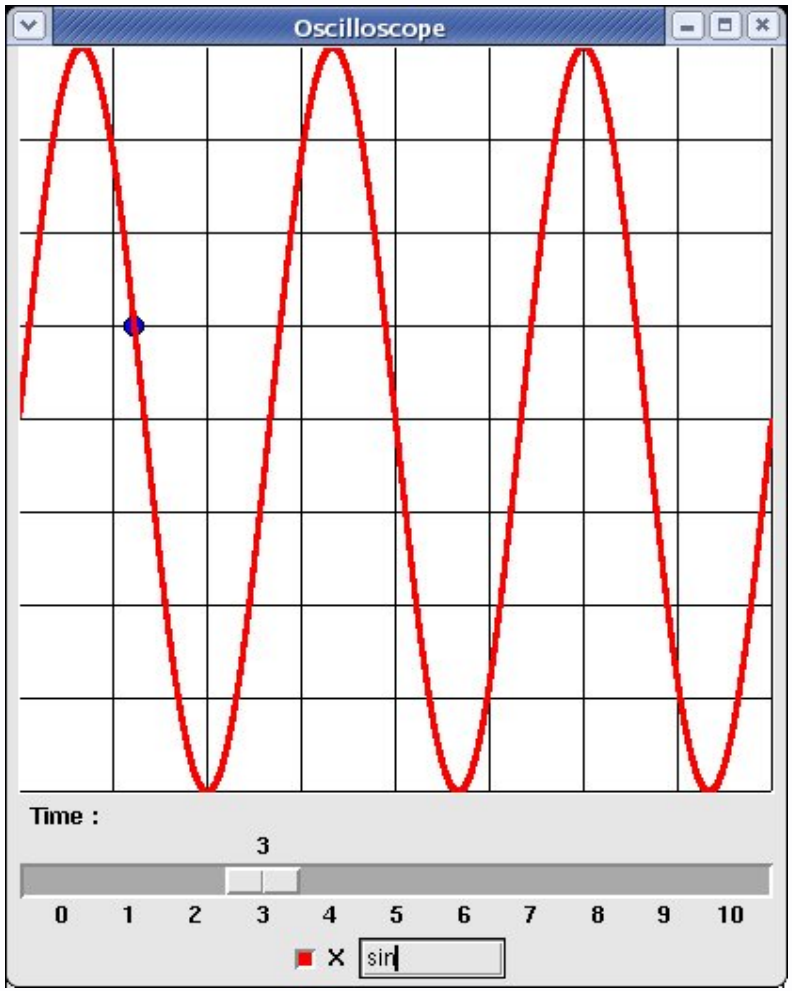
oscilloEvaluateCurve.py

Contrôle du paramétrage de signal

```
1  frameXY=Frame(mw)
2  checkX=IntVar()
3  checkX.set("0")
4  checkXCurve=Checkbutton(frameXY,\
5                          text='X Curve',
6                          variable=checkX,
7                          selectcolor="red")
8  entryX= Entry(frameXY,\
9                width=10,bd=2,\
10               relief=GROOVE)
11 entryX.insert(0,"cos")
```

oscilloEvaluateCurve.py

Tests de paramétrage de signal



oscilloChangeCurveXY.py

Modifications d'amplitude de signal

```
1  def setXMagnitude(x):
2      global step, xsignal
3      xsignal["mag"]=2*float(x)/step
4
5  def setYMagnitude(x):
6      global step, ysignal
7      ysignal["mag"]=2*float(x)/step
```

- amplitude : $[0, 1]$
- nombre de carreaux : *step*
- échelle (`Scale`) : $x \in [0, step/2]$

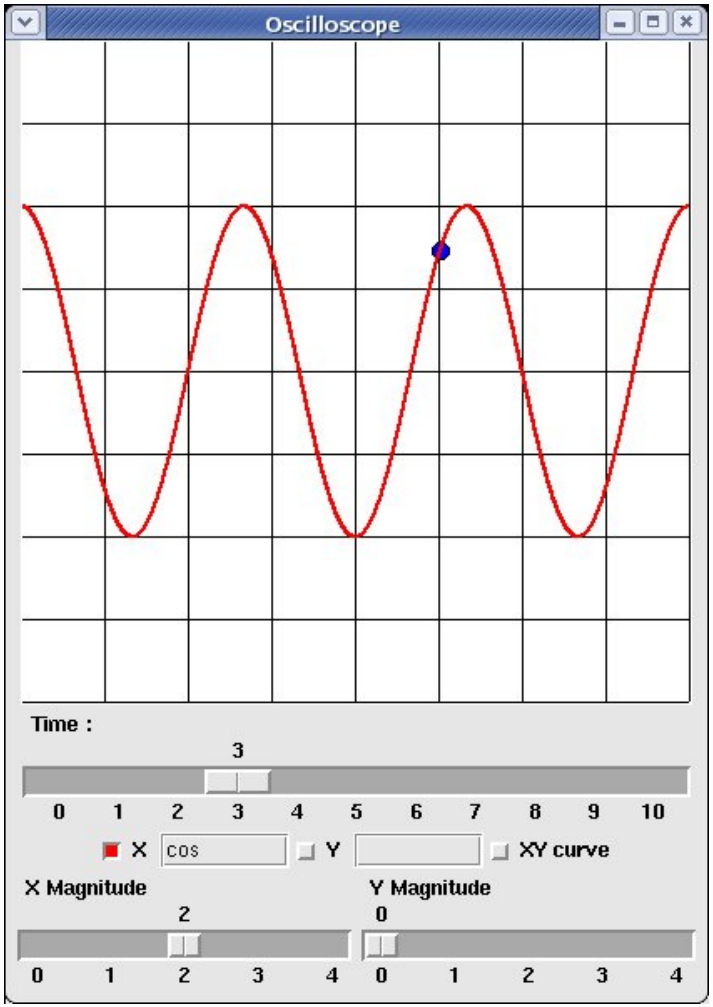
oscilloChangeCurveXY.py

Contrôle de l'amplitude du signal

```
1  frameXYMag=Frame(mw)
2  scaleXMag=Scale(frameXYMag,\
3                  length=screen_w/2,\
4                  orient=HORIZONTAL,\
5                  label='X Magnitude',\
6                  troughcolor='dark grey',\
7                  sliderlength=20,\
8                  showvalue=1,\
9                  from_=0,to=step/2,\
10                 tickinterval=1,\
11                 command=setXMagnitude)
12 scaleYMag=Scale(frameXYMag,\
13                 ...
14                 command=setYMagnitude)
```

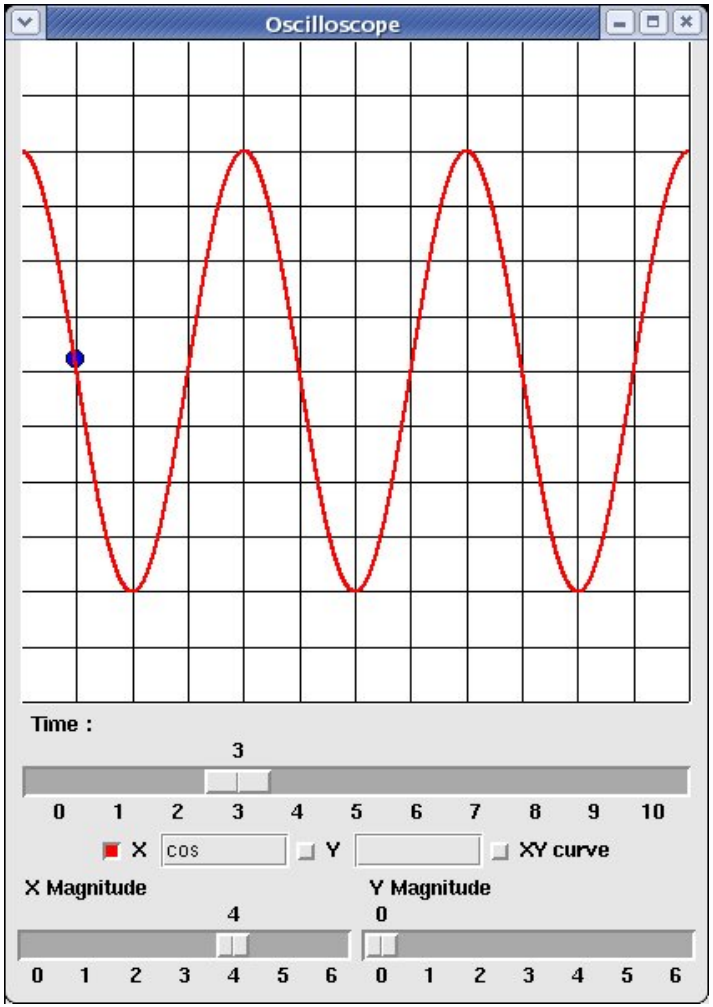
oscilloChangeCurveXY.py

Tests d'amplitude de signal (step=8)



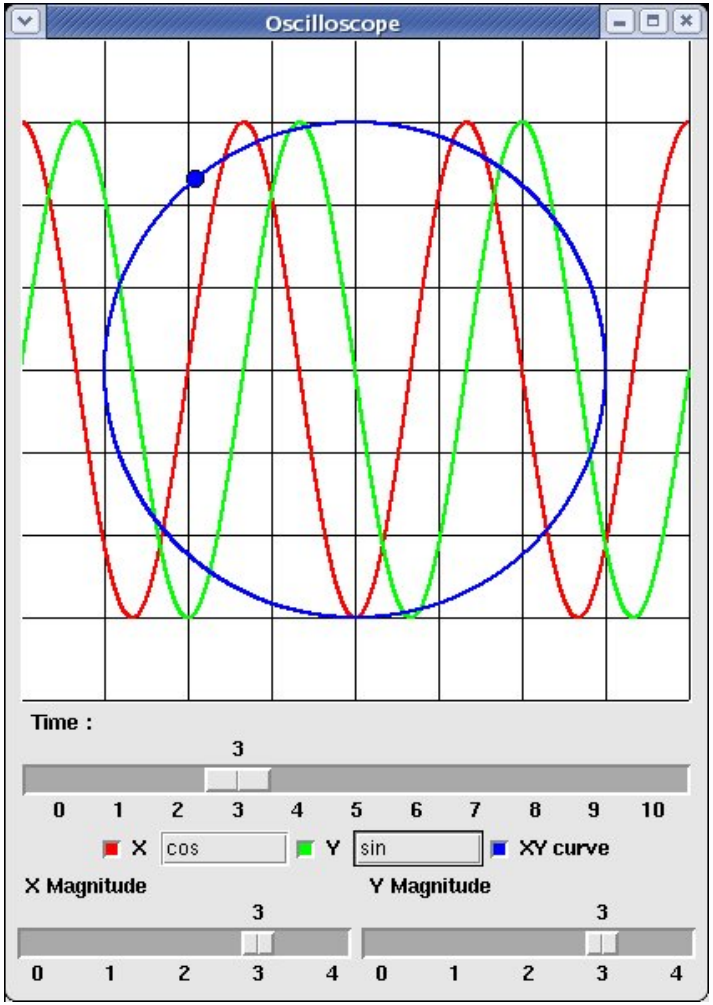
oscilloChangeCurveXY.py

Tests d'amplitude de signal (step=12)



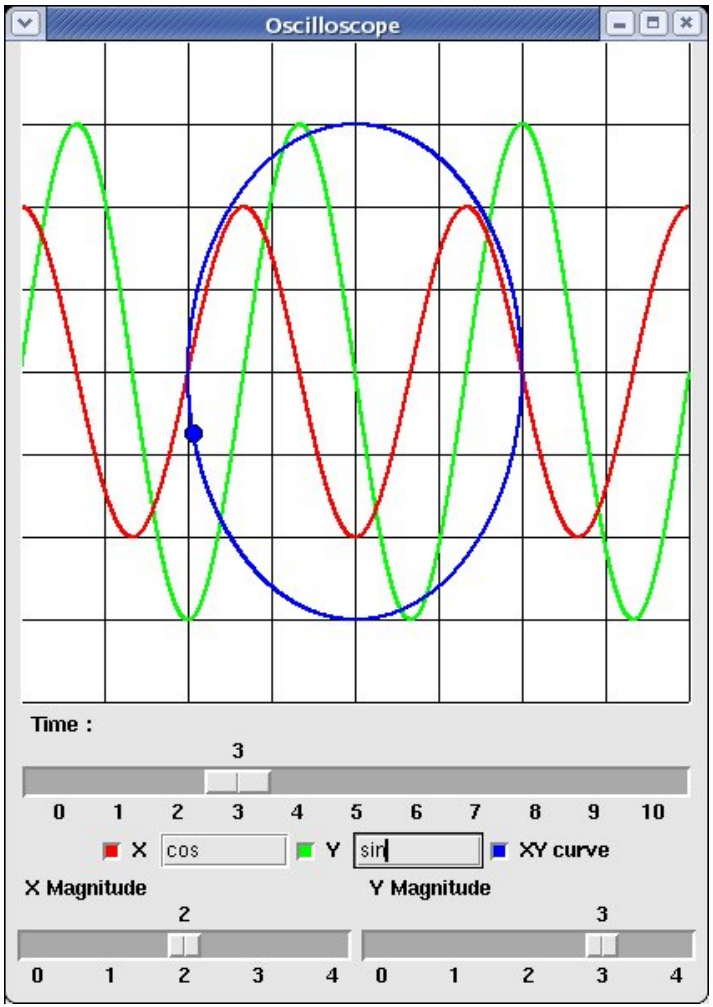
oscilloChangeCurveXY.py

Tests d'amplitude de signal



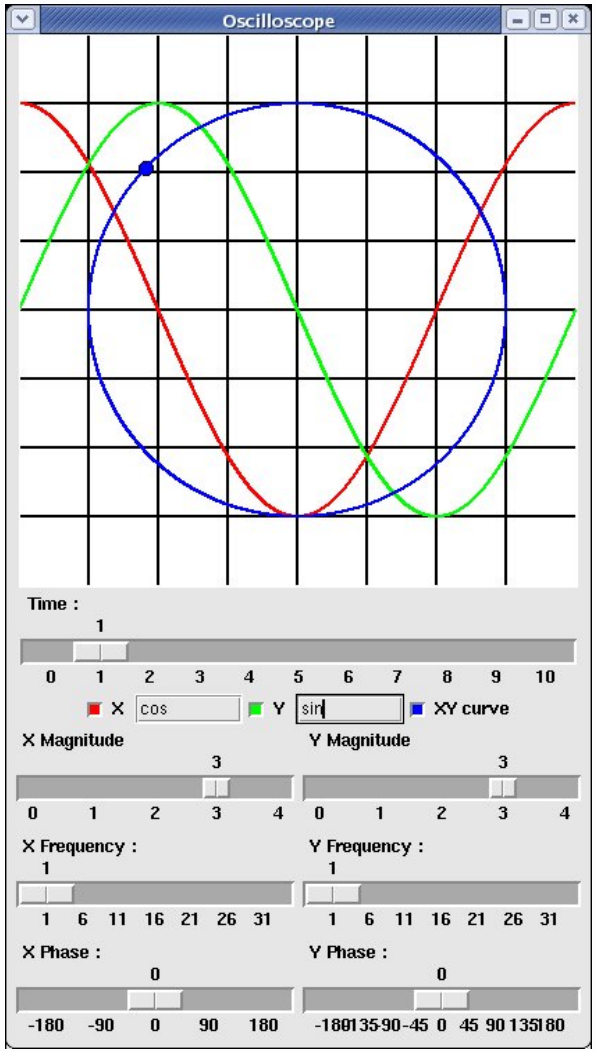
oscilloChangeCurveXY.py

Tests d'amplitude de signal



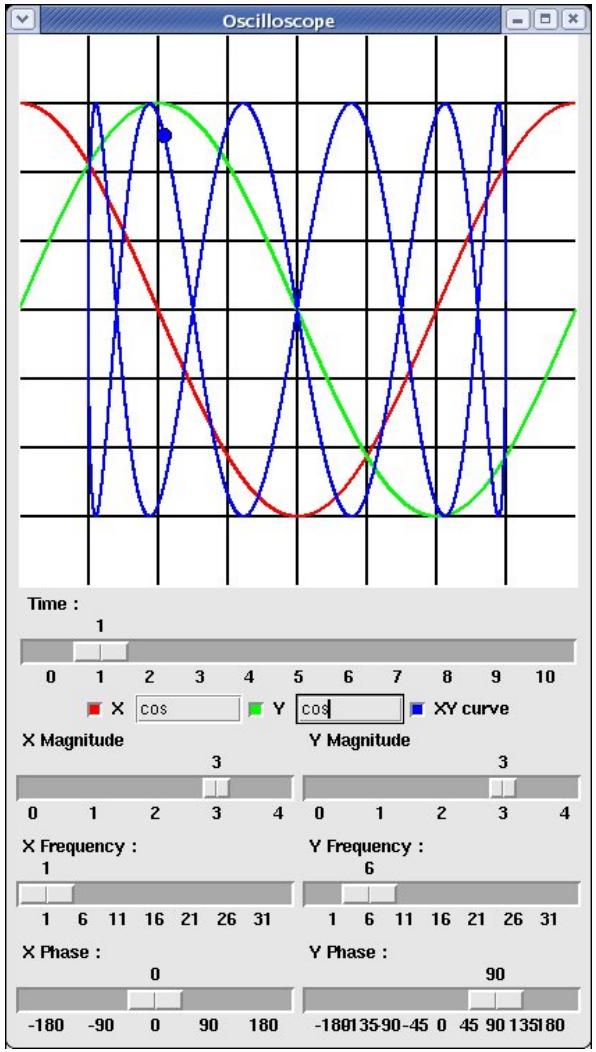
oscilloscope.py

Simulation d'oscilloscope



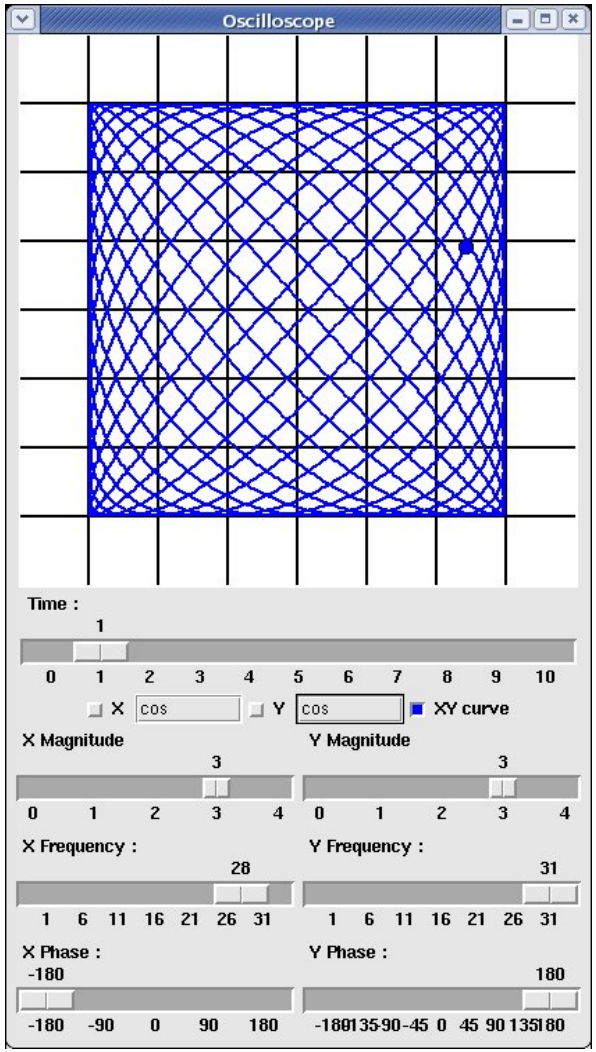
oscilloscope.py

Simulation d'oscilloscope



oscilloscope.py

Simulation d'oscilloscope



Fichier setup.py

```
1  from distutils.core import setup
2  setup(
3          name ='oscillo-simulation',
4          description ='oscilloscope simulation',
5          version='1.0',
6          url='iroise.enib.fr/moodle',
7          author='info1a',
8          author_email='info1a@enib.fr',
9          packages=['oscillo'],
10         package_dir={'oscillo':
11                      'src'}
12         )
```

Test de l'installation

```
# python setup.py install
running install
running build
running build_py
running install_lib
running install_egg_info
...
$ ls -R build
build/:
lib
build/lib:
oscillo
build/lib/oscillo:
create.py  gui.py  __init__.py  signals.py
```

Création de l'archive

```
$ python setup.py sdist

running sdist
reading manifest file 'MANIFEST'
creating oscillo-simulation-1.0
creating oscillo-simulation-1.0/src
making hard links in oscillo-simulation-1.0...
hard linking README.txt -> oscillo-simulation-1.0
...
creating dist
tar -cf dist/oscillo-simulation-1.0.tar oscillo-simulatio
gzip -f9 dist/oscillo-simulation-1.0.tar
removing 'oscillo-simulation-1.0' (and everything under i

$ ls dist
oscilloscope-simulation-1.0.tar.gz
```

Bibliographie

Documents

- Gérard Swinnen :
  “Apprendre à programmer avec Python” (2005)
- Fredrick Lundh :
  “An introduction to Tkinter” (1999)
- John W. Shipman :
  “Tkinter reference : a GUI for Python” (2006)

Adresses “au Net”

- `wiki.python.org/moin/TkInter`
- `fr.wikipedia.org/wiki/Oscilloscope`
- `www.abcelectronique.com/annuaire/cours.php`
- `perso.orange.fr/olivier.granier/electro/fonda.html`