Colección de actividades Aprender Conectados

Nivel Secundario

Programación

Arte digital

Actividad N° 6

APRENDER CONECTADOS

Ministerio de Educación, Cultura, Ciencia y Tecnología
Presidencia de la Nación

Autoridades

Presidente de la Nación
Mauricio Macri

Jefe de Gabinete de Ministros
Marcos Peña

Ministro de Educación, Cultura, Ciencia y Tecnología
Alejandro Finocchiaro

Secretario de Gobierno de Cultura
Pablo Avelluto

Secretario de Gobierno de Ciencia, Tecnología e Innovación Productiva
Lino Barañao

Titular de la Unidad de Coordinación General del Ministerio de Educación, Cultura, Ciencia y Tecnología
Manuel Vidal

Secretaria de Innovación y Calidad Educativa
Mercedes Miguel

Subsecretario de Coordinación Administrativa
Javier Mezzamico

Directora Nacional de Innovación Educativa
María Florencia Ripani

ISBN en trámite

Este contenido fue producido por el Ministerio de Educación, Cultura, Ciencia y Tecnología de la Nación en el marco del Plan Aprender Conectados

Introducción

El Plan Aprender Conectados es la primera iniciativa en la historia de la política educativa nacional que se propone implementar un programa integral de alfabetización digital, con una clara definición sobre los contenidos indispensables para toda la Argentina.

En el marco de esta política pública, el Consejo Federal de Educación aprobó, en 2018, los Núcleos de Aprendizajes Prioritarios (NAP) de Educación Digital, Programación y Robótica (EDPR) para toda la educación obligatoria, es decir, desde la sala de 4 años hasta el fin de la secundaria. Abarcan un campo de saberes interconectados y articulados, orientados a promover el desarrollo de competencias y capacidades necesarias para que los estudiantes puedan integrarse plenamente en la cultura digital, tanto en la socialización, en la continuidad de los estudios y el ejercicio de la ciudadanía, como en el mundo del trabajo.

La incorporación de Aprender Conectados en la Educación Secundaria permite poner a disposición estudiantes y docentes, tecnología y contenidos digitales que generan nuevas oportunidades para reconocer y construir la realidad: abre una ventana al mundo, facilita la comunicación y la iniciación a la producción digital.

La sociedad está cambiando a un ritmo más acelerado que nuestro sistema educativo y la brecha entre las propuestas pedagógicas que presentan las escuelas y la vida de los estudiantes se amplía cada vez más. Garantizar el derecho a aprender en el siglo XXI implica que todos los estudiantes puedan desarrollar las capacidades necesarias para actuar, desenvolverse y participar como ciudadanos en esta sociedad cada vez más compleja, con plena autonomía y libertad.

En este marco, Aprender Conectados presenta actividades, proyectos y una amplia variedad de recursos educativos para orientar la alfabetización digital en la educación obligatoria en todo el país. La actividad que se presenta a continuación y el resto de los recursos del Plan son un punto de partida sobre el cual cada docente podrá construir propuestas y desafíos que inviten a los estudiantes a disfrutar y construir la aventura del aprender.

María Florencia Ripani
Directora Nacional de Innovación Educativa

Objetivos generales

Núcleos de Aprendizajes Prioritarios

Educación Digital, Programación y Robótica – Educación secundaria	Ofrecer situaciones de aprendizaje que promuevan en los alumnas y alumnos: • La comprensión general del funcionamiento de los componentes de *hardware* y *software*, y la forma en que se comunican entre ellos y con otros sistemas, entendiendo los principios básicos de la digitalización de la información y su aplicación en la vida cotidiana. • El desarrollo de proyectos creativos que involucren la selección y la utilización de múltiples aplicaciones, en una variedad de dispositivos, para alcanzar desafíos propuestos, que incluyan la recopilación y el análisis de información. • La creación, la reutilización, la reelaboración y la edición de contenidos digitales en diferentes formatos, entendiendo las características y los modos de representación de lo digital. • La resolución de problemas a partir de su descomposición en partes pequeñas, aplicando diferentes estrategias, utilizando entornos de programación tanto textuales como icónicos, con distintos propósitos, incluyendo el control, la automatización y la simulación de sistemas físicos.

Objetivos de aprendizaje

Esta actividad permitirá introducir al lenguaje de programación Python y está orientada a desarrollar conocimientos iniciales vinculados con los siguientes objetivos de aprendizaje:

- Aprender a generar aleatoriedad en el código.
- Profundizar los conceptos de funciones y estructura de datos.
- Profundizar los conceptos de estructuras de selección y bucles.
- Profundizar en el manejo de gráficos.
- Realizar un proyecto que genere arte aleatorio con figuras geométricas dentro de una ventana con componentes gráficos.

Materiales y recursos

Computadora.
Python 2.x instalado

Desafío

Las artes son consideradas la máxima expresión humana, pero nada impide utilizar un poco de automatización y azar para realizarlas. En esta actividad, te desafiamos a que crees tu propio cuadro abstracto de forma automática y aleatoria utilizando Python.
En esta actividad aprendemos a generar arte aleatorio con la ayuda de figuras geométricas y colores, al azar. Para esto, utilizaremos funciones personalizadas dentro de nuestro código, diferentes estructuras de datos, manejo de entrada y salida de datos, sentencias de decisión, bucles y manejo de gráficos.

< Inicio >

Disparador

"¿Te gustaría hacer arte digital? ¿Qué tal escribir código para que un programa lo haga por vos?"

En esta actividad vamos a escribir un programa que, al ser ejecutado, crea una ventana con figuras geométricas de formas y tamaños al azar, devolviendo un resultado diferente, cada vez que se lo ejecuta.

Las líneas de código son presentadas dentro de este documento con el siguiente formato para su fácil identificación y copiado.

```
# Soy un comentario en el código
print 'Soy una línea de código'
Soy una línea de código
```

Las líneas de color azul son la respuesta al código introducido en las líneas anteriores, y se presentan como un ejemplo del resultado a obtener. No deben ser copiadas ni ejecutadas en IDLE.

También se incluyen comentarios en gris, precedidos por el símbolo numeral. Estos comentarios son notas que dan claridad al código, pero que Python ignora y no son ejecutados.

Se puede comenzar la actividad con un pequeño debate alusivo: ¿Qué es el arte?, ¿Es algo exclusivo de los humanos o una computadora puede realizarlo? Luego, se puede hablar sobre el arte asistido por computadoras, para introducir el ejercicio de esta actividad. Con esto, se fomenta el interés por el desafío y se da lugar al debate y la reflexión entre compañeros.

El ejercicio se propone para ser escrito línea por línea, en IDLE. Es importante que todos los estudiantes estén frente a una computadora con IDLE, de Python, abierto al momento de comenzar la actividad y que, a medida que avanzan, ejecuten el programa para comprobar que su código esté bien y corregir errores, si los hubiere.

Al final de este documento se presentan todas las líneas del programa juntas, que pueden ser guardadas en un archivo desde IDLE, para ser ejecutado.

< Desarrollo >

La creación del programa se puede dividir en los siguientes pasos:

1. Importación de librerías externas.
2. Configuración de la ventana.
3. Creación de funciones para aleatoriedad.
4. Creación de forma personalizada.
5. Definición y ejecución de bucles.

El docente recuerda a los alumnos acerca de la importancia de utilizar los comentarios e invita a colocar los códigos para que Python reconozca todos los acentos y signos en los sistemas operativos Linux y Windows. También se sugiere ingresar los datos del autor y el nombre del programa.

```
# -*- coding: cp1252 -*-
# -*- coding: 850 -*-
# -*- coding: utf-8 -*-
# Creador: Pedro Perez
# Arte Digital
```

Recordá utilizar comentarios de forma frecuente explicando tu código. ¡El programador que lo lea te lo agradecerá!

1) Importación de librerías externas

Existen situaciones en las que solo vamos a utilizar una o dos funciones de una librería en particular. En esta oportunidad vamos a ver cómo cargar solo algunas funciones de una librería, en vez de importar la librería completa. Con esto, optimizamos nuestro código tanto en su tamaño en bytes como en su velocidad de ejecución.

Dado que vamos a trabajar con gráficos, importamos la librería *Turtle*, y en función de que buscamos generar formas con tamaños y colores al azar, vamos a importar dos funciones de la librería "*random*". Esta librería nos provee de funciones para obtener números y selecciones al azar. Para esta actividad utilizaremos las funciones "randint()", que nos devuelve un número entero al azar, y "*choice*()" que nos devuelve un elemento al azar de una lista ordenada de elementos.

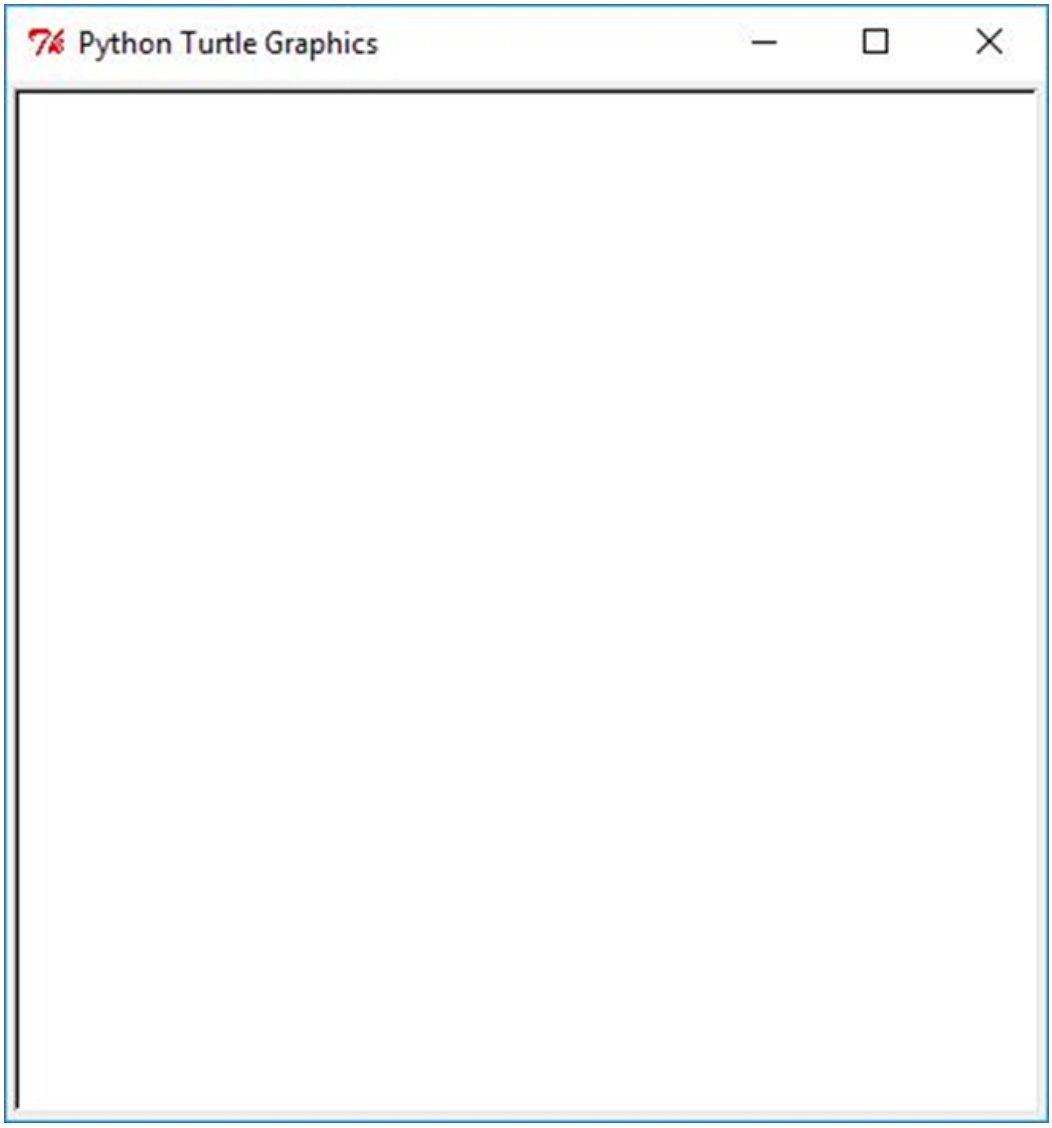
```
from turtle import *
from random import randint
from random import choice
```

¿Se te ocurre cómo optimizar aún más este programa? ¿Se podrían importar solo las funciones de Turtle que usamos en vez de la librería completa?

2) Configuración de la ventana

Comenzaremos por crear y configurar la ventana que nos servirá de lienzo.

```
screen=Screen() # Inicializamos nuestra ventana
screen.colormode(255) # Definimos el rango de intensidad de los colores
entre 0 y 255
screen.setup(420,420) # Definimos un tamaño de pantalla de 420 pixeles
por 420 pixeles
```



Resultado al ejecutar el código de ejemplo

3) Creación de funciones para aleatoriedad

Cada vez que nos encontramos programando cálculos, tareas o acciones que tenemos que repetir una y otra vez, es muy probable que una función sea la forma más eficiente de implementarlo. Las funciones nos permiten escribir solo una vez el código para resolver dicha tarea, y ejecutarla cada vez que lo deseemos con solo invocar la función.

Para este ejercicio vamos a dibujar en pantalla formas con colores aleatorios y en ubicaciones aleatorias. Esto significa que por cada forma que vayamos a dibujar, le vamos a tener que indicar un nuevo color y una nueva ubicación. Calcular el nuevo color y la nueva ubicación son tareas susceptibles de ser creadas como funciones, dado que las vamos a repetir una y otra vez. A continuación, se presenta el código para la creación de estas funciones. Se utiliza la función “randint()” de la librería “random”. Esta función nos devuelve un número entero al azar. Se le puede definir un rango pasándole como parámetros el número entero mínimo y máximo.

```
# Recordemos que el color se forma con la intensidad entre 0 y 255 de
cada una de sus componentes primarias (rojo, verde y azul).
def cualquierColor():
 # Ahora utilizamos randint() para obtener un número al  azar entre 0 y
255 para cada una de las componentes del color
 rojo=randint(0,255)
 verde=randint(0,255)
 azul=randint(0,255)
 color(rojo,verde,azul) # La resultante será un nuevo color al azar cada
vez que ejecutemos la función

# Recordemos que establecemos nuestra posición mediante la función
"goto(x,y)" de la librería Turtle. Se calcula como un eje cartesiano con
cero al centro, donde -210 y 210 son los límites para nuestra ventana de
420x420
def cualquierLugar():
 penup() # Levantamos la lapicera para no dibujar al movernos
 # Definimos coordenadas "x" e "y" como un número entre -190 y 190, para
darle a nuestras formas un margen al borde de nuestra ventana.
 x=randint(-180,180)
 y=randint(-180,180)
 goto(x,y) # Nos movemos hacia la posición x, y, que será diferente cada
vez que se ejecuta la función


# Finalmente, creamos una función para definir una dirección al azar, y
así obtener diferentes rotaciones para nuestras figuras geométricas.
def cualquierDireccion():
 right(randint(0,360)) # Rotamos la figura en sentido del reloj un
número de grados al azar, entre 0 y 360.
```

4) Creación de forma personalizada

Si bien la librería *Turtle* cuenta con formas como flecha, triángulo, círculo, cuadrado y hasta una tortuga, no cuenta con la posibilidad de dibujar rectángulos. Dado que queremos dibujar varios rectángulos al azar, creamos una función que los dibuje de forma aleatoria. Para esto volveremos a utilizar las funciones de movimiento de *Turtle* para dibujar los lados del cuadrado. También utilizamos las funciones "*begin_fill*()" y "*end_fill*()" de *Turtle*. Estas funciones se invocan al comenzar y al finalizar el dibujo de nuestra forma para darle un color sólido como relleno.

```
def dibujarRectangulo():
 hideturtle() # Escondemos nuestro puntero
 ancho=randint(10,100) # Definimos un ancho al zar entre 10 y 100
 alto=randint(10,100) # Definimos un alto al azar entre 10 y 100
# Antes de comenzar a dibujar la forma ejecutamos la función
begin_fill() para darle un relleno sólido.
 begin_fill()
 forward(ancho) # Dibujamos un lado del rectángulo
 right(90) # Rotamos 90 grados sentido agujas del reloj
 forward(alto) # Dibujamos el segundo lado del rectángulo
 right(90) # Rotamos 90 grados sentido agujas del reloj
 forward(ancho) # Dibujamos el tercer lado del rectángulo
 right(90) # Rotamos 90 grados sentido agujas del reloj

 forward(alto) # Dibujamos el cuarto lado del rectángulo
 right(90) # Rotamos 90 grados sentido agujas del reloj
 end_fill() # Finalizamos el relleno de la figura
```

¿Te animás a crear funciones que creen otras formas personalizadas?

6) Definición y ejecución de bucles

Un bucle, en programación, es el método para ejecutar una porción de código, una y otra vez, hasta que se cumpla una condición determinada. Nuestro plan es crear un número finito de formas geométricas de forma automatizada. Para esto, utilizaremos bucles.

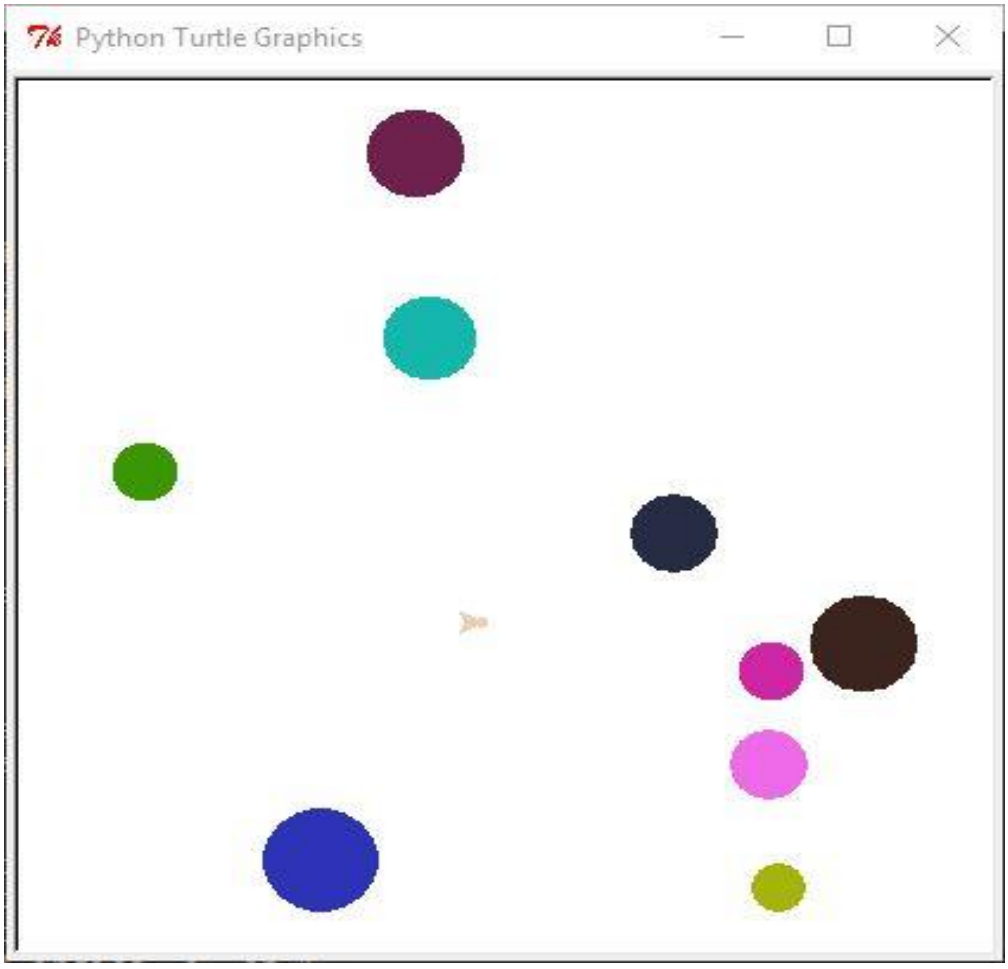
Los bucles son herramientas muy poderosas pero, si no los limitamos correctamente, pueden ejecutarse sin fin, llevando al programa a un estado de bloqueo.

Para evitar esto, debemos definir la condición para que el bucle termine. En este caso, definimos el límite de formas a ser creadas por cada bucle en 25.
Para comenzar nuestro cuadro, dibujaremos algunos círculos al azar utilizando el bucle "*while*" de Python, que nos permite ejecutar un bloque de código, una y otra vez, mientras que la condición indicada sea verdadera. En este caso, nuestra condición es que el índice "i", con valor inicial de 0, sea menor a un número entre 1 y el límite establecido de 25.

```
limite=25  # Límite de hasta 25 objetos por bucle
i=0 # valor inicial del índice

 # Nuestro bucle se ejecutará mientras el índice "i" sea menor al número
que nos devuelve "randint(0,limite)". randint(0,limite) nos devuelve un
número al azar entre 1 y 25.
while(i < randint(1,limite)):
 penup() # Levanta la lapicera para no dibujar al moverse
 cualquierColor() # Selecciona un color al azar
 cualquierLugar() # Selecciona una posición al azar
 dot(randint(0,50)) # Dibuja un círculo de un tamaño entre 0 y 50.
 i += 1 # incrementamos el valor del índice "i" en 1.
```

Este código dibuja al menos un círculo al azar. ¿Se te ocurre cómo hacer para que dibuje al menos 3 círculos al azar?



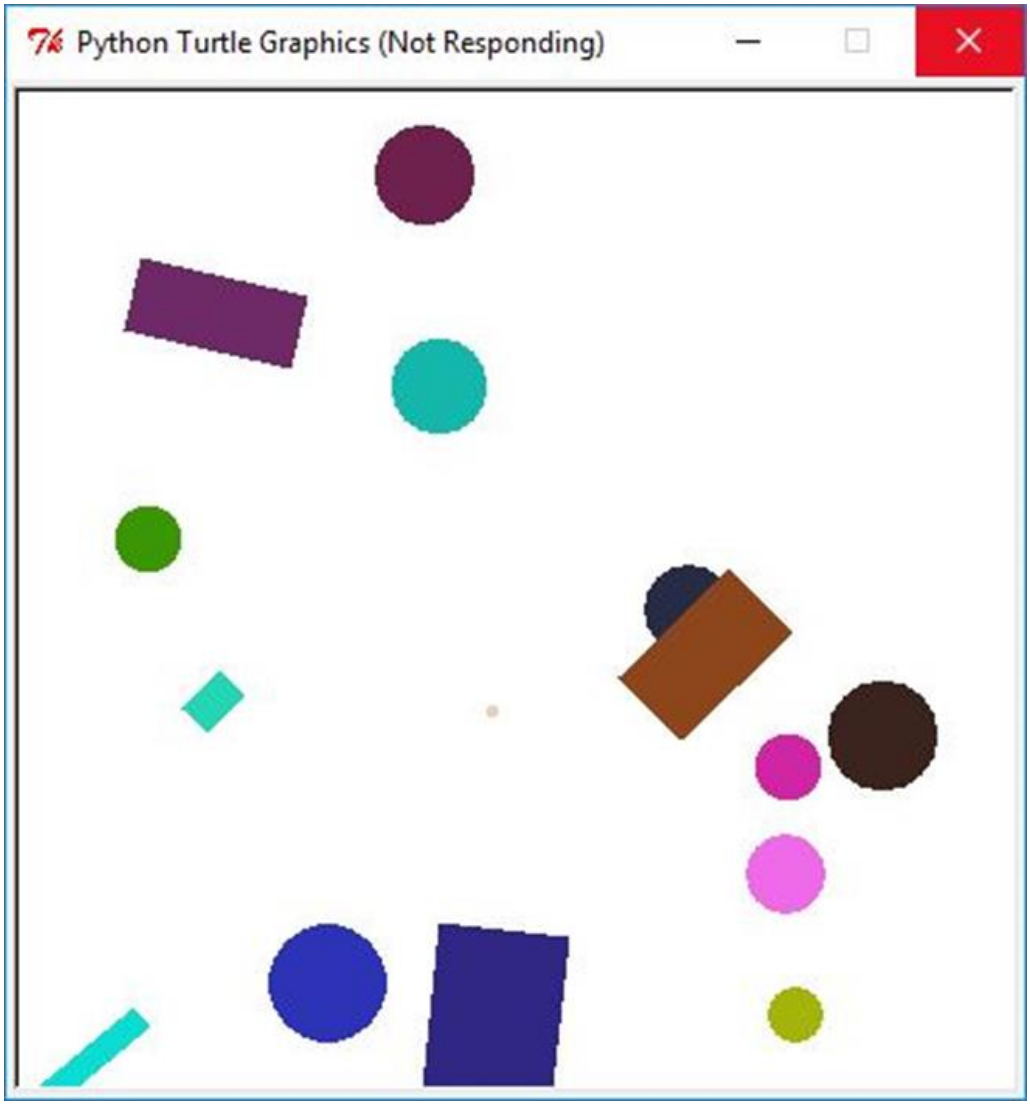
Resultado al ejecutar el código de ejemplo

A continuación vamos a dibujar rectángulos al azar, utilizando la función "dibujarRectangulo()" que creamos anteriormente, y limitando el número de rectángulos con un bucle "*while*".

```
i=0 # Reiniciamos el valor del índice a 0

# Nuevamente nuestro bucle se ejecutará entre 1 y 25 veces, según el
resultado de "randint(1,limite)".
while(i < randint(1,limite)):
 penup() # Levanta la lapicera para no dibujar al moverse
 cualquierColor() # Selecciona un color al azar
 cualquierLugar() # Selecciona una posición al azar
 cualquierDireccion() # Selecciona una dirección al azar
 dibujarRectangulo() # Dibuja un rectángulo al azar.
 i += 1 # incrementamos el valor del índice "i" en 1.
```

¿Notaste que utilizamos la función "cualquierDireccion()"? ¿Por qué no se usó al dibujar los círculos?



Resultado al ejecutar el código de ejemplo

Por último, vamos a imprimir en la ventana las formas de puntero disponibles en *Turtle*, con variados colores y tamaños. Para aumentar la aleatoriedad del resultado definimos las posibles formas del puntero de *Turtle* en una lista. Luego obtendremos un elemento al azar de la misma utilizando la función "*choice*()" de la librería *Random*, y utilizamos esta forma elegida al azar para dibujar en nuestra ventana. Se usan 3 funciones de *Turtle* para esta tarea. La función "*shape*()" que nos permite definir la forma del puntero, la función "*shapezise*()" que nos permite definir el tamaño del puntero, y finalmente la función "*stamp*()" para imprimir el puntero en la ventana.

```
i=0 # Reiniciamos el valor del índice a 0
# Definimos una lista con nombre "formas". Cada elemento es una posible
forma del puntero de turtle.
formas=['arrow', 'turtle', 'circle', 'square', 'triangle', 'classic']
while(i < randint(5,limite+2)):
 shape(choice(formas)) # Elige una forma al azar de las disponibles en
la lista "formas"
 shapesize(None,None,randint(1,5)) # Le da un tamaño entre 1 y 5 al
puntero.
 cualquierColor() # Elige un color al azar
 cualquierLugar() # Elige un lugar al azar
 cualquierDireccion() # Elije una dirección al azar.
 stamp() # Imprime la forma en la ventana.
 i += 1
```

¿Cuál es la cantidad mínima de dibujos que imprimirá esta función? ¿Y cuál es la cantidad máxima?

Código completo

A continuación presentamos el código completo para ser copiado en un nuevo archivo de IDLE. Podés guardarlo con el nombre arte_geometrico.py

```
# -*- coding: cp1252 -*-
# -*- coding: 850 -*-
# -*- coding: utf-8 -*-
# Creador: Pedro Perez
# Arte Digital
# Importamos librerías externas
from turtle import *
from random import randint
from random import choice

# configuración de la ventana
screen=Screen() # Inicializamos nuestra ventana
screen.colormode(255) # Definimos el rango de intensidad de
los colores entre 0 y 255
screen.setup(420,420) # Definimos un tamaño de pantalla de
420 píxeles por 420 píxeles

# Creación de funciones para aleatoriedad
# Recordemos que el color se forma con la intensidad entre 0
y 255 de cada una de sus componentes primarias (rojo, verde y
azul).
def cualquierColor():
 # Ahora utilizamos randint() para obtener un número al  azar
entre 0 y 255 para cada una de las componentes del color
 rojo=randint(0,255)
 verde=randint(0,255)
 azul=randint(0,255)
 color(rojo,verde,azul) # La resultante será un nuevo color
al azar cada vez que ejecutemos la función

# Recordemos que establecemos nuestra posición mediante la
función "goto(x,y)" de la librería Turtle. Se calcula como un
eje cartesiano con cero al centro, donde -210 y 210 son los
límites para nuestra ventana de 420x420
def cualquierLugar():
 penup() # Levantamos la lapicera para no dibujar al movernos
 # Definimos coordenadas "x" e "y" como un número entre -190
y 190, para darle a nuestras formas un margen al borde de
nuestra ventana.
 x=randint(-180,180)
 y=randint(-180,180)
 goto(x,y) # Finalmente nos movemos hacia la posición x, y,
que será diferente cada vez que se ejecuta la función
```

```
# Finalmente, creamos una función para definir una dirección al azar, y así obtener diferentes rotaciones para nuestras figuras geométricas.
def cualquierDireccion():
 right(randint(0,360)) # Rotamos la figura en sentido del reloj un número de grados al azar entre 0 y 360.

# Creación de forma personalizada
def dibujarRectangulo():
 hideturtle() # Escondemos nuestro puntero
 ancho=randint(10,100) # Definimos un ancho al zar entre 10 y 100
 alto=randint(10,100) # Definimos un alto al azar entre 10 y 100
 # Antes de comenzar a dibujar la forma ejecutamos la función begin_fill() para darle un relleno sólido.
 begin_fill()
 forward(ancho) # Dibujamos un lado del rectángulo
 right(90) # Rotamos 90 grados sentido agujas del reloj
 forward(alto) # Dibujamos el segundo lado del rectángulo
 right(90) # Rotamos 90 grados sentido agujas del reloj
 forward(ancho) # Dibujamos el tercer lado del rectángulo
 right(90) # Rotamos 90 grados sentido agujas del reloj
 forward(alto) # Dibujamos el cuarto lado del rectángulo
 right(90) # Rotamos 90 grados sentido agujas del reloj
 end_fill() # Finalizamos el relleno de la figura

# Definición y ejecución de bucles
limite=25  # Límite de hasta 25 objetos por bucle
i=0 # valor inicial del índice

# Dibujando círculos
# Nuestro bucle se ejecutará mientras el índice "i" sea menor al número que nos devuelve "randint(0,limite)".
randint(0,limite) nos devuelve un número al azar entre 1 y 25.
while(i < randint(1,limite)):
 penup() # Levanta la lapicera para no dibujar al moverse
 cualquierColor() # Selecciona un color al azar
 cualquierLugar() # Selecciona una posición al azar
 dot(randint(0,50)) # Dibuja un círculo de un tamaño entre 0 y 50.
 i += 1 # incrementamos el valor del índice "i" en 1.
```

```
# Dibujando rectángulo
i=0 # Reiniciamos el valor del índice a 0
# Nuevamente nuestro bucle se ejecutará entre 1 y 25 veces,
según el resultado de "randint(1,limite)".
while(i < randint(1,limite)):
 penup() # Levanta la lapicera para no dibujar al moverse
 cualquierColor() # Selecciona un color al azar
 cualquierLugar() # Selecciona una posición al azar
 cualquierDireccion() # Selecciona una dirección al azar
 dibujarRectangulo() # Dibuja un rectángulo al azar.
 i += 1 # incrementamos el valor del índice "i" en 1.

#Dibujando con el puntero de Turtle
i=0 # Reiniciamos el valor del índice a 0
# Definimos una lista con nombre "formas". Cada elemento es
una posible forma del puntero de turtle.
formas=['arrow', 'turtle', 'circle', 'square', 'triangle',
'classic']
while(i < randint(5,limite+2)):
 shape(choice(formas)) # Elige una forma al azar de las
disponibles en la lista "formas"
 shapesize(None,None,randint(1,5)) # Le da un tamaño entre
1 y 5 al puntero.
 cualquierColor() # Elige un color al azar
 cualquierLugar() # Elige un lugar al azar
 cualquierDireccion() # Elije una dirección al azar.
 stamp() # Imprime la forma en la ventana.
 i += 1
```

Después de guardar el archivo, podés ejecutarlo desde IDLE, presionando la tecla “F5”.

< Cierre >

El docente pide a los estudiantes que se agrupen para probar sus proyectos. Si ven que algo del programa no funciona, trabajan juntos para encontrar el error y solucionarlo.

Luego, se sugiere hacer una puesta en común para intercambiar sobre lo que aprendieron, lo que más les gustó hacer y sobre los desafíos que se les presentaron.

Evaluación:
El docente puede evaluar el proyecto tanto a través de la observación, durante el desarrollo de las actividades, como en relación al programa final. Puede copiar los proyectos y evaluarlos en detalle, pero también puede utilizar la instancia de puesta en común.

Se tendrán en cuenta los objetivos específicos de la actividad, como otros aspectos vinculados a la creatividad, la cooperación entre pares y el aprendizaje a partir de la exploración y el error.

A continuación, se presentan preguntas orientadoras:

- ¿Logró resolver los desafíos propuestos?
- ¿Pudo crear las funciones de aleatoriedad?
- ¿Trabajó en grupo cuando se encontró con problemas?
- ¿Escribió el programa propuesto de forma lógica y ordenada, solucionando los errores de sintaxis?
- ¿Creó alguna forma geométrica nueva?
- ¿Comprendió la lógica de los bucles que dibujan las formas?

El proceso de evaluación de la evolución del aprendizaje podrá continuar con la actividad propuesta a continuación.

Para seguir aprendiendo

Se puede construir sobre este mismo programa, agregando funcionalidades. Una posibilidad es agregar un contador de movimientos y competir a ver quién lo resuelve con la menor cantidad de movimientos.
También se pueden utilizar los conocimientos adquiridos en la actividad 5 para subir nuestras creaciones propias, o descargadas de internet como formas del puntero. De esta manera se puede realizar la actividad con formas más complejas.

Finalmente sugerimos al docente plantear una actividad en grupo, en la que, con la ayuda de los estudiantes, se piense en un programa que resuelva una problemática en particular y que se valga de las herramientas aprendidas en este ejercicio (creación de funciones, utilización de números al azar, selección de elementos al azar, dibujo de formas en pantalla, manipulación del puntero, utilización de bucles, etc.).
El proyecto será completamente libre, para que puedan aplicar los aprendizajes construidos.