Colección de actividades Aprender Conectados

Nivel Secundario

Programación

Perdido en el Ciberespecio

Actividad N° 5

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. • El reconocimiento y la aplicación de los derechos de la propiedad intelectual —incluyendo el manejo específico de diferentes tipos de licencia- para producciones digitales propias y de otros.

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:

- Integrar creaciones propias dentro de nuestros programas.
- Interactuar en tiempo real con nuestro programa.
- Realizar un proyecto que reciba comandos por teclado y los transforme en un movimiento personalizado de un objeto dentro de una ventana con componentes gráficos.

Materiales y recursos

Computadora.

Python 2.x instalado.

Desafío

Nico, de Somos Digitales, está perdido en el ciberespacio una vez más.
Nuestro desafío es dibujar un laberinto que represente el ciberespacio, donde está perdido Nico, a quien tendremos que ayudar a salir mediante comandos desde el teclado.

< Inicio >

Disparador

¿Sabías que podés crear tus propios juegos de manera muy fácil con Python?

En esta actividad debemos ayudar a Nico a encontrar la salida del ciberespacio. Para esto los alumnos deberán trabajar en grupo y de forma colaborativa, diseñando y resolviendo sus propios laberintos. El desafío, que presenta cierta complejidad, puede ser resuelto fácilmente por los alumnos con ayuda del docente y mediante la descomposición del problema en partes.

A grandes rasgos, el problema se descompone en:

- Crear la imagen de un laberinto.
- Cargar esta imagen en el programa.
- Crear al personaje.
- Cargar este personaje en el programa.
- Programar el movimiento del personaje.
- Sacar al personaje del laberinto.

En este documento, las líneas de código son presentadas 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 sugiere personalizar el ejercicio de manera que los alumnos deben dibujar un laberinto utilizando Paint o algún otro programa de diseño. Es importante que, al terminar la ilustración, se guarde el archivo con extensión GIF. Como ejemplo: "Laberinto.gif". Si no se cuenta con tiempo o recursos para hacer el laberinto, se pueden utilizar los siguientes archivos para completar el ejercicio:
Laberinto:
https://imgur.com/a/OsUfS3x
Personaje Nico:
https://commons.wikimedia.org/wiki/File:SxrP8y2_-_Imgur.gif

Este laberinto fue creado a partir de la siguiente imagen, que cuentan con licencia de uso libre Creative Commons.
Fuente del Laberinto:
https://commons.wikimedia.org/wiki/File:Maze_simple.svg
La foto de Nico también cuenta con licencia CC.
Personaje Nico:
https://commons.wikimedia.org/wiki/File:SxrP8y2_-_Imgur.gif

Este tipo de licencias nos permite utilizar recursos creados por otras personas de forma libre y sin infringir ninguna ley de derechos de autor. Es importante ver cuáles son las restricciones de la licencia aplicada al documento.

La licencia CC (Creative Commons) permite el uso de las siguientes restricciones (y sus diferentes combinaciones):

 **Atribución** (*Attribution*): En cualquier explotación de la obra autorizada por la licencia será necesario reconocer la autoría (obligatoria en todos los casos).

 **No Comercial** (*Non commercial*): La explotación de la obra queda limitada a usos no comerciales.

 **Sin obras derivadas** (*No Derivate Works*): La autorización para explotar la obra no incluye la posibilidad de crear una obra derivada.

 **Compartir Igual** (*Share alike*): La explotación autorizada incluye la creación de obras derivadas siempre que mantengan la misma licencia al ser divulgadas.

Las página oficinal de Creative Commons Argentina detalla las posibles combinaciones:
http://www.creativecommons.org.ar/licencias.html

Existen otras licencias para distribuir contenido de uso libre. Algunas son:
Licencia MIT: https://opensource.org/licenses/MIT
Licencia GPL: https://www.gnu.org/licenses/licenses.es.html

Para resolver la actividad, podés crear tus propias imágenes, o buscar imágenes de licencia libre en internet.

Una vez que está listo el laberinto, el resto del ejercicio se realiza desde la interfaz de desarrollo IDLE.

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 como un programa.

< Desarrollo >

La ejecución de esta actividad se puede dividir en los siguientes pasos:

1. Importación de librerías externas.
2. Configuración de la ventana.
3. Cargar el laberinto al programa.
4. Creación del personaje y sus atributos.
5. Creación de funciones para el movimiento.
6. Asignación de las teclas a las funciones.
7. Posicionamiento inicial e inicialización.

El docente recuerda a los alumnos acerca de la importancia de utilizar los comentarios e invita 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
# Perdido en el Ciberespacio
```

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

1) Importación de librerías externas

Nuestro objetivo es crear un programa que nos permita controlar un personaje para ayudarlo a escapar del laberinto que creamos previamente. Para esto nos valdremos de la librería "OS", que nos permite interactuar con el sistema operativo y "*Turtle*", una librería para crear gráficos que ya utilizamos anteriormente.

Comenzamos importando las librerías a nuestro programa:

```
import turtle
import os
```

2) Configuración de la ventana

El siguiente paso es inicializar y configurar nuestra pantalla. Crearemos una nueva ventana con *Turtle* y le daremos un tamaño de 1000 por 630 píxeles. Este es el tamaño de la imagen ejemplo que utilizamos en este ejercicio. Si creaste tu propio laberinto, debes ajustar el tamaño de la pantalla al de tu imagen.

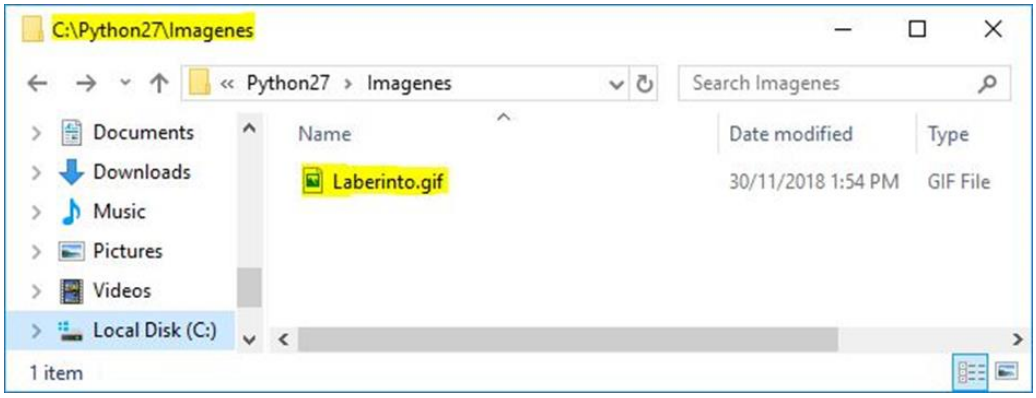
```
screen = turtle.Screen()
screen.setup(1000,630)
```

3) Cargar el laberinto al programa

A continuación, vamos a configurar nuestro laberinto como el fondo de nuestra ventana. Ya contamos con el archivo, pero: ¿cómo hacemos para cargarlo en el programa? Para resolver esto, primero tenemos que saber el directorio desde donde corre nuestro programa, ya que allí vamos a depositar el archivo para que el programa lo pueda cargar fácilmente. Obtenemos esta información, preguntando directamente al sistema operativo, el verdadero encargado del sistema de archivos sobre el que trabaja Python. Para esto, utilizamos la función "getcwd()" de la librería "OS".

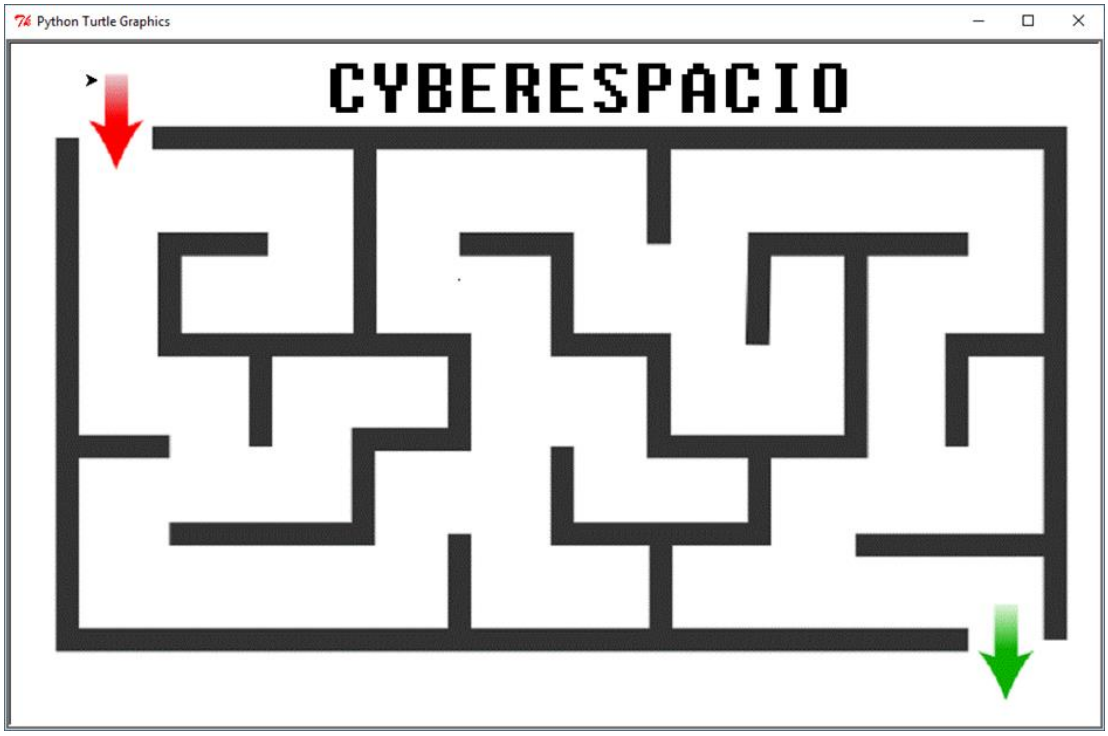
```
os.getcwd()
'C:\\Python27' # A modo de ejemplo. Puede ser otra otra ubicación
```

Con esta información, navega hasta la ubicación con tu explorador de archivos y crea allí una nueva carpeta con nombre "Imágenes". Aquí podés guardar las imágenes para tus proyectos. Copiá tu imagen de laberinto dentro de esta carpeta, con el nombre "Laberinto.gif".



Con la imagen en su lugar, vamos a concatenar la salida de "os.getcwd()" con el resto de la ubicación del archivo y guardamos el resultado en una variable. A continuación, utilizamos esa variable para definir el fondo con la función "bgpic()" de nuestra ventana:

```
archivo_fondo=os.getcwd()+"\Imagenes\Laberinto.gif"
screen.bgpic(archivo_fondo)
```



Resultado al ejecutar el código de ejemplo

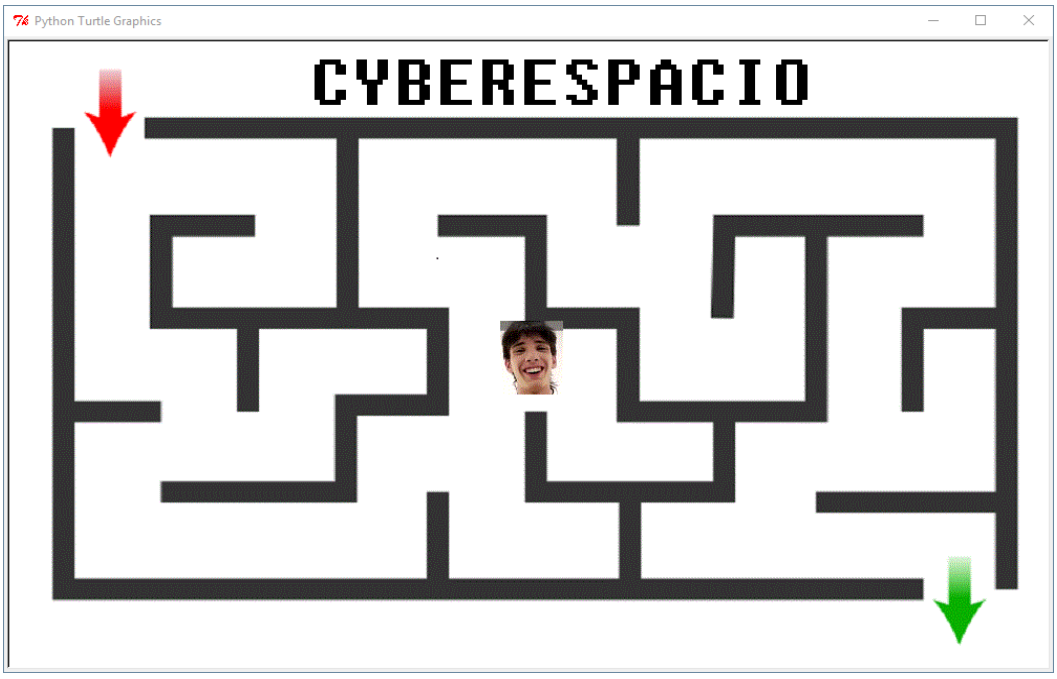
4) Creación del personaje y sus atributos

Necesitamos un personaje para escapar de nuestro laberinto. En este ejemplo, es Nico de Somos Digitales, pero podés crear y utilizar tu propio personaje.

Una vez más salvaremos el archivo con formato gif en la carpeta Imágenes. A continuación subimos nuestra imagen como una forma dentro de nuestro programa. Para esto utilizamos la función screen.register_shape(). Como último paso, definimos nuestra imagen como la forma de nuestro puntero.

```
screen.register_shape(os.getcwd()+"\Imagenes\Nico.gif")
turtle.shape(os.getcwd()+"\Imagenes\Nico.gif")
```

También se pueden utilizar las formas que trae turtle integradas.
Las formas disponibles son: 'arrow', 'turtle', 'circle', 'square', 'triangle' y 'classic'.



Resultado al ejecutar el código de ejemplo

A continuación, definimos la velocidad de movimiento y de giro de nuestra tortuga. Guardamos el valor definido en variables.

```
velocidad_de_movimiento = 10
velocidad_de_giro = 10
```

¿Qué pasa si hacemos más rápida la velocidad de movimiento o giro? ¿Y más lenta?

5) Creación de funciones para el movimiento

En programación, las funciones son segmentos de código que realizan una función determinada. Esto nos permite simplemente llamar a la función en vez de escribir todo el código cada vez que requerimos realizar esa función. Hasta el momento venimos usando funciones codificadas por otros, como “os.getcwd()”, pero llegó el momento de realizar nuestras propias funciones para definir los movimientos de nuestro personaje.

En el código a continuación definimos las funciones de “adelantar()”, “atrasar()”, “girar_a_la_izquierda()” y “girar_a_la_derecha()”. Estas funciones llaman a las funciones de movimiento de *Turtle*, especificando las velocidades establecidas en el paso anterior.

Para poder diferenciar el código que pertenece a la función del resto del código, se debe dejar un espacio al frente de cada línea que pertenezca a la función. A esta forma de ordenar el código se la llama **indentación**. Una vez que terminamos de escribir la función, dejamos una línea en blanco y continuamos la siguiente sin la indentación.

```
def adelantar(): # Definimos el nombre de la función
 turtle.forward(velocidad_de_movimiento) # Nótese el espacio al comienzo d
la línea.

def atrasar(): # Aqui comienza otra funcion. Nótese no hay más indentación
 turtle.backward(velocidad_de_movimiento)

def girar_a_la_izquierda():
 turtle.left(velocidad_de_giro)

def girar_a_la_derecha():
 turtle.right(velocidad_de_giro)
```

¿Se te ocurren otras funciones para crear? ¿Se te ocurre cómo crear una función que mueva el personaje a la derecha o a la izquierda solo con las funciones de girar y avanzar ya presentadas?

6) Asignación de las teclas a las funciones

Ahora que contamos con las funciones de movimiento definidas, las vamos a asociar a las flechas del teclado, utilizando la función “onkey()” de nuestra ventana. Esta función detecta cada vez que presionamos las teclas y ejecuta las funciones que le pasamos como parámetro.

```
screen.onkey(adelantar, "Up") # Al presionar la flecha hacia arriba,
ejecutar "adelantar()"
screen.onkey(atrasar, "Down") # Al presionar la flecha hacia abajo,
ejecutar "atrasar()"
screen.onkey(girar_a_la_izquierda, "Left") # Al presionar la flecha
izquierda, ejecutar "girar_a_la_izquierda()"
screen.onkey(girar_a_la_derecha, "Right") # Al presionar la flecha
derecha, ejecutar "girar_a_la_derecha()"
```

Podés asignar teclas a cualquier función que no reciba parámetros.

7) Posicionamiento inicial e inicialización

Como último paso, levantamos la lapicera con la función "*turtle.penup*()" para no marcar el laberinto, y nos posicionamos al comienzo del mismo. Te dejamos 5 posiciones de ejemplo para que uses la que mejor se adapte a tu laberinto personal.

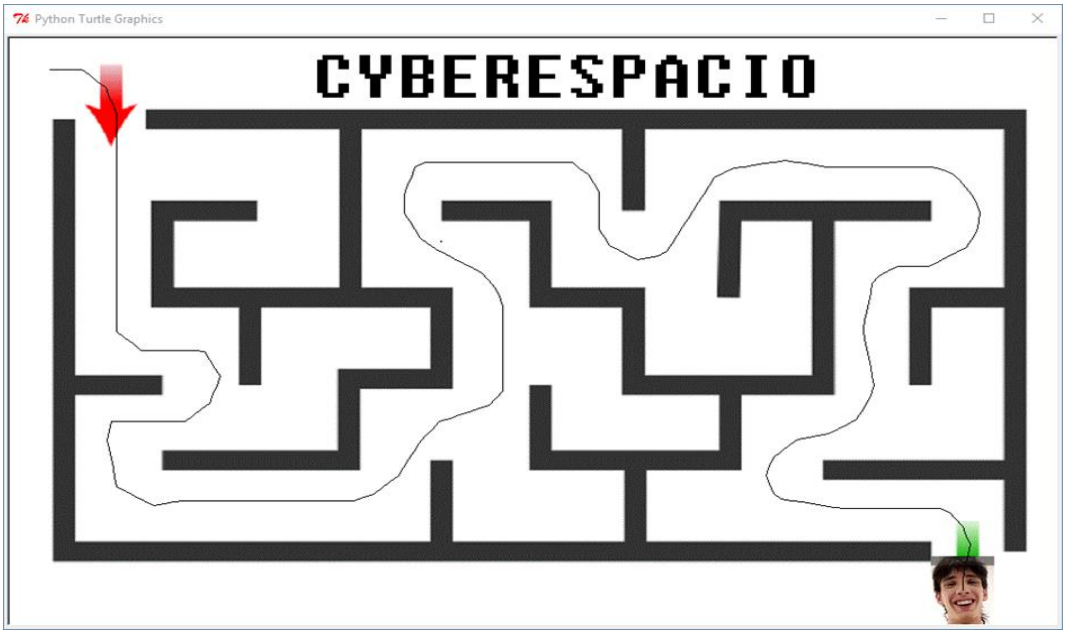
```
# Desactivamos el trazo del recorrido de nuestro personaje.
turtle.penup()

# Elige una de las opciones a continuación. la que más se adapte a tu
laberinto
turtle.goto(-480,-280) # Comienzo en esquina inferior izquierda
turtle.goto(-480,280) # Comienzo en esquina superior izquierda
turtle.goto(480,-280) # Comienzo en esquina inferior derecha
turtle.goto(480,280) # Comienzo en esquina superior derecha
turtle.home() # Comienzo en el centro de la ventana

# Si quieres que tu personaje dibuje el camino que realiza, ejecuta la
siguiente línea
turtle.pendown()
```

Ahora sí: ¡Que empiece la diversión! Le decimos a nuestra ventana que comience a escuchar eventos del teclado y ya estamos listos para llevar a nuestra imagen hasta la salida.

```
screen.listen()
```



Resultado al ejecutar el código de ejemplo

Código completo

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

```
# -*- coding: cp1252 -*-
# -*- coding: 850 -*-
# -*- coding: utf-8 -*-
# Creador: Pedro Perez
# Perdido en el Ciberespacio

# Importamos las librerías
import turtle
import os

# Configuramos nuestra ventana
screen = turtle.Screen()
screen.setup(1000,630)

# Cargamos los archivos y los definimos como fondo y personaje
os.getcwd()

archivo_fondo=os.getcwd()+"\Imagenes\laberinto.gif"
screen.bgpic(archivo_fondo)

screen.register_shape(os.getcwd()+"\Imagenes\Nico.gif")
turtle.shape(os.getcwd()+"\Imagenes\Nico.gif")

# Definimos las velocidades de movimiento y  giro de nuestro personaje

velocidad_de_movimiento = 10
velocidad_de_giro = 10

# Funciones de movimiento
def adelantar():
 turtle.forward(velocidad_de_movimiento)

def atrasar():
 turtle.backward(velocidad_de_movimiento)

def girar_a_la_izquierda():
 turtle.left(velocidad_de_giro)

def girar_a_la_derecha():
 turtle.right(velocidad_de_giro)

# Asociamos nuestras funciones a las flechas del teclado
screen.onkey(adelantar, "Up") # Al presionar la flecha hacia arriba,
ejecutar "adelantar()"
screen.onkey(atrasar, "Down") # Al presionar la flecha hacia abajo,
ejecutar "atrasar()"
screen.onkey(girar_a_la_izquierda, "Left") # Al presionar la flecha
```

```
izquierda, ejecutar "girar_a_la_izquierda()"
screen.onkey(girar_a_la_derecha, "Right") # Al presionar la
flecha derecha, ejecutar "girar_a_la_derecha()"

# Desactivamos el trazo del recorrido de nuestro personaje.
turtle.penup()

# Le damos la posición inicial a nuestro personaje dentro del
laberinto.
# (Puedes elegir una de las opciones a continuación. la que más
se adapte a tu laberinto)
#turtle.goto(-460,-280) # Comienzo en esquina inferior
izquierda
turtle.goto(-460,280) # Comienzo en esquina superior izquierda
#turtle.goto(460,-280) # Comienzo en esquina inferior derecha
#turtle.goto(460,280) # Comienzo en esquina superior derecha
#turtle.home() # Comienzo en el centro de la ventana

# Si querés que tu personaje dibuje el camino que realiza,
ejecutá la siguiente línea
turtle.pendown()

# Comenzamos a escuchar los eventos del teclado dentro de
nuestra ventana
screen.listen()
turtle.done() # Esta línea previene que la ventana se cierre al
ser ejecutado como un programa.
```

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

< Cierre >

El docente pide a los estudiantes que se agrupen de a dos, 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, para lo cual podrá acercarse a cada alumno, o bien, si se quisiera ver en detalle el código, se pueden copiar los archivos con los programas desarrollados.

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. Para evaluar esto, se le puede pedir a cada alumno que explique qué desafíos encontró y cómo los solucionó.

A continuación, se presentan preguntas orientadoras:

- ¿Logró resolver el desafío propuesto?
- ¿Pudo cargar sin dificultades la imagen al programa ?
- ¿Trabajó en grupo cuando se encontró con problemas?
- ¿Escribió el programa propuesto de forma lógica y ordenada, solucionando los errores de sintaxis, de haberse presentado?
- ¿Modificó de alguna forma su código para mejorar o modificar el del ejemplo de la actividad?
- ¿Comprendió la lógica que guía al movimiento del personaje?

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.

Para continuar, 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 pueda ser resuelto con las herramientas aprendidas en el ejercicio (utilización de imágenes propias, creación de funciones, interacción con el teclado, control por teclado y dibujo en pantalla).

El proyecto será completamente libre, para que puedan aplicar los aprendizajes construidos.

Algunos ejemplos:

1. una pista de autos para hacer carreras por tiempo;
2. un tablero de dibujo estilo Etch A Sketch;
3. un teclado visual, que al presionar distintas teclas, dibuja figuras en la ventana.