

Edumóvil



Python: Módulo Turtle

M.C. Gabriel Gerónimo Castillo
 Laboratorio Edumóvil
 Universidad Tecnológica de la Mixteca
gceron@gmail.com

Introducción

Turtle es una buena opción para introducir a la programación a los niños, de una forma visual y sencilla.

El módulo Turtle proporciona primitivas gráficas tanto orientadas a objeto como orientada a procedimientos.

Para trabajar con Turtle se debe tener instalada una versión de Python con soporte para Tk.



Laboratorio Edumóvil. M.C. Gabriel Gerónimo C.

Iniciando turtle

Para trabajar con turtle, iniciamos primero el shell de Python e importamos el módulo turtle

```
>> import turtle
```

Si nos marca un error, debemos entonces que tener que instalar tkinter. Ya sea que lo hagamos por medio de **apt-get install** o en forma mas sencilla abriendo el **Ubuntu Software Center** y colocamos python-tk en la búsqueda, aparecerá “**Tkinter - Writing Tk Applications with Python**”, lo seleccionamos e instalamos.

Iniciando turtle

Ya instalado, también podemos invocarlo desde un script colocando lo siguiente en las primeras líneas de dicho script: primero invitamos a Python y luego importamos el módulo turtle

```
#!/usr/bin/python
```

```
import turtle
```

Ya iniciado el módulo, entonces invitamos al canvas para trabajar en el, esto lo hacemos utilizando la función Pen, que automáticamente crea el canvas.

```
>>> t = turtle.Pen()
```

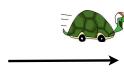
Comandos de movimiento

Mover y Dibujar

forward(distancia) o **fd(distancia)**: Avanzar la tortuga hacia adelante la **distancia** especificada en pixeles, en la dirección de la cabeza de la tortuga.

Ejemplo:

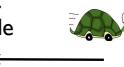
```
>>> turtle.forward (50)
>>> turtle.forward (-100)
```



backward(distancia) o **bk(distancia)** o **back(distancia)**: Mueve la tortuga hacia atrás una distancia, opuesta a la dirección de su cabeza. No cambia la dirección de la cabeza de la tortuga.

Ejemplo:

```
>>> turtle.backward(50)
```



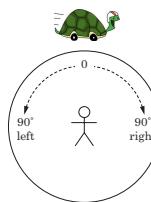
Comandos de movimiento

Mover y Dibujar

right (ángulo) o rt (ángulo): Gira la tortuga hacia la derecha un ángulo de unidades. Las unidades están por default en grados, pero pueden ser cambiadas por medio de las funciones `degrees()` y `radians()`. El ángulo de orientación depende del modo de la tortuga.

Ejemplo:

```
>>> turtle.right (90)
```



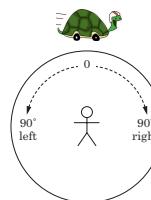
Comandos de movimiento

Mover y Dibujar

left(ángulo) o lt(ángulo): Gira la tortuga hacia la izquierda un ángulo de unidades. Las unidades están por default en grados, pero pueden ser cambiadas por medio de las funciones `degrees()` y `radians()`. El ángulo de orientación depende del modo de la tortuga.

Ejemplo:

```
>>> turtle.left (90)
```



Comandos de movimiento

Mover y Dibujar

goto(x,y | vector) o setpos(x,y | vector) o setposition (x,y | vector): Mueve la tortuga a una posición absoluta, donde **x** y **y** son números enteros o reales. Si la pluma está abajo, dibuja la línea. No realiza el cambio de orientación de la tortuga. **vector** es un par de coordenadas.

Ejemplo:

```
>>> pos=turtle.pos()
>>> pos
(0.00,0.00)
>>> turtle.setpos(80,30)
>>> turtle.setpos(pos)
>>>
```



Comandos de movimiento

Mover y Dibujar

setx(x): Coloca a la tortuga en la coordenada **x**, el valor de la segunda coordenada no es cambiada, **x** es un número entero o real.

sety(y): Coloca a la tortuga en la coordenada **y**, el valor de la primera coordenada no es cambiada, **y** es un número entero o real.

Ejemplo:

```
>>> pos=turtle.pos()
>>> pos
(0.00,0.00)
>>> turtle.setx(10)
>>> turtle.position()
(10.00,0.00)
>>> turtle.sety(10)
>>> turtle.position()
(10.00,10.00)
>>>
```



Laboratorio Edumóvil. M.C. Gabriel Gerónimo C.

Comandos de movimiento

Mover y Dibujar

setheading (ángulo) o **seth(ángulo)**: Coloca la orientación de la tortuga a un **ángulo**. Donde **ángulo** es un número entero o real. Algunas direcciones comunes en grados son:

modo estándar	modo logo
0 - Este	0 - Norte
90 - Norte	90 - Este
180 - Oeste	180 - Sur
270 - Sur	270 - Oeste



Comandos de movimiento

Mover y Dibujar

home(): Mueve la tortuga a la posición original (0,0), y fija la cabeza en su orientación inicial

Ejemplo:

```
>>> turtle.heading()
0.0
>>> turtle.setheading(90)
>>> turtle.heading()
90.0
>>> turtle.home()
>>> turtle.heading()
0.0
>>>
```



Comandos de movimiento

Mover y Dibujar

circle (radio, medida=opcional, pasos=opcional): Dibuja un círculo con un radio determinado.

El centro del círculo está en unidades del radio a la izquierda de la tortuga.

medida(un ángulo) determina que parte del círculo es dibujada. Si no se da la **medida**, se dibuja el círculo completo. Si **medida** no está en el círculo, un punto final del arco es la posición actual de la pluma. Dibuja el arco en dirección hacia la izquierda si el radio es positivo, si no en dirección opuesta.



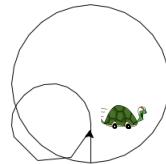
Comandos de movimiento

Mover y Dibujar

Finalmente la dirección de la tortuga es cambiada por la cantidad de **medida**. Como el círculo es aproximado por un polígono regular inscrito, los **pasos** determina el número de pasos a usar. Si no se dan, se calculará de forma automática. Se puede utilizar para dibujar polígonos regulares.

Ejemplo:

```
>>> turtle.circle(100)
>>> turtle.left(90)
>>> turtle.forward(50)
>>> turtle.circle(50,50)
>>> turtle.circle(50,150)
>>> turtle.circle(50,150,3)
>>>
```



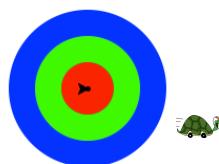
Comandos de movimiento

Mover y Dibujar

dot (tamaño (opcional), color): Dibuja un punto circular con un determinado **tamaño** de diámetro usando un determinado **color**. Si no se le da el **tamaño**, el máximo es pincel+4 y 2*pincel. **tamaño** es opcional, y si se asigna es un entero ≥ 1 , y **color** es una cadena o tupla numérica.

Ejemplo:

```
>>> turtle.dot(150,"blue")
>>> turtle.dot(100,"green")
>>> turtle.dot(50,"red")
>>> turtle.dot()
```



Laboratorio Edumóvil. M.C. Gabriel Gerónimo C.

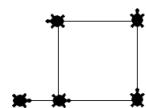
Comandos de movimiento

Mover y Dibujar

stamp (): Estampa una copia de la forma de la tortuga sobre el canvas en la posición actual de la tortuga. Retorna un **stamp_id** que puede ser usado para eliminarlo llamando a **clearstamp (stamp_id)**.

Ejemplo:

```
>>> t=turtle.Turtle()
>>> t.shape("turtle")
>>> id=t.stamp();print id
5
```



Comandos de movimiento

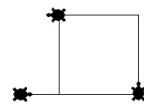
Mover y Dibujar

```
>>> t.forward(50);id2=t.stamp();print id2
6
>>> t.forward(100);t.left(90);id3=t.stamp();print id3
7
>>> t.forward(100);id4=t.stamp();print id4
8
>>> t.left(90);t.forward(100);id5=t.stamp();print id5
9
>>> t.left(90);t.forward(100);t.home()
```



APLICANDO **clearstamp (stamp_id)**.

```
>>> t.clearstamp(6)
>>> t.clearstamp(8)
```



Comandos de movimiento

Mover y Dibujar

clearstamp(stamp_id): Borra la estampa asociada con el identificador **stamp_id**, el cual fue retornado por **stamp()**.

clearstamps(n=opcional): Borra todas o las primeras/ultimas estampas. Si **n** no se coloca, borra todas las estampas, si **n > 0** borra las primeras **n** estampas, si **n < 0** borra las ultimas **n** estampas.

undo(). Deshace (repetidamente) la última(s) acción(es). El número de válido de las acciones a deshacer depende del tamaño del buffer intermedio.

Comandos de movimiento

Mover y Dibujar

sleep(velocidad=opcional): Fija la **velocidad** de movimiento de la tortuga en un rango de 0 a 10, si el argumento no es dado retorna la velocidad actual. También se puede colocar una cadena en **velocidad**, como se indica a continuación:

- 0 es igual a colocar “fastest”
- 10 es igual a colocar “fast”
- 6 es igual a colocar “normal”
- 3 es igual a colocar “slow”
- 1 es igual a colocar “slowest”

Si colocamos 0 en **velocidad**, no existe animación cuando se mueve la tortuga, esta se vera que ejecuta la instrucción y un salto al punto final.

Comandos de movimiento

Estado de la Tortuga

position() o **pos()**: Retorna la posición (x,y) actual de la tortuga, como un vector 2d.

towards(x,y | vector): Retorna el ángulo entre las líneas de la posición actual de la tortuga y la posición especificada por **(x,y)** , o el **vector** u otra tortuga.

xcor(): Retorna la coordenada en x de la tortuga.

ycor(): Retorna la coordenada en y de la tortuga.

heading(): Retorna el ángulo actual de la cabeza de la tortuga (el valor depende del modo).

distance(x,y | vector): Retorna la distancia de la tortuga al punto **(x,y)** o a otra tortuga en unidades de pasos.

Comandos de movimiento

Ajustar y Medir

degrees(): Fija las unidades de medida de los ángulos, es decir, el número de grados para un círculo completo. El valor default es 360 grados.

radians(): Fija la unidad de medida de los ángulos en radianes, el equivalente a grados es: $2 * \pi$.

Comandos de control de la pluma

Dibujar estado

pendown() o **pd()** o **down()**: Coloca la pluma hacia abajo, es decir, dibuja al movimiento de la tortuga.

penup() o **pu()** o **up()**: Coloca la pluma hacia arriba, es decir, no dibuja al movimiento de la tortuga.

pensize (ancho=opcional) o **width (ancho=opcional)**: Ajusta el grosor de la línea al número dado en **ancho**, si este aparece, en caso de no colocar ancho, devuelve el número (positivo) de grosor. Si Resizemode se establece en "auto" y turtleshape es un polígono, dicho polígono se dibuja con el mismo grosor de la línea.

isdown() : Retorna **True** si la pluma está abajo, **False** si está arriba.

Comandos de control de la pluma

Dibujar estado

pen(pluma=opcional, **lista_arg): Retorna o fija atributos de la pluma en un "diccionario" con los siguientes datos:

- "shown": **True/False**
- "pendown": **True/False**
- "pencolor": Cadena o tupla
- "fillcolor": Cadena o tupla
- "pensize": Número positivo
- "speed": Número en el rango de 0 a 10
- "resizemode": "auto" o "user" o "noresize"
- "stretchfactor": Par de números positivos de la forma (x,y)
- "outline": Número positivo
- "tilt": Número

Comandos de control de la pluma

Dibujar estado

Este diccionario puede ser utilizado como argumento para un llamado posterior a **pen()** para restaurar el estado anterior de la pluma. Por otra parte uno o mas atributos pueden presentarse como palabras claves de argumentos. Estos se pueden utilizar para configurar varios atributos de la pluma en un determinado estado.

Comandos de control de la pluma

Control de color

pencolor(*argumentos): Retorna o fija el color de la pluma. Cuatro formas de llamado son permitidas:

1. **pencolor()**: Retorna el actual **pencolor()** como una cadena donde se especifica el color. Esta salida puede ser usada como entrada a otro llamado como: **color**, **pencolor**, **fillcolor**.

Ejemplo:

```
>>> turtle.pencolor()
'black'
>>>
```

Comandos de control de la pluma

Control de color

2. **pencolor (cadena)**: Fija el color de la pluma a **cadena**, el cual es un color de Tk especificado en dicha cadena, tal como "red", "yellow" o "#NumHexadecimal"

3. **pencolor ((r, g, b))**: Fija el color de la pluma formado por la tripleta (r,g,b). Cada r, g y b debe estar en el rango de 0 a colormode, donde el colormode esta en el rango de 1.0 a 255.

4. **pencolor (r, g, b)**: Fija el color de la pluma en formato RGB representado por r(red), g (green), b (blue). Cada r, g y b debe estar en el rango de 0 a colormode.

Comandos de control de la pluma

Control de color

fillcolor(*argumentos): Retorna o fija el color de relleno. Son permitidos cuatro formatos de entrada:

1. **fillcolor()**: Retorna el actual color de relleno como una cadena. Puede ser usada como entrada a otro llamado de color/pencolor/fillcolor.

2. **fillcolor(colorstring)**: Fija el color de relleno a **colorstring**, la cual es una cadena de tipo Tk color, tal como:"red", "yellow" o "#45cc8d"

3. **fillcolor((r,g,b))**: Fija el color de relleno a color RGB, representado por r,g y b. Cada uno de ellos en un rango de 0 a colormode, donde el colormode podrá ser de 1.0 a 255.

4. **fillcolor(r,g,b)**: Fija el color de relleno a color RGB, representado por r,g y b. Cada uno de ellos en un rango de 0 a colormode.

Comandos de control de la pluma

Control de color

color(*argumentos): Retorna o fija el color de la pluma y el relleno. Varias entradas son permitidas, ellas usan de 0 a 3 argumentos como sigue:

color(): Retorna el actual color de la pluma y del relleno como un par de cadena o tuplas como lo retorna pencolor() y fillcolor().

color(colorstring), color((r,g,b)), color(r,g,b): Entradas como en pencolor, fija pencolor y fillcolor().

color(colorstring1, colorstring2), color((r1,g1,b1), (r2,g2,b2)): Equivalente a pencolor(colorstring), y fillcolor (colorstring2), y análogamente si otros formatos de entradas son usados.

Comandos de control de la pluma

Relleno

fill(banderas): El argumento **bandera** puede ser **True/False** o 1/0 respectivamente. Se debe colocar **fill(True)** antes de dibujar la forma que se quiera llenar, y **fill(False)** para aplicar el relleno. Cuando se utiliza sin argumento se retorna el estado del relleno, True si está lleno y False en caso contrario.

Ejemplo:

```
>>> import turtle
>>> turtle.fill(True)
>>> for _ in range (3): turtle.forward(150);turtle.left(120)
...
>>> turtle.fill(False)
>>>
```



Comandos de control de la pluma

Relleno

begin_fill(): Iniciar bloque para aplicar relleno, equivalente a **fill(True)**.

end_fill(): Finalizar bloque de aplicar relleno, después de aplicar **begin_fill()**. Equivalente a **fill(False)**.

Ejemplo:

```
>>> turtle.color("black","red")
>>> turtle.begin_fill()
>>> turtle.circle(80)
>>> turtle.end_fill()
>>>
```



Comandos de control de la pluma

Otros comandos

reset(): Borra los dibujos hechos por la tortuga, centra nuevamente la tortuga, y reinicia las variables al valor por default.

clear(): Borra los dibujos hechos por la tortuga. No mueve la posición actual de la tortuga, y cambia el estado, así como los dibujos de otras tortugas no son afectados.

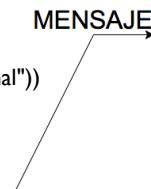
Comandos de control de la pluma

Otros comandos

write(arg, move, align, font): Escribe un texto, la cadena asignada en **arg** en la posición actual de la tortuga, con alineación **align** ya sea “left”, “center” o “right” y con el **font** dado. **font** es una triple de la forma (nombre del font, tamaño del font, tipo del font). Por otra parte de **move** es True, el pincel se mueve a la esquina inferior derecha del texto, por default **move** es False.

Ejemplo:

```
>>> turtle.goto(100,200)
>>> turtle.write("MENSAJE",True,"center",("Arial", 32, "normal"))
>>>
```



Estado de la tortuga

Visible

showturtle() o **st()**: Hacer la tortuga invisible, es una buena idea cuando se está realizando algún dibujo complejo, ya que acelera el dibujo a observar.

hideturtle() o **ht()**: Hace que la tortuga sea visible.

isvisible(): Retorna **True** si la tortuga es mostrada, **False** si está no es mostrada.

Estado de la tortuga

Apariencia

shape(): Fija la forma de la tortuga pasando como parámetro el nombre de dicha forma, si el nombre no es dado retorna el nombre de la actual forma. El nombre de la forma debe existir en el diccionario de formas de TurtleScreen. Inicialmente existen las siguientes formas poligonales: "arrow", "turtle", "circle", "square", "triangle", "classic".

Ejemplo:

```
>>> turtle.shape()
'classic'
>>> turtle.shape("turtle")
>>> turtle.shape()
'turtle'
>>>
```

Forma clásica



Forma tortuga



Estado de la tortuga

Apariencia

resizemode(rmodo=opcional): Fija el resizemode a uno de los siguientes valores: "auto", "user", "noresize". Si el **rmodo** no es dado retorna el actual resizemode. Los siguientes resizemode tienen los siguientes efectos:

auto: Adopta la apariencia de la tortuga correspondiente a el valor de pensize.

user: Adopta la apariencia de la tortuga acorde al valor de stretchfactor y outlinewidth (outline), las cuales son fijadas por shapesize().

noresize: No hace cambios, y se queda en la apariencia actual.

Estado de la tortuga

Apariencia

shapesize(stretch_wid=opcional, stretch_len=opcional, outline=opcional) o
turtlesize(stretch_wid=opcional, stretch_len=opcional, outline=opcional):

Retorna o fija los atributos del pincel factores stretch y/o outline. Fija el resizemode a "user" Si y sólo si resizemode es colocado a "user", la tortuga será mostrada acorde a sus stretchfactors: stretch_wid es stretchfactor perpendicular a su orientación, stretch_len es stretch factor en dirección a su orientación, outline determina el ancho de la forma de salida de la linea.

Ejemplo:

```
>>> turtle.shapesize()
(1, 1, 1)
>>> turtle.resizemode("user")
>>> turtle.shapesize(4,4,5)
>>> turtle.shapesize()
(4, 4, 5)
>>> turtle.shapesize(outline=8)
>>> turtle.shapesize()
(4, 4, 8)
>>>
```



Estado de la tortuga

Apariencia

tilt(ángulo): Rota la forma de la tortuga un determinado ángulo, desde su actual ángulo de inclinación, pero no cambia la dirección de pintado o de desplazamiento.

Ejemplo:

```
>>> turtle.tilt(30)
>>> turtle.forward(50)
>>> turtle.tilt(30)
>>> turtle.forward(150)
>>>
```



Estado de la tortuga

Apariencia

settiltangle(ángulo): Rota la forma de la tortuga a un punto en la dirección especificada por **ángulo**, independientemente de su ángulo de inclinación actual. No cambia la dirección de la cabeza de la tortuga (dirección de movimiento).

tiltangle(): Retorna el actual ángulo de inclinación, es decir, el ángulo entre la orientación de la forma de la tortuga y la cabeza de la tortuga (su dirección de movimiento).



Eventos y métodos especiales

Eventos

onclick(función, botón=1, add=opcional): Enlaza la **función** al evento click del mouse sobre la tortuga, si en lugar de función se coloca **None** el enlace es removido.

Dicha **función** debe ser definida con dos argumentos que son las coordenadas del punto donde se da el click del mouse (se toman automáticamente dichas coordenadas al dar el click).

El número que se coloca en **botón** es por default 1 (botón izquierdo del mouse).

En **add**, si se coloca, se aplica **True** o **False**, si es **True** un nuevo enlace se anexa, en otro caso se reemplaza un nuevo enlace.

Eventos y métodos especiales

Eventos

Ejemplo:

```
>>> def tu(x,y): turtle.left(180);turtle.forward(100)
...
>>> turtle.onclick(tu)
>>> turtle.onclick(None)
>>>
```

onrelease(función, botón=1, add=opcional): Similar a la función anterior, pero ahora con el botón de liberación del mouse.

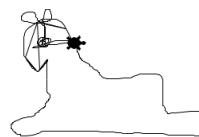
Eventos y métodos especiales

Eventos

ondrag(función, botón=1, add=opcional): Parámetros similares a los comandos anteriores. Solo que este comando enlaza la **función** con el evento de movimiento del mouse. Si se coloca None se remueve el enlace.

Ejemplo:

```
>>> turtle.ondrag(turtle.goto)
>>>
```



Si sostenemos el click del mouse y arrastramos la tortuga esta se moverá en la pantalla produciendo dibujos a mano alzada.

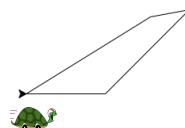
mainloop() o **done()**: Inicia el evento de Ciclo, realiza la ultima instrucción realizada.

Eventos y métodos especiales

Métodos especiales

begin_poly(): Inicia a grabar los vértices de un polígono. La posición de la tortuga actual es el primer vértice.

end_poly(): Para la grabación de los vértices de un polígono. La actual posición de la tortuga es el último vértice. Este debe ser conectado con el primero.



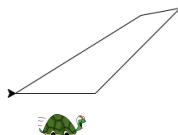
Eventos y métodos especiales

Métodos especiales

get_poly(): Retorna la última grabación de un polígono.

Ejemplo:

```
>>> turtle.begin_poly()
>>> turtle.fd(100)
>>> turtle.left(45)
>>> turtle.fd(150)
>>> turtle.left(145)
>>> turtle.fd(50)
>>> turtle.home()
>>> turtle.end_poly()
>>> p=turtle.get_poly()
>>> p
((0.00,0.00), (100.00,0.00), (206.07,106.07), (156.83,97.38), (0.00,0.00))
>>>
```



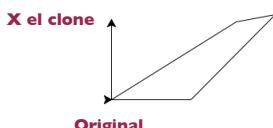
Eventos y métodos especiales

Métodos especiales

clone(): Crea y retorna un clon de la tortuga, con la misma posición, rumbo y propiedades de la original.

Ejemplo:

```
>>> x=turtle.clone()
>>> x.left(90)
>>> x.forward(100)
>>>
```



Eventos y métodos especiales

Métodos especiales

getturtle() o **getpen()**: Retorna la dirección donde se encuentre localizado el objeto tortuga.

Ejemplo:

```
>>> turtle.getturtle()
<turtle.Turtle object at 0x10116a150>
>>>
```

getscreen(): Retorna el objeto sobre el cual la tortuga está dibujando. Métodos TurtleScreen pueden ser invocados por este objeto.

Ejemplo:

```
>>> turtle.getscreen()
<turtle._Screen object at 0x1011507d0>
>>>
```

Eventos y métodos especiales

Métodos especiales

setundobuffer(): Fija o desactiva el buffer de deshacer lo borrado. Si se coloca un entero como parametro indica que el numero máximo de acciones que guardará para recuperar lo borrado. Si se coloca None entonces se desactiva el buffer de deshacer.

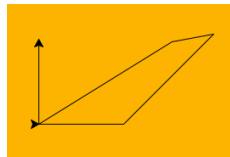
undobufferentries(): Retorna el número de entradas del buffer de deshacer.

Métodos de pantalla

Control de ventana

bgcolor(*arg): Fija o retorna el color de fondo de la pantalla. En **arg** se coloca una cadena del color o tres números en el rango de 0 a colormode, o una tripleta de tales números.

Ejemplo:
>>> turtle.bgcolor("orange")
>>>



Métodos de pantalla

Control de ventana

bgpic(imagen=opcional): Coloca/retorna una imagen de fondo de pantalla. Si se coloca “nopic” se borra la imagen de fondo. Si no se coloca nada como argumento se retorna el nombre de la imagen actual.

```
Ejemplo:  
>>> turtle.bgpic("bosque.gif")  
>>> turtle.bgpic()  
'bosque.gif'  
>>>
```



Métodos de pantalla

Control de ventana

clearscreen(): Borra todos los dibujos de todas las tortugas de la pantalla. Reinicia el estado de la pantalla: color blanco de fondo, no imagen de fondo, no eventos enlazados y trazado iniciado.

reset() o resetscreen(): Reinicia todas las tortugas en la pantalla a su estado inicial.

Laboratorio Edumóvil. M.C. Gabriel Gerónimo C.

Métodos de pantalla

Control de ventana

screensize(ancho=opc, alto=opc, fondo=opc): **ancho** y **alto** son números enteros para el ancho y alto del canvas en pixeles, **fondo** es una cadena del color o tripleta del nuevo color de fondo. Si no se colocan argumentos, se retorna el actual (ancho, alto) del canvas.

Ejemplo:

```
>>> turtle.screensize()
(400, 300)
>>>
```

setworldcoordinates(ix,ly,ux,uy): Cambia el sistema de coordenadas que el usuario considere. Donde **ix,ly** son las coordenadas de la esquina inferior izquierda del canvas, y **ux,uy** son las coordenadas de la esquina superior derecha del canvas.

Laboratorio Edumóvil. M.C. Gabriel Gerónimo C.

Métodos de pantalla

Control de animación

delay(retardo=opc): Fija o retorna el retardo de dibujo en milisegundos.

Ejemplo:

```
>>> turtle.delay()
10
>>> turtle.delay(50)
>>> turtle.reset()
>>> for _ in range (8): turtle.left (45); turtle.forward(2)
...
>>>
```

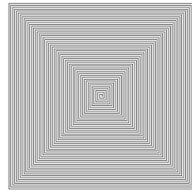
Métodos de pantalla

Control de animación

tracer(n=opc, retardo=opc): Gira la animación on/off de la tortuga y fija el retardo para actualizar el dibujo. Si **n** es dado, solo cada **n-veces** regulares actualiza la pantalla de ejecución, el segundo parámetro es el retardo.

Ejemplo:

```
>>> turtle.reset()
>>> turtle.tracer(8,50)
>>> d=2
>>> for i in range(200): turtle.fd(d);turtle.lt(90);d+=2
...
>>> turtle.tracer()
8
>>>
```



update(): Realiza una actualización de la pantalla. Debe ser usado cuando el trazado está apagado

Métodos de pantalla

Eventos de pantalla

listen(x=opc,y=opc) : Establece el foco en la pantalla para recopilar eventos. Los argumentos **x** y **y** son proporcionados en orden para que listen() este pendiente al método onclick.

onkey(funcion, tecla) : Enlaza una **funcion** al evento de liberación de **tecla**. Si no se coloca argumento o None, el evento es enlazado es removido.

Ejemplo:

```
>>> turtle.listen()
>>> def x(): turtle.fd(50);turtle.rt(50)
...
>>> turtle.onkey(x,"Up")
```

Métodos de pantalla

Eventos de pantalla

onclick(función, botón=1, add=opc): Enlaza una **función** al evento del click del mouse. 1 en **botón** es el default del botón izquierdo del mouse. Con True **add** en se enlaza un nuevo evento, sino se reemplaza la nueva función.

onscreenclick(función, botón=1, add=opc): Semejante al anterior comando.

ontimer(función, t=0): Instala un timer que llama a la **función** (sin argumentos) después de **t** milisegundos, donde **t** es número mayor o igual a 0.

Métodos de pantalla

Métodos especiales

mode(modo=opc): Fija el modo de la tortuga (los cuales pueden ser, "standard","logo", o "world"), y ejecuta un reinicio. Si **modo** no es dado se retorna el modo actual. El modo "standard" es compatible con el modo anterior de turtle, el modo "logo" es compatible con los gráficos de Logo turtle, el modo "world" es usado para que el usuario defina "coordenadas del mundo"

colormode(cmudo=opc): Retorna el modo de color o fija este de 1.0 a 255. Subsecuentemente la tripleta r,g,b pueden ser asignadas y están en el rango de 0 a cmode.

Métodos de pantalla

Métodos especiales

getcanvas(): Retorna el canvas asociado a la pantalla.

Ejemplo:

```
>>> turtle.getcanvas()
<turtle.ScrolledCanvas instance at 0x1004e5320>
>>>
```

getshapes(): Retorna la lista de nombres de las actuales formas de la tortuga.

Ejemplo:

```
>>> turtle.getshapes()
['arrow', 'blank', 'circle', 'classic', 'square', 'triangle', 'turtle']
>>>
```

Métodos de pantalla

Métodos especiales

register_shape(name, shape=opc) o **addshape(name, shape=opc)**: Hay de formas de invocar a esta función:

1. **name** es el nombre del archivo gif, y **shape** será None o no se coloca, así se anexa a la lista de formas la imagen del archivo (Nota: La imagen no rota).

Ejemplo:

```
>>> turtle.addshape("raton.gif")
>>> turtle.shape("raton.gif")
>>> turtle.fd(100)
>>>
```



Métodos de pantalla

Métodos especiales

2. **name** es una cadena arbitraria y **shape** son pares de coordenadas, para instalar la correspondiente forma del polígono.

Ejemplo:

```
>>> turtle.register_shape("x",((5,-3),(0,5),(-5,-3)))
>>> turtle.getshapes()
['arrow', 'blank', 'circle', 'classic', 'raton.gif', 'square', 'triangle', 'turtle', 'x']
>>> turtle.shape("x")
>>>
```

3. **name** es una cadena arbitraria y **shape** es un objeto compuesto.



Métodos de pantalla

Métodos especiales

turtles(): Retorna la lista de tortugas en la pantalla.

Ejemplo:

```
>>> turtle.turtles()
[<turtle.Turtle object at 0x10116a150>]
>>> t=turtle.shape("turtle") ➔ → ⚡
>>> t=turtle.Pen()
>>> turtle.turtles()
[<turtle.Turtle object at 0x10116a150>, <turtle.Turtle object at
0x1007ae850>]
>>> for turtle in turtle.turtles(): turtle.color("red")
```

Métodos de pantalla

Métodos especiales

window_height(): Retorna el alto de la ventana de la tortuga.

window_width(): Retorna el ancho de la ventana de la tortuga.

bye(): Cierra la ventana del gráfico de la tortuga.

exitonclick() : Enlaza el método **bye()** al click del mouse sobre la pantalla.

title("cadena"): Coloca el título de la **cadena** a la ventana de la tortuga.

Ejemplo:

```
>>> turtle.title("Tortuga de GERO")
>>>
```

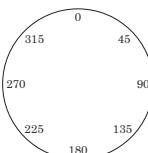
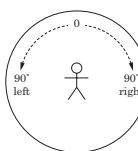
Conceptos

Un **pixel** es un simple punto en la pantalla, el elemento mas pequeño que se puede representar.

Si quieres aprender sobre grados, imagina que estas parado en el centro de un círculo, de esta forma:

1. A la dirección donde estas viendo son 0 grados
2. A tu mano izquierda extendida son 90 grados a la izquierda
3. A tu mano derecha extendida son 90 grados a la derecha
4. Debajo de ti, son 180 grados.

Los grados van de 0 a 360, y puedes moverte para alcanzarlos de izquierda a derecha en contra de las manecillas de un reloj o a favor de las manecilla del reloj de derecha a izquierda.

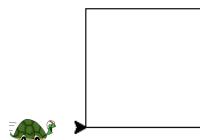


Laboratorio Edumóvil. M.C. Gabriel Gerónimo C.

Dibujando un cuadrado

Para dibujar un cuadrado utilizamos las instrucciones **forward()** y **left()** o **right()** según deseemos que gire la cabeza de la tortuga.

```
>>> turtle.forward(100)
>>> turtle.left(90)
>>> turtle.forward(100)
>>> turtle.left(90)
>>> turtle.forward(100)
>>> turtle.left(90)
>>> turtle.forward(100)
>>> turtle.left(90)
```



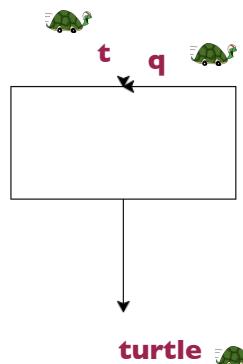
Reduciendo el código para dibujar el cuadro

```
>>> for i in range (1,5): turtle.forward(100);turtle.left(90)
...
>>>
```

Laboratorio Edumóvil. M.C. Gabriel Gerónimo C.

Tres tortugas en acción

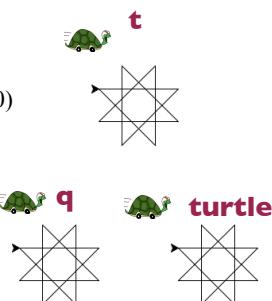
```
>>> import turtle
>>> t=turtle.Pen()
>>> q=turtle.Pen()
>>> t.forward(100)
>>> q.backward(100)
>>> t.left(90)
>>> q.right(90)
>>> t.forward(100)
>>> q.backward(100)
>>> t.left(90)
>>> q.right(90)
>>> t.forward(100)
>>> q.backward(100)
>>> t.left(90)
>>> turtle.right(90)
>>> turtle.forward(100)
>>>
```



Laboratorio Edumóvil. M.C. Gabriel Gerónimo C.

Dibujando estrellas

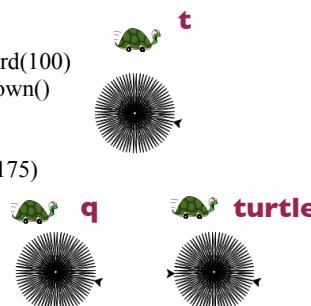
```
>>> t.reset();q.reset();turtle.reset()
>>> q.up();t.up()
>>> t.left(90);t.forward(200);t.left(90);t.forward(100)
>>> t.left(180);q.backward(200);q.down();t.down()
>>> for i in range (1,9):
    t.forward(100);t.left(225);q.forward(100)
    q.left(225);turtle.forward(100);turtle.left(225)
...
>>>
```



Con código similar, solo variando el ángulo y la cantidad de repeticiones puedes crear diferentes tipos de estrellas

Dibujando estrellas

```
>>> t=turtle.Pen()
>>> q=turtle.Pen()
>>> t.reset();q.reset();turtle.reset()
>>> q.up();t.up()
>>> t.left(90);t.forward(200);t.left(90);t.forward(100)
>>> t.left(180);q.backward(200);q.down();t.down()
>>> for i in range (1,76):
    t.forward(100);t.left(175);q.forward(100)
    q.left(175);turtle.forward(100);turtle.left(175)
...
>>>
```



Laboratorio Edumóvil. M.C. Gabriel Gerónimo C.

Funciones

Función de circulo con color

```
def circulo(rojo, verde, azul, radio)
    t.color (rojo, verde, azul)
    t.begin_fill()
    t.circle(radio)
    t.end_fill()
```

invocando a la función circulo

```
circulo(0, 1, 0, 10)
circulo(0.5, 1, 0, 50)
```

invocado a la función cuadro

```
cuadro(50)
```

Función de cuadro

```
def cuadro(tam)
    for i in range (1,5):
        t.forward(tam)
        t.left(90)
```

* Realizar una función para cuadros rellenos de un color *

Funciones

Función de estrella con color

```
def estrella(tam, con):
    if con == True:
        t.begin_fill()
    for x in range (1, 19):
        t.forward(tam)
        if x % 2 == 0:
            t.left(175)
        else:
            t.left(225)
    if con == True:
        t.end_fill()
```

Invocando a la función estrella

```
t.color(0, 0.5, 0)
estrella(120, True)
```

Laboratorio Edumóvil. M.C. Gabriel Gerónimo C.

Tabla de colores (r,g,b)

Color	RGB Values
Aqua	(0, 255, 255)
Black	(0, 0, 0)
Blue	(0, 0, 255)
Fuchsia	(255, 0, 255)
Gray	(128, 128, 128)
Green	(0, 128, 0)
Lime	(0, 255, 0)
Maroon	(128, 0, 0)
Navy Blue	(0, 0, 128)
Olive	(128, 128, 0)
Purple	(128, 0, 128)
Red	(255, 0, 0)
Silver	(192, 192, 192)
Teal	(0, 128, 128)
White	(255, 255, 255)
Yellow	(255, 255, 0)

Cyan (0,255,255)

Referencias



1. Turtle graphics for Tk - Python v2.7.6 documentation. <http://docs.python.org/2/library/turtle.html> . Consultado: 11/febrero/2014
2. Python for Kids. Jason R. Briggs. No Starch Press, Inc. 2013



Python

M.C. Gabriel Gerónimo Castillo
 Laboratorio Edumóvil
 Universidad Tecnológica de la Mixteca
gcer@outlook.com

Introducción

- Python es un lenguaje interpretado, con tipos de datos dinámicos, multiplataforma (Unix, Linux, Windows, MacOS), funcional, y también orientado a objetos.
- Su página oficial www.python.org
- Es popular para desarrollo de juegos, pero además para crear desde aplicaciones hasta sitios web

Ejecutando python

- Cuando ejecutamos **python** en la línea de comando, se puede ver lo siguiente

```
Python 2.6.5 (r265:79063, Apr 16 2010, 13:09:56)
[GCC 4.4.3] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Podemos observar que tenemos la versión **2.6.5** de Python, y el prompt de su shell es **>>>**, lo cuál indica que está esperando comandos válidos del lenguaje para interpretarlos. Para salir del interprete escribimos `exit()` o presionamos las teclas `ctrl + d`.

Ejecutando python

Podemos escribir código de dos formas:

1. Escribir líneas de código en el interprete y obtener una respuesta del interprete por cada línea introducida, o
2. Escribir el código en un archivo de texto, cambiarle el modo (chmod +x) y ejecutarlo (a esto se le llama ejecutar un script). Recuerde aquí colocar como primera línea la ubicación del interprete, es decir, escribir:

```
#!/usr/bin/python
```

Ejecutando python

- Ejecutando una instrucción en el interprete

```
>>> print "Hola, mundo!"  
Hola, mundo!  
>>> print 'Hola, "mundito", mundo!'  
>>> Hola, "mundito", mundo!
```

- Números

```
>>> 2+3/4+2  
4  
>>> 2+3./4+2  
4.75
```

```
#!/usr/bin/python  
c=1  
while c<=4:  
    print "Hola mundo"  
    c+=1
```

Práctica 1.py

Laboratorio Edumóvil. M.C. Gabriel Gerónimo C.

Ejecutando python

- Números

```
>>> 15%6  
3  
>>> a=100.0; b=2.;p=15.  
>>> (a*b)*(p/100.)  
30.0  
>>> c=1  
>>> while c<=4: print "HOLA";c  
+=1  
....  
hola  
hola  
hola  
hola  
>>>
```

Operator	Description
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
==	Equal to
!=	Not equal to

Tipos de datos

Python 2.6.1 (r261:67515, Jun 24 2010, 21:47:49)

[GCC 4.2.1 (Apple Inc. build 5646)] on darwin

Type "help", "copyright", "credits" or "license" for more information.

>>> e = 23

>>> type(e)

<type 'int'>

>>> c = "holo mundo"

>>> type(c)

<type 'str'>

>>> e = 5.45

>>> type(e)

<type 'float'>

>>>

Operadores aritméticos

Operador	Descripción	Ejemplo
+	Suma	r = 3 + 2 # r es 5
-	Resta	r = 4 - 7 # r es -3

Operador	Descripción	Ejemplo
-	Negación	r = -7 # r es -7
*	Multiplicación	r = 2 * 6 # r es 12
**	Exponente	r = 2 ** 6 # r es 64
/	División	r = 3.5 / 2 # r es 1.75
//	División entera	r = 3.5 // 2 # r es 1.0
%	Módulo	r = 7 % 2 # r es 1



Operadores a nivel de bit

Operador	Descripción	Ejemplo
&	and	r = 3 & 2 # r es 2
	or	r = 3 2 # r es 3
^	xor	r = 3 ^ 2 # r es 1
~	not	r = ~3 # r es -4
<<	Desplazamiento izq.	r = 3 << 1 # r es 6
>>	Desplazamiento der.	r = 3 >> 1 # r es 1



Cadenas

```
>>> print "hola mundo
\n\n"
hola mundo

>>> a = "uno"
>>> b = "dos"
>>> c=a + b
>>> print c
unodos
>>> c = a * 3
>>> print c
unounouno
>>>
```



Listas

- La lista es un tipo de colección ordenada. Sería equivalente a lo que en otros lenguajes se conoce por arrays, o vectores.

```
>>> lista = [11, False]
>>> var = lista [0]
>>> print var
11
>>> var = lista [1]
>>> print var
False
>>> lista = ["una lista", 2, [2,3]]
>>> var = lista [2][1]
>>> print var
3
>>> print lista[-1]
[2,3]
```



Laboratorio Edumévil. M.C. Gabriel Gerónimo C.

Lista

```
>>> print lista[-1]
[2, 3]
>>> print lista[0:2]
['una lista', 2]
>>> print lista[0:4:2]
['una lista', [2, 3]]
>>> print lista[0:3:2]
['una lista', [2, 3]]
>>> print lista[0:3:1]
['una lista', 2, [2, 3]]
```

```
>>> c= "hola
JOHANNA"
>>> print c[0]
h
>>> print c[5:]
JOHANNA
>>> print c[::-2]
hl OAN
```



Iteraciones de orden superior sobre listas

filter(función, secuencia):

La **filter** verifica que los elementos de una secuencia cumplan una determinada condición, devolviendo una secuencia con los elementos que cumplen esa condición. Es decir, para cada elemento de **secuencia** se aplica la **función**; si el resultado es True se añade a la lista y en caso contrario se descarta.

```
>>> l = [1, 2, 3, 4, 5, 6]
>>> def es_par(n): return (n % 2.0 == 0)
...
>>> l2 = filter (es_par, l)
>>> print l, l2
[1, 2, 3, 4, 5, 6] [2, 4, 6]
>>>
```

Laboratorio Edumévil. M.C. Gabriel Gerónimo C.

Iteraciones de orden superior sobre listas

map(función, secuencia[, secuencia, ...]):

La función **map** aplica una **función** a cada elemento de una **secuencia** y devuelve una lista con el resultado de aplicar la función a cada elemento. Si se pasan como parámetros n secuencias, la función tendrá que aceptar n argumentos.

```
>>> def cuadrado (n): return n ** 2
...
>>> l = [1, 2, 3]
>>> l2 = map (cuadrado, l)
>>> print l, l2
[1, 2, 3] [1, 4, 9]
>>>
>>> l4 = map (lambda n: n **2, l)
>>> print l4
[1, 4, 9, 16, 25, 36]
>>>
```

Iteraciones de orden superior sobre listas

reduce(función, secuencia[, inicial]):

La función **reduce** aplica una **función** a pares de elementos de una **secuencia** hasta dejarla en un solo valor.

```
>>> def sumar (x, y): return x + y
...
>>> l = [1, 2, 3, 4, 5, 6]
>>> l3 = reduce (sumar, l)
>>> print l3
21
>>>
>>> l3 = reduce (lambda x, y: x + y, l)
```

Diccionarios

- Los diccionarios, también llamados matrices asociativas, deben su nombre a que son colecciones que relacionan una clave y un valor.
- >>> d = {"a": "abaco arbol ardilla", "b": "barril barco becerro", "c": "casa cama", "d": "dato dedo"}
- >>> print d["b"]
barril barco becerro



Sentencias condicionales

- La forma más simple de un estamento condicional es un **if** seguido de la condición a evaluar, dos puntos (**:**) y en la siguiente línea e indentado, el código a ejecutar en caso de que se cumpla dicha condición.

```
if fav == "manzana":
    print "Tienes buen gusto!"
    print "Gracias"
else:
    print "Vaya, que lástima"
```

```
if numero < 0:
    print "Negativo"
elif numero > 0:
    print "Positivo"
else:
    print "Cero"
```



Bucles

Los bucles permiten ejecutar un mismo fragmento de código un cierto número de veces, mientras se cumpla una determinada condición.

```
#!/usr/bin/python
while True:
    ent = raw_input ("> ")
    if ent == "adios":
        break
    else:
        print ent
```

```
#!/usr/bin/python
salir = False
while not salir:
    entrada = raw_input()
    if entrada == "adios":
        salir = True
    else:
        print entrada
```

Práctica 2.py

for ... in

```
>>> s = ["a", "b", "c", "d",
        "e", "f"]
>>> for letra in s: print
        letra
...
a
b
c
d
e
f
>>>
```

Funciones

Las funciones se declaran de la siguiente forma:

```
def mi_funcion(param1, param2):
    print param1
    print param2
```

```
def mi_funcion(param1,  
param2):  
    """ Esta funcion imprime los  
    dos valores  
    pasados como parametros """  
    print param1  
    print param2
```

Funciones

```
#!/usr/bin/python
def f(x, y):
    """esta es una funcion que
    imprime los valores por
    referencia"""
    x=x+ 3
    y.append(23)
    print x, y
x = 22
y = [22]
f(x, y)
print x, y
```

```
def sumar (x,y):  
    return x + y  
  
print sumar (3,2)  
  
def fun (x,y):  
    return x * 2, y * 2  
  
a,b = fun (1,2)  
print a, b
```

Práctica 3.py

Módulos

- Para facilitar el mantenimiento y la lectura los programas demasiado largos pueden dividirse en módulos, agrupando elementos relacionados.

modulo.py	programa.py
<pre>def mi_funcion(): print "Una funcion"</pre>	<pre>#!/usr/bin/python</pre>
<pre>class MiClase: def __init__(self): print "Una clase"</pre>	<pre>import modulo</pre>
<pre>print "Un modulo"</pre>	<pre>modulo.mi_función()</pre>
	<pre>#Gero\$./prog.py Un modulo Una funcion #Gero\$</pre>

Práctica 4.py

Módulos

- Algunos módulos por defecto
 - os**. Los módulos de la distribución por defecto de Python que engloba funcionalidad relacionada al Sistema Operativo
 - sys**. Con funcionalidad relacionada con el mismo interprete de Python
 - time**. Se almacenan funciones para manipular fechas y horas.
 - Sin embargo es posible utilizar la construcción **from-import** para ahorrar el indicar el nombre del módulo antes del objeto que interesa. De esta forma se importa el objeto o los objetos que indiquemos al espacio de nombres actual.

Ejemplo:

```
import os, sys, time
print time.asctime()
```

Ejemplo:

```
from time import asctime
print asctime()
```

Módulos

- A la hora de importar un módulo Python recorre todos los directorios indicados en la variable de entorno **PYTHONPATH** en busca de un archivo con el nombre adecuado. El valor de la variable **PYTHONPATH** se puede consultar desde Python mediante `sys.path`

```
>>> import sys
>>> sys.path
```

De esta forma para que nuestro módulo estuviera disponible para todos los programas del sistema bastará con copiarlo a uno de los directorios indicados en **PYTHONPATH**.

En el caso de que Python no encontrara ningún módulo con el nombre especificado, se lanzaría una excepción de tipo **ImportError**.

Entrada estándar

```

try:
    edad = raw_input("Que edad tienes?\n")
    dias = int(edad) * 365
    print "Has vivido " + str(dias) + " dias"

except ValueError:
    print "Eso no es un numero"

```

Parámetros de línea de comando

Para tener acceso a los parámetros de la línea de órdenes, debemos importar el módulo **sys**.

sys.argv[0] contiene siempre el nombre del programa tal como lo ha ejecutado el usuario, **sys.argv[1]**, si existe, sería el primer parámetro; **sys.argv[2]** el segundo, y así sucesivamente.

```

import sys
if(len(sys.argv) > 1):
    print "Abriendo " + sys.argv[1]
else:
    print "Debes indicar el nombre del archivo"

```

Archivos

Los archivos en Python son objetos de tipo **file** creados mediante la función **open**.

Esta función toma como parámetros una cadena con la ruta al archivo a abrir, que puede ser relativa o absoluta; una cadena opcional indicando el modo de acceso (si no se especifica se accede en modo lectura).

Archivos

El modo de acceso puede ser cualquier combinación lógica de los siguientes modos:

- **'r'**: Modo lectura. El archivo tiene que existir previamente.
- **'w'**: Modo escritura. Si el archivo no existe se crea. Si existe, sobreescribe el contenido.
- **'a'**: Modo añadir (escribir). Se comienza a escribir al final del archivo.
- **'b'**: binario.
- **'+'**: Permite lectura y escritura simultáneas.
- **'U'**: Universal newline, saltos de línea universales. Permite trabajar con archivos que tengan un formato para los saltos de línea que no coincide con el de la plataforma actual (en Windows se utiliza el carácter CR LF, en Unix LF y en Mac OS CR).

Lectura de archivos

Para la lectura de archivos se utilizan los métodos **read**, **readline** y **readlines**.

read(): Devuelve una cadena con el contenido del archivo o bien el contenido de los primeros n bytes, si se especifica el tamaño máximo a leer:

```
completo = f.read()
parte = f2.read(512)
```

readline(): Sirve para leer las líneas del fichero una por una.

readlines(): Lee todas las líneas del archivo y devolviendo una lista con las líneas leídas.

Laboratorio Edumóvil. M.C. Gabriel Gerónimo C.

Lectura de archivos

```
if (len(sys.argv) > 1):
    print "Abriendo" + sys.argv[1]
    f=open (sys.argv[1], "r+")
    while true:
        linea = f.readline()
        if not linea: break
        print linea,
    else:
        print "Debe indicar el
        archivo"
```

Práctica 5.py

Laboratorio Edumóvil. M.C. Gabriel Gerónimo C.

import

- Existen varios caminos para importar módulos al código. Accesando a las clases y funciones que contiene dicho modulo.
- `import mimodulo`
o una función del modulo ya cargado
`mimodulo.funcion()`
- una clase específica de un modulo
`from mimodulo import miclase`
- Importa cada una de las cosas del modulo
`from mimodulo import *`

import

- `>>> from datetime import datetime`
`>>> the_time = datetime.now()`
`>>> the_time.ctime()`
`'Thu Sep 10 19:01:26 2013'`
- `>>> from math import *`
`>>> print pi`
`3.14159265359`
`>>> sin (20)`
`0.91294525072762767`
- `>>> import random`
`>>> print random.randint (1,6)`
`4`

import

Class	Description
<code>timedelta</code>	Stores a difference between two times
<code>date</code>	Stores a date value
<code>datetime</code>	Stores date and time values
<code>time</code>	Stores a time value

Function	Description
<code>seed</code>	Seeds the random number generator
<code>randint</code>	Returns a random integer between two values
<code>choice</code>	Selects a random element from a collection
<code>random</code>	Return a float between 0 and 1

Referencias



1. Python para todos. Raúl González Duque. Licencia Creative Commons.
<http://mundogeek.net/tutorial-python/>
2. Python Programming. Wikibooks.org. 2012.
3. Python Algorithms. Magnus Lie Hetland. Apress. 2010.
4. Game Programming with Python. Sean Riley. Charles River Media. 2004.
5. Invent Your Own Computer Games with Python. Al Sweigart. Licencia Creative Commons. 2010.



Python: Módulo Pygame

M.C. Gabriel Gerónimo Castillo
Laboratorio Edumóvil
Universidad Tecnológica de la Mixteca
gcgero@gmail.com

pygame

- Página oficial <http://www.pygame.org/>
- Es una librería para la creación de juegos basada en SDL.

```
>>> import pygame
>>> print pygame.ver
1.9.1release
>>>
```
- El paquete **pygame** contiene un conjunto de módulos (de cada dispositivo) que pueden ser usados de forma independiente.

Módulos en el paquete pygame

Module Name	Purpose
pygame.cdrom	Accesses and controls CD drives
pygame.cursors	Loads cursor images
pygame.display	Accesses the display
pygame.draw	Draws shapes, lines, and points
pygame.event	Manages external events
pygame.font	Uses system fonts
pygame.image	Loads and saves an image
pygame.joystick	Uses joysticks and similar devices
pygame.key	Reads key presses from the keyboard
pygame.mixer	Loads and plays sounds
pygame.mouse	Manages the mouse
pygame.movie	Plays movie files
pygame.music	Works with music and streaming audio
pygame.overlay	Accesses advanced video overlays
pygame	Contains high-level Pygame functions
pygame.rect	Manages rectangular areas
pygame.sndarray	Manipulates sound data
pygame.sprite	Manages moving images
pygame.surface	Manages images and the screen
pygame.surfarray	Manipulates image pixel data
pygame.time	Manages timing and frame rate
pygame.transform	Resizes and moves images

Resoluciones para el Display

- ```
>>> import pygame
>>> pygame.init()
(6, 0)
>>> pygame.display.list_modes()
[(1280, 800), (1024, 768), (832, 624),
(800, 600), (720, 400), (640, 480),
(640, 400), (640, 350)]
```

## Ejemplo: Colocar imagen en pantalla

```
#!/usr/bin/python

import pygame
from pygame.locals import *
from sys import exit

imagen = "plato.jpg"

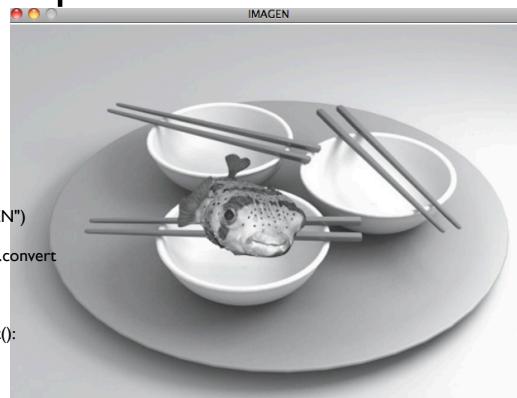
pygame.init()

pantalla = pygame.display.set_mode(
((640,480), 0, 32))
pygame.display.set_caption ("IMAGEN")

fondo = pygame.image.load(imagen).convert()

while True:
 for evento in pygame.event.get():
 if evento.type == QUIT:
 exit()
 pantalla.blit(fondo, (0,0))

 pygame.display.update()
```



**Práctica 5.py**

Laboratorio Edumóvil. M.C. Gabriel Gerónimo C.

# pygame.display

```
• pantalla = pygame.display.set_mode((640, 480), 0, 32)
 pygame.display.set_caption("IMAGEN")
```

El despliegue es una ventana en la computadora o una entrada en la pantalla, pero siempre se accede por medio un objeto de superficie de pygame. El llamado a ***pygame.display.set\_mode*** retorna dicha superficie que representa la ventana en la computadora.

# pygame.display

Sus parámetros son:

1. La dupla que contiene el ancho y alto. Este es el requerido, los siguientes son opcionales.
2. Bandera que se usa para el modo de despliegue. Se puede combinar las banderas con el operador **OR (|)**.
3. Profundidad. Especifica la cantidad de bits usados para almacenar los colores en el despliegue.

## Banderas y Bit de profundidad

| Flag       | Purpose                                                                             |
|------------|-------------------------------------------------------------------------------------|
| FULLSCREEN | Creates a display that fills the entire screen.                                     |
| DOUBLEBUF  | Creates a "double-buffered" display. Recommended for HWSURFACE or OPENGL.           |
| HWSURFACE  | Creates a hardware-accelerated display (must be combined with the FULLSCREEN flag). |
| OPENGL     | Creates an OpenGL renderable display.                                               |
| RESIZABLE  | Creates a resizable display.                                                        |
| NOFRAME    | Removes the border and title bar from the display.                                  |

| Bit Depth | Number of Colors                                          |
|-----------|-----------------------------------------------------------|
| 8 bits    | 256 colors, chosen from a larger <i>palette</i> of colors |
| 15 bits   | 32,768 colors, with a spare bit                           |
| 16 bits   | 65,536 colors                                             |
| 24 bits   | 16.7 million colors                                       |
| 32 bits   | 16.7 million colors, with a spare 8 bits                  |

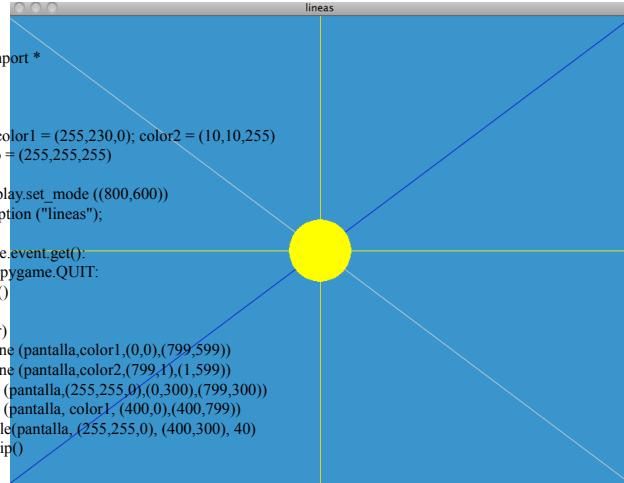
## Dibuja líneas en pantalla

```
#!/usr/bin/python
import pygame
from pygame.locals import *
from sys import exit

pygame.init()
color = (50,150,200); color1 = (255,230,0); color2 = (10,10,255)
negro = (0,0,0); blanco = (255,255,255)

pantalla = pygame.display.set_mode ((800,600))
pygame.display.set_caption ("líneas");
while True:

 for evento in pygame.event.get():
 if evento.type == pygame.QUIT:
 pygame.quit()
 sys.exit()
 pantalla.fill (color)
 pygame.draw.aaline (pantalla,color1,(0,0),(799,599))
 pygame.draw.aaline (pantalla,color2,(799,1),(1,599))
 pygame.draw.line (pantalla,(255,255,0),(0,300),(799,300))
 pygame.draw.line (pantalla, color1,(400,0),(400,799))
 pygame.draw.circle(pantalla, (255,255,0), (400,300), 40)
 pygame.display.flip()
```



## Dibuja líneas en pantalla y mueve cuadro

**ANEXAR.**

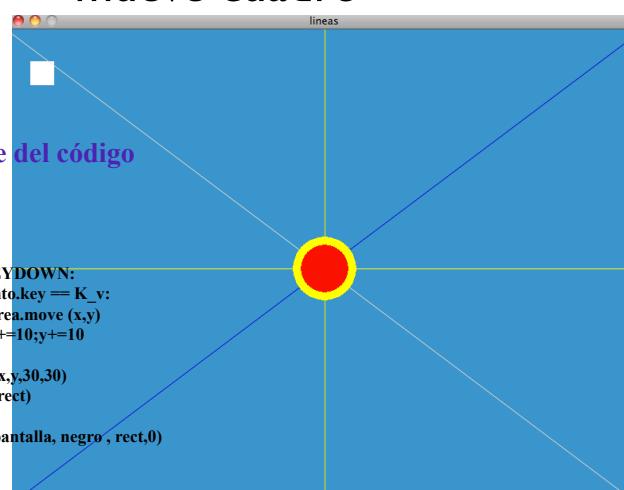
**En que parte del código anterior??**

```
x,y=10,20

if evento.type == KEYDOWN:
 if evento.key == K_v:
 area.move (x,y)
 x+=10;y+=10

rect = pygame.Rect(x,y,30,30)
area= pygame.Rect(rect)

pygame.draw.rect (pantalla, negro , rect,0)
```



## Mueve una pelota con cualquier evento

```
#!/usr/bin/python
import pygame,sys
pygame.init()
```

```
tam = ancho, alto = 320, 240
velocidad = [2, 2]
negro = (0, 0, 0)
pygame.display.set_caption ("Mueve - Pelota");
pantalla = pygame.display.set_mode(tam)
pelota = pygame.image.load("ball.bmp")
recpelota = pelota.get_rect()
while True:

 for event in pygame.event.get():
 if event.type == pygame.QUIT:
 sys.exit()
 recpelota = recpelota.move(velocidad)
 if recpelota.left < 0 or recpelota.right > ancho:
 velocidad[0] = -velocidad[0]
 if recpelota.top < 0 or recpelota.bottom > alto:
 velocidad[1] = -velocidad[1]

 pantalla.fill(negro)
 pantalla.blit(pelota, recpelota)
 pygame.display.flip()
```



## Ejemplo: Coloca imagen en pantalla y cambia imagen del curso

```
#!/usr/bin/python
import pygame
from pygame.locals import *
from sys import exit

imagen = "plato.jpg"; mouse = "Near.png"
pygame.init()
pantalla = pygame.display.set_mode ((640,480), 0, 32)
pygame.display.set_caption ("IMAGENES")

fondo = pygame.image.load(imagen).convert()
cursor = pygame.image.load (mouse).convert_alpha()
while True:
 for event in pygame.event.get():
 if event.type == QUIT:
 exit()
 x, y = pygame.mouse.get_pos()
 x -= cursor.get_width()/2
 y -= cursor.get_height()/2
 pantalla.blit(fondo, (0,0))
 pantalla.blit(cursor, (x,y))

 pygame.display.update()
```

## Práctica 6.py

## Desplegando imágenes

- Se usa la función `load` de `pygame.image` para cargar imágenes de fondo y del cursor del mouse.  
`fondo =`  
`pygame.image.load(imagen).convert()`  
`cursor = pygame.image.load (mouse).convert_alpha()`
- Se leen los archivos y se almacenan en una superficie en memoria, no se muestran hasta que se manden a dibujar.
- `convert()` convierte la imagen en el mismo formato del display, y `convert_alpha()` hace que la porción de imagen sea translúcida o completamente invisible.
- el miembro `blit` del objeto `surface` de la función, toma la imagen fuente y la posición destino (esquina izquierda superior de la pantalla)  
`pantalla.blit(fondo, (0,0))`  
`pantalla.blit (cursor, (x, y))`

## Desplegando imágenes

- Cuando construimos una imagen a través de blits sobre la superficie de la pantalla, no la vemos de forma inmediata. Esto es porque Pygame primero construye la imagen en un buffer alterno, lo cual lo hace invisible en memoria. El llamado a `pygame.display.update()` es necesario para garantizar que sean mostradas.

## Ejemplo: Mensaje pasando en la pantalla

```
#!/usr/bin/python
import pygame
from pygame.locals import *
from sys import exit
```

```
pygame.init()
imagen = "plato.jpg"
mensaje = " Demostracion de mensaje con scroll"
pantalla = pygame.display.set_mode ((640,480), 0, 32)
pygame.display.set_caption ("MENSAJE SCROLLY")
fuente = pygame.font.SysFont ("arial",60);
texto = fuente.render (mensaje, True, (0,0,255))
fondo = pygame.image.load(imagen).convert()

px = 0; py = (500 - texto.get_height()) / 2
while True:
 for event in pygame.event.get():
 if event.type == QUIT:
 exit()
 pantalla.blit(fondo, (0,0))
 px -=1
 if px <- texto.get_width():
 px = 0
 pantalla.blit(texto,(px,py))
 pantalla.blit (texto,(px+texto.get_width(), py))
 pygame.display.update()
```



**Práctica 6.1.py**

Laboratorio Edumévil. M.C. Gabriel Gerónimo C.

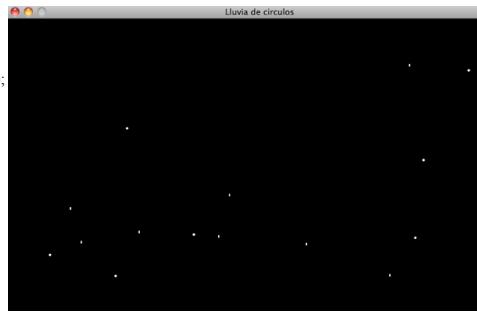
## Eventos estándares

| Event           | Purpose                                       | Parameters        |
|-----------------|-----------------------------------------------|-------------------|
| QUIT            | User has clicked the close button.            | none              |
| ACTIVEEVENT     | Pygame has been activated or hidden.          | gain, state       |
| KEYDOWN         | Key has been pressed.                         | unicode, key, mod |
| KEYUP           | Key has been released.                        | key, mod          |
| MOUSEMOTION     | Mouse has been moved.                         | pos, rel, buttons |
| MOUSEBUTTONDOWN | Mouse button was pressed.                     | pos, button       |
| MOUSEBUTTONUP   | Mouse button was released.                    | pos, button       |
| JOYAXISMOTION   | Joystick or pad was moved.                    | joy, axis, value  |
| JOYBALLMOTION   | Joy ball was moved.                           | joy, ball, rel    |
| JOYHATMOTION    | Joystick hat was moved.                       | joy, hat, value   |
| JOYBUTTONDOWN   | Joystick or pad button was pressed.           | joy, button       |
| JOYBUTTONUP     | Joystick or pad button was released.          | joy, button       |
| VIDEORESIZE     | Pygame window was resized.                    | size, w, h        |
| VIDEOEXPOSE     | Part or all of the Pygame window was exposed. | none              |
| USEREVENT       | A user event has occurred.                    | code              |

## Lluvia de círculos

```
#!/usr/bin/python
import pygame,sys, random
pygame.init()

negro = (0,0,0); blanco = (255,255,255)
pantalla = pygame.display.set_mode ((800,600))
pygame.display.set_caption ("Lluvia de circulos");
lista=[]
for i in range (20):
 x=random.randrange(0,800)
 y=random.randrange(0,800)
 r=random.randrange(1,3)
 lista.append([x,y,r])
while True:
 for evento in pygame.event.get():
 if evento.type == pygame.QUIT:
 pygame.quit(); sys.exit()
 pantalla.fill (negro)
 for i in range (len(lista)):
 x,y,radio=lista[i][0],lista[i][1],lista[i][2]
 pygame.draw.circle(pantalla,blanco,(x,y),radio)
 lista[i][1]-=1
 if lista[i][1]>590:
 y=random.randrange(-50,-10)
 lista[i][1] = y
 x=random.randrange(0,800)
 lista[i][0] = x
 pygame.display.flip()
pygame.quit ()
```



## Colocando una imagen bajo la lluvia de círculos

```
#!/usr/bin/python
import pygame,sys, random
from pygame.locals import *
from sys import exit

pygame.init()
negro = (0,0,0); blanco = (255,255,255)
pantalla = pygame.display.set_mode ((800,600))
pygame.display.set_caption ("Mario");
imagen = pygame.image.load("mario.png")
x = 100 ;y = 500

mario = pygame.sprite.Sprite()
mario.image = imagen
mario.rect = imagen.get_rect()

oldx = mario.rect.top = y
oldy = mario.rect.left = x
reloj = pygame.time.Clock()
lista=[]; listarect=[]
for i in range (20):
 x=random.randrange(0,800)
 y=random.randrange(0,800)
 r=random.randrange(1,3)
 lista.append([x,y,r])
 listarect.append (pygame.Rect (x-r,y+r,2*r,2*r))

c=0
```

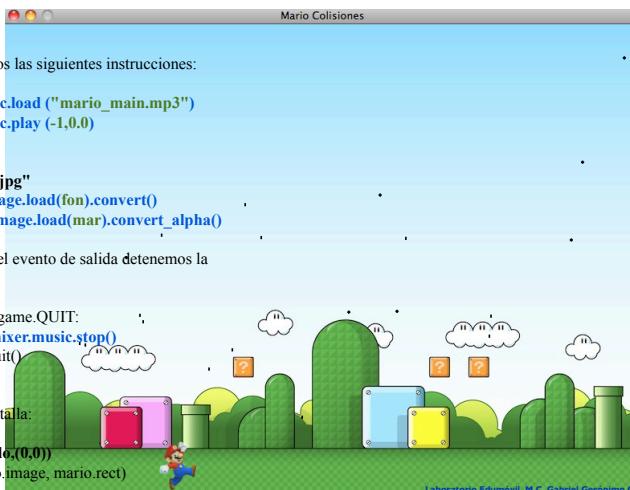
## Colocando una imagen bajo la lluvia de círculos



```
for i in range (len(lista)):
 x,y, radio=lista[i][0],lista[i][1], lista[i][2]
 pygame.draw.circle(pantalla,negro,(x,y),radio)
 lista[i][1]+=1
 if lista[i][1]>590:
 y=random.randrange(-50,-10)
 lista[i][1] = y
 x=random.randrange(0,800)
 lista[i][0] = x
 listarect.append(pygame.Rect (x-radio,y+radio,2*radio,2*radio))
pygame.display.flip()
pygame.quit()
```



## Mario bajo la lluvia de círculos, con imagen y música de fondo



Para esto aumentamos las siguientes instrucciones:

```
pygame.mixer.music.load ("mario_main.mp3")
pygame.mixer.music.play (-1,0.0)

mar="mario.png"
fon="fondo_mario.jpg"
fondo = pygame.image.load(fon).convert()
imagen = pygame.image.load(mar).convert_alpha()
```

Cuando chequemos el evento de salida detenemos la música:

```
if evento.type == pygame.QUIT:
 pygame.mixer.music.stop()
 pygame.quit()
 sys.exit()
```

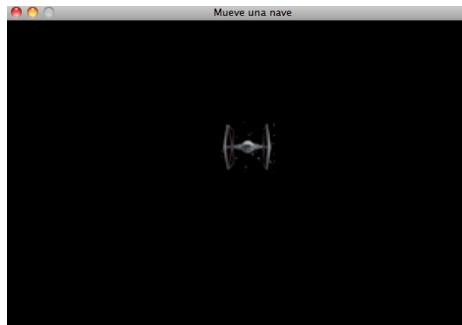
Y para mandar a pantalla:

```
pantalla.blit(fondo,(0,0))
pantalla.blit(mario.image, mario.rect)
```

## Mueve una nave en la pantalla

```
#!/usr/bin/python
import pygame
from pygame.locals import *
from sys import exit

pygame.init()
nave = "nave.jpg"
pantalla = pygame.display.set_mode ((640,480), 0, 32)
pygame.display.set_caption ("Mueve una nave")
nave1 = pygame.image.load(nave).convert()
x, y = 0, 0; mx, my = 0, 0
while True:
 for event in pygame.event.get():
 if event.type == QUIT:
 exit()
 if event.type == KEYDOWN:
 if event.key == K_LEFT: mx = -1
 if event.key == K_RIGHT: mx = +1
 if event.key == K_UP: my = -1
 if event.key == K_DOWN: my = +1
 elif event.type == KEYUP:
 if event.key == K_LEFT: mx = 0
 if event.key == K_RIGHT: mx = 0
 if event.key == K_UP: my = 0
 if event.key == K_DOWN: my = 0
 x+= mx; y+= my
 pantalla.fill ((0,0,0))
 pantalla.blit(nave1, (x,y))
 pygame.display.update()
```



## Práctica 8.py

```
#!/usr/bin/python
import pygame
from pygame.locals import *
from sys import exit
```

```
pygame.init()
nave = "nave.jpg"
pantalla = pygame.display.set_mode ((640,480), 0, 32)
pygame.display.set_caption ("Mueve dos naves")
nave1 = pygame.image.load(nave).convert()
nave2 = pygame.image.load(nave).convert()
x, y = 0, 0; r, s = 450, 10
mx, my , mr, ms = 0,0,0,0
while True:
 for event in pygame.event.get():
 if event.type == QUIT: exit()
 if event.type == KEYDOWN:
 if event.key == K_a: mr = -1
 if event.key == K_z: mr = +1
 if event.key == K_s: ms = -1
 if event.key == K_x: ms = +1
 if event.key == K_LEFT: mx = -1
 if event.key == K_RIGHT: mx = +1
 if event.key == K_UP: my = -1
 if event.key == K_DOWN: my = +1
 elif event.type == KEYUP:
 if event.key == K_a: mr = 0
 if event.key == K_z: mr = 0
 if event.key == K_s: ms = 0
 if event.key == K_x: ms = 0
 if event.key == K_LEFT: mx = 0
 if event.key == K_RIGHT: mx = 0
 if event.key == K_UP: my = 0
 if event.key == K_DOWN: my = 0
 x+= mx; y+= my;r+= mr;s+= ms
 pantalla.fill ((0,0,0))
 pantalla.blit(nave1, (x,y))
 pantalla.blit(nave2, (r,s))
 pygame.display.update()
```



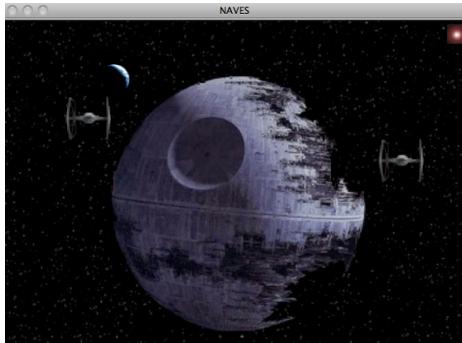
## Práctica 9.py

Laboratorio Edumóvil. M.C. Gabriel Gerónimo C.

## Mueve dos naves en la pantalla con imagen de fondo

```
#!/usr/bin/python
import pygame
from pygame.locals import *
from sys import exit
pygame.init()
```

```
nave = "nave.png"
estrella = "e_mueru.jpg"
pantalla = pygame.display.set_mode ((640,480), 0, 32)
pygame.display.set_caption ("NAVES")
fondo = pygame.image.load(estrella).convert()
nave1 = pygame.image.load(nave).convert_alpha()
nave2 = pygame.image.load(nave).convert_alpha()
x, y = 0, 0; r, s = 450, 10
mx, my , mr, ms = 0,0,0,0
while True:
 for event in pygame.event.get():
 if event.type == QUIT: exit()
 if event.type == KEYDOWN:
 if event.key == K_a: mr = -1
 if event.key == K_z: mr = +1
 if event.key == K_s: ms = -1
 if event.key == K_x: ms = +1
 if event.key == K_LEFT: mx = -1
 if event.key == K_RIGHT: mx = +1
 if event.key == K_UP: my = -1
 if event.key == K_DOWN: my = +1
 elif event.type == KEYUP:
 if event.key == K_a or event.key == K_z: mr = 0
 if event.key == K_s or event.key == K_x: ms = 0
 if event.key == K_LEFT or event.key == K_RIGHT : mx = 0
 if event.key == K_UP or event.key == K_DOWN: my = 0
 x+= mx;y+= my; r+= mr;s+= ms
 pantalla.fill ((0,0,0))
 pantalla.blit(fondo,(0,0))
 pantalla.blit(nave1, (x,y))
 pantalla.blit(nave2, (r,s))
 pygame.display.update()
```



## Práctica 10.py

# Referencias



1. Making Games with Python & Pygame. Al Sweigart. Licensed under a Creative Commons.2012.
2. Begin Game Development with Python and Pygame. Will McGugan. Apress. 2007.
2. The Art of Computer Game Design. Chris Crawford.

---

---

---

---

---

---

---

---

---