PYTHON APLICADO A LA CIENCIA Y LA INGENIERÍA

TEORÍA, SIMULACIONES Y PROYECTOS

Edward Johan Marín García
• Néstor Fabio Montoya Palacios •
José Neftalí Torres Marín



Universidad del Valle
Programa Editorial

La programación computacional se ha convertido en una habilidad fundamental para los profesionales del siglo XXI, incluyendo físicos, matemáticos e ingenieros. El lenguaje Python ha emergido como una herramienta poderosa para estos campos, que permite automatizar cálculos, visualizar datos, interactuar con instrumentos y hacer simulaciones.

Este libro busca introducir al lector en el apasionante mundo de Python aplicado a las ciencias y la ingeniería. A través de sus páginas encontrará temas tan variados como simulaciones de fenómenos físicos, lectura de sensores con Arduino y Raspberry Pi, análisis estadístico de datos y conceptos de probabilidad.

Con un enfoque práctico y muchos ejemplos paso a paso, este material sirve tanto para quien se inicia en la programación científica como para quien desea expandir sus habilidades. Al finalizar el recorrido, el lector estará en capacidad de desarrollar sus propios programas en Python para modelar sistemas físicos, adquirir y procesar datos experimentales, y comprender principios matemáticos y estadísticos.

La complejidad creciente de los problemas contemporáneos requiere de enfoques multidisciplinarios. Este libro inspira al lector a combinar conceptos de programación, física, química, matemática y electrónica para diseñar soluciones integrales como preparación para los desafíos del futuro.

¡Bienvenido a este apasionante viaje al mundo de Python en acción!

PYTHON APLICADO A LA CIENCIA Y LA INGENIERÍA

TEORÍA, SIMULACIONES Y PROYECTOS



Colección Ciencias Naturales y Exactas

Marín García, Edward Jhohan
Python aplicado a la ciencia y la ingeniería: Teoría, simulaciones y proyectos / Edward Jhohan Marín García, Néstor Fabio Montoya, José Neftalí Torres Marín - Primera edición
Cali : Universidad del Valle - Programa Editorial, 2026.
368 páginas ; 17 x 24 cm. -- (Colección: Ciencias Naturales y Exactas)
1. Python (Lenguaje de programación) – 2. Programación científica --
3. Física – 4. Matemáticas --5. Electrónica – 6. Estadística

005.13 CDD. 22 ed.
M337

Universidad del Valle - Biblioteca Mario Carvajal

Universidad del Valle
Programa Editorial
Título: Python aplicado a la ciencia y la ingeniería: Teoría, simulaciones y proyectos
Autores: Edward Jhohan Marín García, Néstor Fabio Montoya, José Neftalí Torres Marín
ISBN-POD: 978-958-507-595-5
ISBN-Pdf: 978-958-507-596-2
DOI: 10.25100/peu.997
Colección: Ciencias Naturales y Exactas
Primera edición

© Universidad del Valle
© autores

Diseño de caratula: Angela Maria Arboleda
Diagramación: Danny Stivenz Pacheco Bravo
Corrección de estilo: Daniel Hurtado Huaca

Esta publicación fue sometida al proceso de evaluación de pares externos para garantizar altos estándares académicos.
El contenido de esta obra corresponde al derecho de expresión del autor y no compromete el pensamiento institucional de la Universidad del Valle, ni genera responsabilidad frente a terceros. El autor es el responsable del respeto a los derechos de autor y del material contenido en la publicación, razón por la cual la universidad no puede asumir ninguna responsabilidad en caso de omisiones o errores.

Prohibida la reproducción total o parcial en cualquier forma, o por cualquier medio, sin autorización escrita de la Universidad del Valle.

Cali, Colombia, octubre de 2026

PYTHON APLICADO A LA CIENCIA Y LA INGENIERÍA

TEORÍA, SIMULACIONES Y PROYECTOS

Edward Jhohan Marín García
• Néstor Fabio Montoya Palacios •
José Neftalí Torres Marín



Colección Ciencias Naturales y Exactas

PREFACIO

El avance acelerado de la tecnología ha revolucionado todos los campos del conocimiento, desde la ciencia hasta la ingeniería, demandando nuevas herramientas que permitan abordar los desafíos de un mundo cada vez más complejo. Este libro nace con el propósito de ofrecer una guía comprensible y accesible para aquellos que desean sumergirse en el fascinante universo de Python, una de las herramientas más versátiles y poderosas del siglo XXI, aplicada en áreas como la física, la matemática y la ingeniería.

Python, un lenguaje de programación de alto nivel y fácil de aprender, se ha convertido en una herramienta clave para investigadores, profesionales y estudiantes que buscan analizar, modelar y resolver problemas reales de manera eficiente. A lo largo de estas páginas, el lector descubrirá cómo utilizar este lenguaje para crear simulaciones dinámicas, realizar cálculos complejos, visualizar fenómenos físicos y analizar grandes volúmenes de datos, todo con un enfoque práctico y orientado a la solución de problemas concretos.

El texto que tiene en sus manos no es un manual de programación convencional. Su enfoque va más allá de las meras instrucciones para ejecutar código: busca integrar conceptos científicos y de ingeniería con la programación, brindando al lector un entendimiento profundo de cómo aplicar Python para resolver problemas específicos en distintas disciplinas. A través de una combinación de teoría, ejemplos detallados y proyectos prácticos, se promueve el aprendizaje activo y la capacidad de aplicar el conocimiento adquirido en situaciones reales.

Uno de los grandes atractivos de este libro es la inclusión de VPython, una poderosa extensión de Python que permite crear visualizaciones en 3D. Con esta herramienta, los conceptos abstractos de la física y las matemáticas toman forma, lo que facilita su comprensión y permite a los usuarios experimentar de manera interactiva con los problemas que están resolviendo. Ya sea simulando el movimiento armónico de un péndulo, las ondas estacionarias en una cuerda tensa o el efecto Doppler, las visualizaciones creadas con VPython ofrecen una experiencia de aprendizaje envolvente.

Este enfoque interdisciplinario también se refleja en los proyectos propuestos a lo largo del libro, donde se invita a los lectores a aplicar lo aprendido en situaciones prácticas. Estos proyectos, cuidadosamente diseñados, van desde simulaciones de fenómenos físicos hasta análisis estadísticos complejos, proporcionando un entorno de experimentación que fomenta la creatividad y el pensamiento crítico.

En resumen, este libro es una puerta de entrada al vasto mundo de Python aplicado a la ciencia y la ingeniería. Está dirigido tanto a estudiantes como a profesionales que buscan una herramienta poderosa para resolver problemas técnicos de manera eficiente. No importa si el lector es nuevo en el mundo de la programación o si ya cuenta con experiencia previa, los conceptos y proyectos aquí presentados están estructurados de manera que todos puedan beneficiarse, avanzar a su propio ritmo, y lo más importante: desarrollar una comprensión sólida de cómo Python puede transformar la manera en que abordamos los desafíos científicos y de ingeniería.

CONTENIDO

CAPÍTULO 1
DIBUJOS, GRÁFICAS Y MÁS EN PYTHON, APLICADOS A PROBLEMAS DE FÍSICA Y MATEMÁTICAS
Explorando el poder de VPython: simulación y programación científica aplicada a fenómenos físicos y matemáticos 13
Introducción del capítulo 13
Objetivos del capítulo 14
Contenido del capítulo 15
Escena y objetos 16
Dibujo de puntos 25
Dibujo de curvas 29
Dibujo de cilindros 48
Dibujo de cajas 52
Dibujo de flechas 56
Funciones matemáticas 65
Etiquetas 66
Movimiento armónico simple 68
Gráficos en VPython 73
Tiro oblicuo con resistencia del aire 84
Péndulo simple 96
Dibujo de hélices 102
Oscilaciones amortiguadas 103
Problema de la braquistócrona 110
Objeto anillo 127
Shapes 131
Transmisión del movimiento circular uniforme 158
Figuras de Lissajous o curvas de Bowditch 162
Difracción de la luz por una ranura rectangular 169
Difracción producida por una rendija delgada 170
Conclusiones del capítulo. 186
Proyectos propuestos del capítulo 187

CAPÍTULO 2
MENDELEEV Y SUS APLICACIONES EN LA QUÍMICA Y LA ELECTRÓNICA
Explorando las aplicaciones del paquete Mendeleev en la intersección de la química y la electrónica: acceso y análisis de las propiedades de los elementos para innovaciones tecnológicas 191
Introducción del capítulo 191
Objetivos del capítulo 193
Contenido del capítulo 194
Mendeleev 194
Uso interactivo básico 195
Atributos extendidos 199
Estados de oxidación 199
Energías de ionización 200
Isótopos 201
Configuración electrónica 203
Número cuántico principal 204
Funciones útiles para calcular propiedades 204
Dibujo de la tabla periódica 205
Conclusión del capítulo. 209
Proyectos del capítulo 210

CAPÍTULO 3
PYTHON Y LA INTERACCIÓN CON ARDUINO
Potenciando el desarrollo electrónico: La sinergia entre Python y Arduino para el desarrollo de proyectos tecnológicos innovadores **213**
Introducción del capítulo 213
Objetivos del capítulo 215
Contenido del capítulo 216
Arduino 216
Características técnicas 218
Pines y sus funciones 218
Firmata 220
Controlar Arduino con la librería PySerial 226
Sensor de temperatura y humedad DHT11 230
Comunicación con Python 234
Sensor de ultrasonido HC-SR04 237
Conclusiones del capítulo. 245
Proyectos del capítulo 247

CAPÍTULO 4
RASPBERRY PI Y SUS APLICACIONES EN LA ELECTRÓNICA
Explorando el poder de Raspberry Pi: desde proyectos educativos hasta aplicaciones avanzadas en electrónica y robótica 249
Introducción del capítulo 249
Objetivos del capítulo 251
Contenido del capítulo 252
Raspberry Pi 252
Control de GPIO con Python en Raspberry Pi 253
Pines especiales de la GPIO 255
Conclusiones del capítulo 259
Proyectos del capítulo 261

CAPÍTULO 5
PYTHON EN APLICACIONES DE MATEMÁTICA COMBINATORIA . 263
Python como herramienta clave en la resolución eficiente de problemas de combinatoria mediante el uso de la biblioteca Itertools 263
Introducción del capítulo 263
Objetivos del capítulo 265
Contenido del capítulo 266
Módulo Itertools 266
Combinatoria 270
Conclusiones del capítulo 287
Proyectos del capítulo 289

CAPÍTULO 6
ESTADÍSTICA 291
Explorando la estadística descriptiva e inferencial con Python: de los conceptos básicos a la implementación práctica 291
Introducción del capítulo 291
Objetivos del capítulo 293
Contenido del capítulo 293
Estadística con Python 294
Estadística 294
Estadística descriptiva 295
Estadística inferencial 313
Conclusiones del capítulo 359
Proyectos del capítulo 361

REFERENCIAS 365

LOS AUTORES 369

CAPÍTULO 1

DIBUJOS, GRÁFICAS Y MÁS EN PYTHON, APLICADOS A PROBLEMAS DE FÍSICA Y MATEMÁTICAS

Explorando el poder de VPython: simulación y programación científica aplicada a fenómenos físicos y matemáticos

Introducción del capítulo

Este primer capítulo se adentra en el fascinante mundo de VPython, una herramienta revolucionaria diseñada para dar vida a conceptos de física y matemáticas mediante la creación de visualizaciones 3D interactivas. Iniciando con un recorrido histórico que destaca el origen y la evolución de VPython, se establece el contexto que subraya la importancia de esta herramienta en la educación y la investigación científica. La instalación y la configuración inicial se presentan de manera detallada, asegurando que incluso aquellos sin experiencia previa en programación puedan comenzar a explorar sus posibilidades sin demora.

A través de ejemplos claros y didácticos, el capítulo guía a los lectores en el proceso de crear objetos tridimensionales básicos, tales como esferas y cajas, que son los cimientos para construcciones más complejas y animaciones dinámicas. Al introducir conceptos de animación, se enfatiza cómo VPython facilita no solo la comprensión de fenómenos físicos a través de la visualización, sino también el desarrollo del pensamiento computacional y la resolución de problemas. Con un enfoque en la aplicación práctica, se exploran diversos ejemplos que van desde la representación de movimientos armónicos hasta la simulación del efecto Doppler, cada uno acompañado de código fuente y explicaciones paso a paso.

Este método de aprendizaje activo no solo ayuda a solidificar el conocimiento teórico, sino que también estimula la curiosidad y el deseo de explorar más allá de los límites del salón de clases.

Este capítulo, por lo tanto, no solo sirve como una introducción a VPython, también se convierte en una puerta de entrada hacia una comprensión más profunda de la física y las matemáticas. Al finalizar, los lectores no solo habrán adquirido habilidades fundamentales en programación y visualización científica, sino que también estarán preparados para aplicar estos conocimientos en la resolución de problemas complejos, fomentando así un aprendizaje integral y aplicado. El capítulo se estructura en secciones dedicadas a aspectos fundamentales de VPython y sus aplicaciones. Comienza con una introducción a VPython, incluyendo su instalación y configuración básica, lo que permite a los lectores preparar su entorno de desarrollo. A continuación, se aborda la creación de objetos 3D, como esferas y cajas, y se explican las técnicas para manipular sus propiedades y posiciones. Se profundiza en la animación de estos objetos, enseñando a los usuarios a simular movimiento y cambios en el tiempo. También se incluyen ejercicios prácticos sobre dibujo de puntos y líneas, generación de curvas y representación de ondas armónicas, culminando en una discusión sobre el efecto Doppler y su simulación. Cada sección está diseñada para construir sobre la anterior, guiando al lector desde conceptos básicos hasta aplicaciones más avanzadas, y proporcionando una comprensión completa de las posibilidades que VPython ofrece para la visualización científica.

Objetivos del capítulo

- **Comprender el uso y la configuración de VPython:** el estudiante será capaz de instalar, configurar y utilizar VPython para crear objetos 3D y animaciones aplicadas a problemas de física y matemáticas.
- **Desarrollar habilidades en programación gráfica:** a través de ejemplos prácticos, el estudiante aprenderá a manipular objetos tridimensionales y aplicar estos conocimientos para resolver problemas científicos.

- **Simular fenómenos físicos:** los estudiantes podrán realizar simulaciones de movimientos armónicos y el efecto Doppler utilizando VPython, comprendiendo mejor estos fenómenos mediante la visualización.
- **Resolver problemas mediante animaciones:** el estudiante aprenderá a representar soluciones gráficas y animadas de problemas físicos aplicando el conocimiento teórico a la programación visual.
- **Desarrollar proyectos científicos aplicados:** mediante VPython, los estudiantes podrán desarrollar proyectos complejos en los campos de la física y las matemáticas, mejorando su capacidad de resolución de problemas computacionales.

Contenido del capítulo

- Visualización y animación 3D con VPython.
- Creación de objetos 3D: esferas y cajas.
- Animación de objetos en VPython.
- Representación de movimientos armónicos.
- Simulación del efecto Doppler.
- Visualización de ondas armónicas.
- Dibujo de puntos y curvas en VPython.
- Conclusiones.
- Proyectos prácticos con VPython

Visualización y animación 3D con VPython

El módulo VPython es un recurso que permite construir animaciones tridimensionales. VPython fue creado por David Scherer en 2000. En 2011, David Scherer y Bruce Sherwood iniciaron el desarrollo de GlowScript, un entorno de programación similar pero que se ejecuta en un navegador. Originalmente, los programas tenían que estar escritos en JavaScript, pero en 2014 fue posible utilizar Python; En la actualidad, los programas de GlowScript se pueden escribir en Python o en JavaScript. VPython también se puede usar con una instalación de Python, gracias al trabajo originado por John Coady en 2014 y que aún se encuentra en continuo desarrollo por John Coady, Ruth Chabay, Bruce Sherwood, Steve Spicklemire y Matt Craig. Utiliza la sintaxis de GlowScript VPython, pero

con Python, lo que proporciona acceso a los módulos estándar de Python (Scherer *et al.*, s.f.).

El sitio de referencia de VPython es https://vpython.org, donde se encuentra mucha información, ejemplos e instrucciones para descargar e instalar el módulo, el cual está disponible de forma gratuita para Windows, Macintosh y Linux. La versión más actualizada es VPython 7. Además, el proyecto VPython se centra actualmente en el uso de una función llamada GlowScript que permite escribir y editar un programa directamente en el navegador, y también es capaz de generar manera automática, a partir de un programa VPython, un archivo HTML 5 en el que toda la parte dinámica (animación, etc.) se implementa en código JavaScript.

Para utilizar GlowScript en línea, visite http://www.glowscript.org. Una vez registrado, se puede acceder al área donde se crean y guardan los programas. Para instalar VPython en Anaconda se escribe la terminal de *conda*:

conda install -c vpython vpython

Para hacer la instalación en Python se escribe en la terminal CMD:
pip install vpython

Escena y objetos

Los recursos de programación para la visualización 3D están contenidos en un módulo llamado VPython (en versiones anteriores, el nombre del paquete era Visual). GlowScript importa automáticamente el módulo, no es necesario ingresar un comando de importación; sin embargo, si se trabaja en Spyder o en otro entorno de programación, sí hay que importar el módulo:

*from vpython import **

Para demostrar las características básicas que ofrece VPython para la visualización 3D, es necesario crear un objeto. Por ejemplo, el comando `box ()` crea una «caja» (en realidad, un cubo) (Qiu, 2020). Si escribe esta línea de comando en la ventana de edición del programa y la ejecuta, aparecerá la imagen que se muestra en la Figura 1.

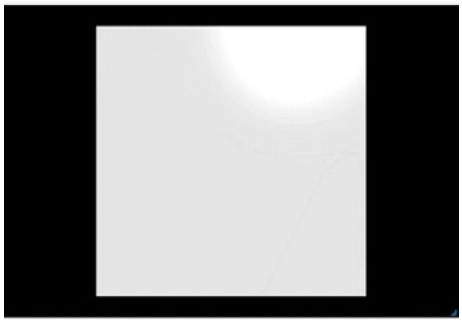


Figura 1. Dibujo de una caja en VPython

Como tal, no parece una ventana de visualización 3D, ya que solo muestra un cuadrado gris, pero si se arrastra el *mouse* sobre la ventana, con el botón derecho presionado, se puede modificar la vista y verificar que la ventana muestre de hecho, un cubo, tal como se aprecia en la Figura 2.

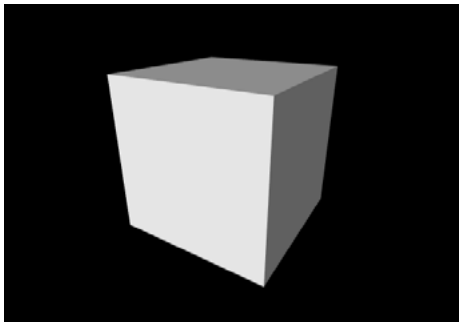


Figura 2. Caja girada con el botón derecho del mouse

Además, al mover el *mouse* sobre la ventana, hacia arriba o hacia abajo, con los dos botones presionados, puede acercar o alejar la cámara. Hay que tener presente que estas manipulaciones con el *mouse* afectan a la cámara y no a los objetos presentes en la escena (Scherer *et al.*, s. f.).

Dibujo de esferas

El siguiente código crea una esfera de color azul centrada en el punto (0,0,0), tal como se muestra en la Figura 3.

particula = sphere(pos = vector(0,0,0),color = color.blue)

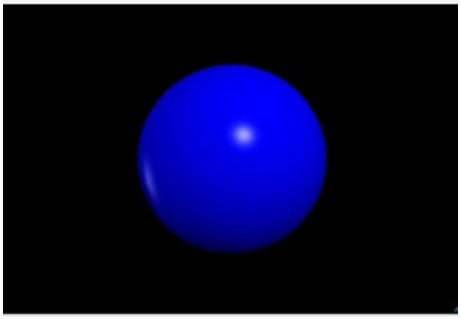


Figura 3. Dibujo de una esfera en fondo negro

Por defecto, el fondo de la ventana donde se muestran los gráficos es de color negro (Qiu, 2020). Se puede crear un lienzo de un tamaño y color determinado con el siguiente código:

```
# Importa todas las funciones de la librería vpython
from vpython import *
# Crea una nueva ventana para gráficos 3D con un título, posición, tamaño,
# punto central, rango de visualización y color de fondo especificados.

scene = canvas(title = 'Dibujo de una esfera',x = 0,y = 0,
          width = 400, height = 400, center = vector(0,0,0),range = 20,
          background = vector(1,1,1))

# Crea una esfera en la posición especificada (en este caso, el centro de la
# ventana), con un radio y color especificados.

particula = sphere(pos = vector(0,0,0),radius = 10,color = color.blue)
```

El resultado de la simulación se muestra en la Figura 4.



Figura 4. Dibujo de una esfera de fondo blanco

El código se puede digitar directamente en https://www.glowscript.org, en la pestaña «Create New Program». También puede descargarse una versión portable en https://github.com/vpython/glowscript

Usando VPython se plantea el siguiente código:

```
# Importa todas las funciones de la librería vpython
from vpython import *

# Crea una nueva ventana para gráficos 3D. Los parámetros especifican
# el título de la ventana ('Dibujo de una esfera'),
# la posición de la ventana en la pantalla (x=0, y=0), el tamaño de la
# ventana (width=400, height=400),
# el punto central de la vista (center=vector(0,0,0)), el rango#
# de visualización (range=20),
# y el color de fondo de la ventana (background=vector(1,1,1)).

scene = canvas(title = 'Dibujo de una esfera',
          x = 0, y = 0,
          width = 400, height = 400,
          center = vector(0,0,0),
          range = 20,
          background = vector(1,1,1))

# Crea una esfera en la posición especificada (pos=vector(0,0,0)),
# con un radio especificado (radius=10),
# y un color especificado (color=color.blue). Esta esfera se dibujará
# en la ventana creada anteriormente.

particula = sphere(pos = vector(0,0,0), radius = 10, color = color.blue)
```

Algunas aclaraciones sobre este programa:

- Se utilizó la función *canvas (...)* para configurar el área de dibujo en la que se mostrará la escena, especificando por ejemplo el color de fondo de la ventana y un título que aparecerá encima de ella. Tenga en cuenta que la escena principal se llama *scene*. Sería posible crear otras áreas de dibujo, con otros nombres, para mostrar más de una escena.
- La opción de rango, definida en la función de *canvas*, sirve para establecer la relación entre las unidades utilizadas en el programa y en la visualización en pantalla. En el ejemplo anterior se usaron centímetros para definir las dimensiones de la esfera, y la opción *range = 20* significa que la región visible en la ventana será de veinte centímetros en cada dirección, comenzando desde el centro.

- Es posible definir un color especificando las proporciones de rojo, verde y azul como componentes de un vector.

VPython utiliza un sistema de ejes tal que, por defecto (es decir, en ausencia de cambiar el punto de vista con el *mouse*), el eje *x* es horizontal, con sentido positivo hacia la derecha, el eje *y* es vertical, con sentido positivo hacia arriba, y el eje *z* es perpendicular a la pantalla, dejando la pantalla (Qiu, 2020). El origen de los ejes está en el centro de la ventana. La Figura 5 presenta la descripción anterior.

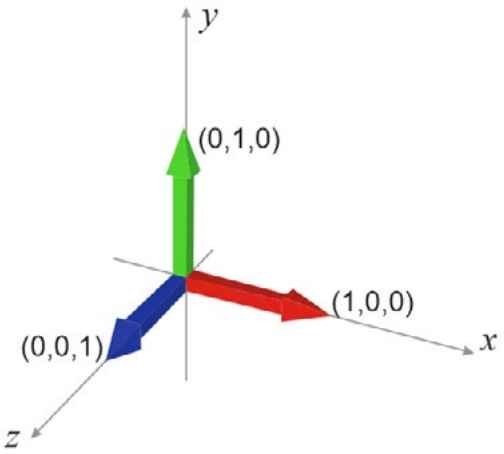


Figura 5. Ejes X, Y, Z y vectores unitarios en VPython

Se pueden asignar nombres a los objetos colocados en la escena. Esto es necesario si se necesita modificar las propiedades del objeto durante la ejecución del programa. Al crear un objeto, se puede crear con parámetros. Estos parámetros describen este objeto único. También se puede cambiar estos parámetros más adelante. Para acceder a un parámetro, se escribe el nombre del objeto, luego un punto («punto») y, a continuación, el parámetro (Qiu, 2020).

Se escribe el código usando VPython de la siguiente manera:

```
# Importa todas las funciones de la librería vpython
from vpython import *

# Crea una esfera con los valores predeterminados.
esfera = sphere()

# Cambia el color de la esfera a amarillo. Los colores se especifican
# como vectores RGB, donde cada componente del vector va de 0 a 1.
esfera.color = vec(1, 1, 0)

# Cambia el radio de la esfera a 0.5.
esfera.radius = 0.5

# Cambia la posición de la esfera a (0, 0.5, 0).
esfera.pos = vec (0, 0.5, 0)

# Crea un cubo con los valores predeterminados.
cubo = box()

# Cambia el color del cubo a azul.
cubo.color = vec(0, 0, 1)

# Cambia la opacidad del cubo a 0.5, lo que lo hace semitransparente.
cubo.opacity = 0.5
```

El código anterior es utilizado para dibujar una esfera amarilla y un cuadro de color azul, tal como se muestra en la Figura 6.

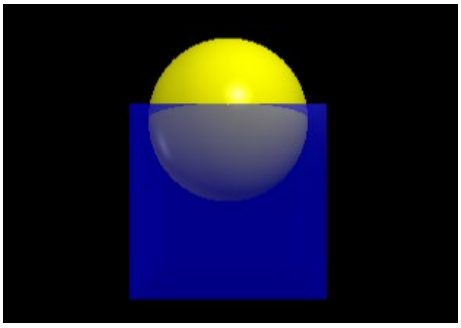


Figura 6. Dibujo de una esfera amarilla y un cubo azul

Objeto canvas

Inicialmente, hay un lienzo de VPython 3D llamado Scene. El código es:

```
lienzo = canvas(title='Ventana',
          width=600, height=200,
          center=vector(5,0,0),
          background=color.cyan)
```

Crea un lienzo con los atributos especificados de 600 pixeles de ancho por 200 pixeles de alto, con el título «Ventana», con el centro determinado por el vector (5,0,0) y con un color de fondo de relleno cian. El punto clave es que después de crear un lienzo, los objetos 3D se colocarán de forma predeterminada en el lienzo creado más recientemente, a menos que especifique explícitamente un lienzo diferente para el objeto (Qiu, 2020). A continuación, se presenta el código que permite dibujar un cubo y una esfera en dos objetos *canvas* diferentes, usando VPython de la siguiente manera:

```
# Importa todas las funciones de la librería vpython
from vpython import *

# Crea un lienzo de 200x200 píxeles, centrado en (0,0,0), con un
# fondo de color cian.
# Este lienzo se mostrará en una ventana titulada 'Ventana1'.

lienzo1 = canvas(title='Ventana1',
          width=200, height=200,
          center=vector(0,0,0),
          background=color.cyan)

# Crea otro lienzo de 200x200 píxeles, centrado en (0,0,0), con un
# fondo de color negro.
# Este lienzo se mostrará en una ventana titulada 'Ventana2'.

lienzo2 = canvas(title='Ventana2',
          width=200, height=200,
          center=vector(0,0,0),
          background=color.black)

# Crea una caja azul en el lienzo1. La caja está centrada en (0,0,0).
caja = box(canvas = lienzo1, pos = vector(0,0,0), color = color.blue)

# Crea una esfera roja en el lienzo2. La esfera está centrada en (0,0,0).
bola = sphere(canvas = lienzo2, pos = vector(0,0,0), color = color.red)
```

La Figura 7 presenta el resultado de la simulación del código anterior.

Opciones de uso general (Scherer *et al.*, s.f.):

- *width*, *height*: ancho y alto del área del lienzo en píxeles. Los valores predeterminados son 640 × 400. Puede especificarse al crear el lienzo, como en el ejemplo anterior, o establecerse o cambiarse más tarde.

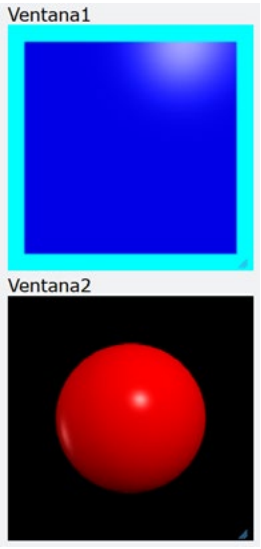


Figura 7. Dibujo de un cubo y una esfera en dos objetos canvas diferentes

- *align* (alinear): establecer en "*left*" (lienzo forzado al lado izquierdo de la ventana), "*right*" (lienzo forzado al lado derecho de la ventana) o *none* (ninguno) (la alineación predeterminada). Si tiene un solo lienzo, la configuración de alinear a la «izquierda» hace que el título del lienzo se muestre a la derecha del lienzo. Si desea colocar un gráfico a la derecha de un lienzo, establezca el atributo de alineación del lienzo en la cadena «izquierda» y el atributo de alineación del gráfico en la cadena «derecha». Si la ventana es demasiado estrecha, el objeto que está a la derecha se mostrará debajo del otro objeto. Si desea colocar un gráfico a la derecha del lienzo, pero mantener el título del lienzo debajo del lienzo, cree el gráfico primero con la alineación establecida en «derecha» y active el gráfico trazando algo en él, luego cree el lienzo sin especificar su valor de alineación. Otra opción es especificar *align = 'left'* para todos los lienzos y gráficos, en cuyo caso colindarán entre sí.
- *resizable*: si es *True* (el valor predeterminado), el usuario puede cambiar el tamaño del lienzo. Si *C* es un lienzo de tamaño variable, *C.bind ('resize',R)* ejecutará la función *R* cuando el usuario cambie el tamaño de ese lienzo. Si *R* tiene la forma *def R (ev)*, entonces el evento redimensionar será *ev.canvas*.

- *background* (fondo): establece el color que se utilizará para rellenar el lienzo; el predeterminado es negro.
- *ambient* (ambiente): color de la iluminación no direccional ("*ambient*"). El valor predeterminado es *color.gray (0.2)*.
- *lights* (luces): lista de objetos de luz creados para este lienzo. Por defecto, s es esta lista:
  [distant_light (direction = vec (0.22, 0.44, 0.88), color = color.gray (0.8)),
  distant_light (direction = vec (-0.88, -0.22, -0.44), color = color.gray (0.3))]

Son equivalentes a las luces predeterminadas que se crean automáticamente. Las luces se pueden cambiar solo después de crear el lienzo. Se puede eliminar las luces configurando *scene.lights = []*, una lista vacía, luego se puedes crear luces diferentes. Para cambiar el color de la primera de las dos luces predeterminadas, haga esto, donde 0 se refiere a la primera de las dos luces:

scene.lights [0] .color = color.red

- *objects* (objetos): obtiene una lista de todos los objetos visibles en el lienzo; los objetos invisibles y las luces no se enumeran (*scene.lights* es una lista de las luces existentes). Por ejemplo, el siguiente código en VPython hace que todos los cuadros visibles en la escena sean rojos:

```
for obj in scene2.objects:
         if isinstance(obj,box):
         obj.color = color.red
```

- *visible*: configurar *scene.visible = False* significa que no se muestran objetos, hasta que *scene.visible* se establezca de nuevo en True.
- *delete ()*: elimina todos los objetos en este lienzo y luego elimina el lienzo en sí.
- *capture(filename)*: envía a su carpeta de descargas una captura de pantalla en formato png del lienzo. Si el nombre del archivo es la cadena «boxes» o «boxes.png», el archivo se denominará «boxes.png». Si ejecuta *scene.capture (cuadros)* repetidamente,

los archivos adicionales se denominarán «cuadros(1).png», «cuadros (2).png», etc. Porque los objetos de etiqueta se muestran en un lienzo transparente 2D separado delante del lienzo 3D, las etiquetas no se muestran en una escena capturada. Si necesita que el texto sea parte de la escena capturada, use el objeto de texto 3D.

- Para obtener la ubicación actual de la cámara, consulte a continuación los detalles de *scene.camera.pos*.
- *camera.follow*: si se escribe *scene.camera.follow (bola)*, el centro de la escena se restablecerá continuamente a la posición actual del objeto bola. Para dejar de seguir el objeto, ejecute *scene.camera.follow (None)*. En lugar de especificar un objeto a seguir, puede proporcionar una función propia:

```
def f():
        return 2*ball.pos
        scene.camera.follow(f)
```

- *pixel_to_world*: da el ancho de un píxel en coordenadas mundiales (es decir, las coordenadas que usa para colocar objetos). Esto es de solo lectura, no puede configurarlo. Se determina a partir del valor actual de *scene.range*. Un ejemplo de su uso es que, si desea que el radio de un objeto de curva sea de 5 píxeles, establezca el radio en *5*scene.pixel_to_world*.

  Aquí hay opciones de lienzo para esperar las actualizaciones de lienzo:

  · *scene.waitfor ("redraw")*: espera el inicio de la próxima actualización del lienzo por parte del navegador web.
  · *scene.waitfor ("draw_complete")*: espera el final de la próxima actualización del lienzo por parte del navegador web.

Dibujo de puntos

Para dibujar un punto se utiliza la instrucción *points* y como parámetro se especifica su posición y el radio. El radio predeterminado de un punto es 2,5, es decir, una esfera cuyo radio es 2,5 píxeles (un diámetro de 5 píxeles). El atributo *radius* está en píxeles de pantalla si s*ize_units* =

"píxeles" (valor predeterminado), pero *si size_units = "world"*, el radio está en los mismos términos que otros objetos. Establecer el radio en 0 es lo mismo que no especificar un tamaño, en cuyo caso el radio será de 2,5 píxeles. Si *size_units = "world"*, el radio predeterminado es 0,5 pixeles (Scherer *et al.*, s.f.).

- Con *size_units = "píxeles"* (valor predeterminado), un punto en el plano z = 0 tendrá el radio especificado, pero se verá más grande o pequeño que el radio especificado si está más cerca o más lejos de la cámara.
- A diferencia del objeto *curve* (curva), un objeto de puntos puede ser transparente (opacidad <1).

A continuación, se presenta la implementación del código utilizando GlowScript 3.1 en el entorno de VPython.

```
P1 = points(pos=vector(0,0,0), color=color.red)
P2 = points(pos=vector(1,0,0), color=color.blue)
P3 = points(pos=vector(1,1,0), color=color.yellow)
P4 = points(pos=vector(0,1,0), color=color.white)
P5 = points(pos=vector(0,1,1), color=color.orange)
P6 = points(pos=vector(0,0,1), color=color.green)
P7 = points(pos=vector(1,0,1), color=color.cyan)
P8 = points(pos=vector(1,1,1), color=color.purple)
```

La Figura 8 presenta la descripción anterior.

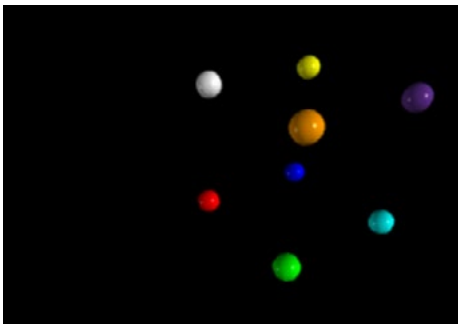


Figura 8. Dibujo de ocho puntos en las esquinas de un cubo

Se pueden crear los puntos por medio de una lista. El código para esto se obtiene de la siguiente manera:

```
# Crea un vector de colores aleatorios. random() genera un
# número aleatorio entre 0 y 1.
colores = vector(random(),random(),random())

# Crea un conjunto de puntos en las posiciones especificadas.
# Los puntos son de tamaño "world" (lo que significa que su tamaño
# no cambia con la perspectiva)
# y tienen un radio de 0.1. Todos los puntos tienen el mismo
# color, definido por el vector de colores aleatorios.
puntos = points(pos=[vector(0,0,0),      #Posición del primer punto
                     vector(1,0,0),      #Posición del segundo punto
                     vector(1,1,0),      #Posición del tercer punto
                     vector(0,1,0),      #Posición del cuarto punto
                     vector(0,1,1),      #Posición del quinto punto
                     vector(0,0,1),      #Posición del sexto punto
                     vector(1,0,1),      #Posición del séptimo punto
                     vector(1,1,1)],     #Posición del octavo punto
                     radius = 0.1,       #Radio de cada punto
                     size_units="world",  #Unidades de tamaño de
                                         #cada punto
                     color = colores     #Color de cada punto
                     )
```

La Figura 9 presenta el dibujo de los ocho puntos usando una lista de posición.

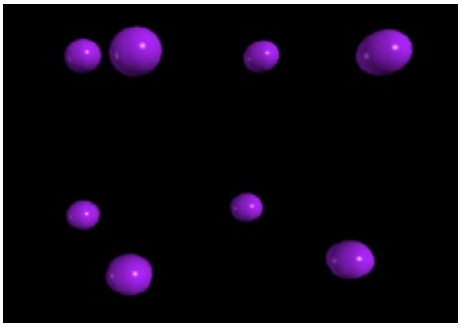


Figura 9. Dibujo de los ocho puntos por medio de una lista

Ejemplo 1.1: realice la gráfica de la función $z = sen\sqrt{x^2 + y^2}$ usando GlowScript 3.1 en el entorno de VPython.

Para darle solución al ejemplo, el código desarrollado es el siguiente.

```
# Importamos las funciones necesarias de vpython y math
from vpython import *
from math import sqrt, sin

# Definimos una función que toma dos argumentos, x e y, y devuelve el seno
de la raíz cuadrada
# de la suma de sus cuadrados.
def f(x,y):
    r =sqrt(x**2 + y**2) #Calcula la raíz cuadrada de la suma
                         #de los cuadrados de x e y.
    f = sin(r)   #Calcula el seno de r.
    return f     #Devuelve el valor calculado.

# Creamos una lista vacía para almacenar los puntos.
lista = []

#Iteramos sobre los valores de x de -4 a 4 en incrementos de 0.1.
for x in range(-4,4,0.1):
#Iteramos sobre los valores de y de -4 a 4 en incrementos de 0.1.
        for y in range(-4,4,0.1):

#Añadimos un vector a la lista.
#El vector tiene x e y como sus dos primeras componentes, y el
#valor de la función f en (x, y) como su tercera componente.
        lista.append(vector(x,y,f(x,y)))

#Dibujamos los puntos en la lista con radio=0.1 y un color azul.
points(pos=lista, color=color.blue,radius=0.1,size_units="world")
```

La Figura 10 presenta la gráfica de la función requerida en el Ejemplo 1.1.

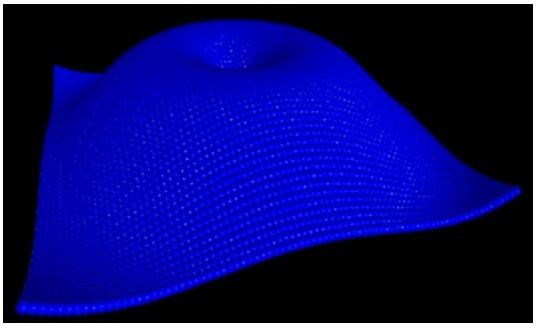


Figura 10. Gráfica de la función $z = sen\sqrt{x^2 + y^2}$ para el Ejemplo 1.1

Dibujo de curvas

El objeto *curve* muestra líneas rectas entre puntos, y si los puntos están lo suficientemente cerca, se obtiene la apariencia de una curva suave.

Detalles del objeto curve

El ejemplo más simple de una curva es la siguiente declaración, que dibuja una línea recta blanca de longitud 2, entre las posiciones (-1,-1,0) y (1,-1,0):

c = curve (vector (-1,-1,0),vector (1,-1,0))

Se puede agregar puntos adicionales a esta curva de esta manera, completando un cuadrado de ancho 2 cuyo centro está en la ubicación (0,0,0).

c.append (vector (1,1,0),vector (-1,1,0),vector (-1,-1,0))

Se desarrolla el código usando VPython para dibujar un cuadrado:

```
c = curve (vector (-1,-1,0), vector (1,-1,0))
cuadrado = c.append(vector (1,1,0),
          vector (-1,1,0),
          vector (-1,-1,0))
```

La Figura 11 presenta el resultado de la simulación.



Figura 11. Dibujo de un cuadrado

Para dibujar el cuadrado, también se puede así:

```
c = curve()
cuadrado = c.append(
          vector(-1,-1,0),
          vector(1,-1,0),
          vector(1,1,0),
          vector(-1,1,0),
          vector(-1,-1,0)
          )
```

Suponga que se tiene dos vectores llamados *v1* y *v2*. Cualquiera de los siguientes códigos dibujará una línea recta blanca desde la posición *v1* hasta la posición *v2*:

- *c = curve(v1,v2)*
- *c = curve([v1,v2])*
- *c = curve(pos=[v1,v2])*
- *c = curve(v1)c.append(v2)*
- *c = curve()c.append(v1,v2)*

Ejemplo 1.2: dibuje un cuadrado en el plano *xy* con el método *curve(pos=[v1,v2])*, usando GlowScript 3.1 en el entorno de VPython.

Para darle solución al Ejemplo 1.2, se realiza el siguiente código:

```
cuadrado = curve(pos=[vector(0,0,0),
          vector(1,0,0),
          vector(1,1,0),
          vector(0,1,0),
          vector(0,0,0)]
          )
```

La Figura 12 presenta el cuadrado en una posición preestablecida por el usuario.

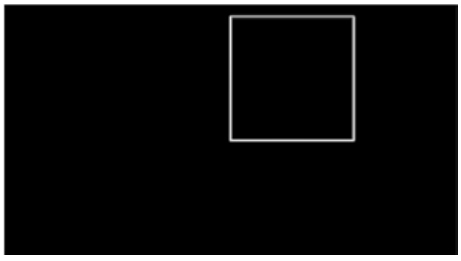


Figura 12. Dibujo de una cuadrado con el método *c = curve(pos=[v1,v2])*

Las curvas pueden tener grosor, especificado por el radio de una sección transversal de la curva (la curva tiene un grosor o diámetro que es el doble de este radio). A continuación, se presenta la implementación del código utilizando el entorno de VPython:

```
cuadrado = curve(pos=[vector(0,0,0),
          vector(1,0,0),
          vector(1,1,0),
          vector(0,1,0),
          vector(0,0,0),
          radius = 0.05])
```

La Figura 13 presenta el cuadrado con la modificación del grosor.



Figura 13. Grosor de un objeto curve por medio del parámetro radius

El radio predeterminado es 0, que dibuja una curva delgada. Un radio distinto de cero crea una curva «gruesa», pero un radio muy pequeño puede formar una curva demasiado delgada para verla.

Especificar el color y el radio en forma general

Por defecto, una curva es blanca, con un radio de sección transversal de solo unos pocos píxeles (Scherer *et al.*, s.f.). Se puede cambiar el color y el radio predeterminados para todos los puntos, utilizando cualquiera de los siguientes esquemas:

- *c = curva (pos = [v1,v2],color = color.cyan,radio = 0.3)*
- *c = curva (color = color.cyan,radio = 0.3)c.append (v1,v2)*

Especificar el color y el radio de puntos individuales

Se puede especificar el color y el radio de puntos individuales que anulan el color y el radio especificados para la curva como un todo (Scherer *et al.*, s.f.). La siguiente secuencia creará dos diccionarios para usar como puntos individuales, luego los usará para dibujar una línea cuyo color se difumina de rojo a azul y cuyo radio aumenta de 0,1 a 0,3. La solución se desarrolla haciendo uso de las capacidades visuales y computacionales de VPython, así:

```
v1 = vector(0,0,0)
v2 = vector(10,0,0)
p1 = {‘pos’:v1, ‘color’:color.red,  ‘radius’:0.1}
p2 = {‘pos’:v2, ‘color’:color.blue, ‘radius’:0.3}
curve(p1, p2)
```

La Figura 14 es el resultado del uso de color y el grosor en un objeto establecido.

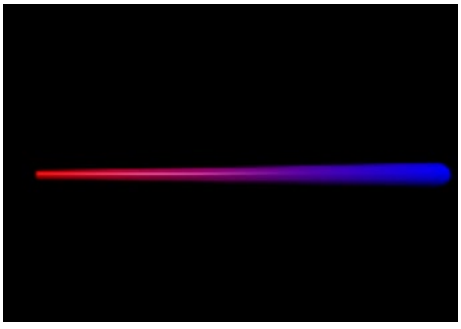


Figura 14. Modificación del color y el radio del objeto curve

Ejemplo 1.3: realizar la gráfica de la función seno en el intervalo [-2π, 2π], usando GlowScript 3.1 en el entorno de VPython.

Para darle solución al ejemplo se plantea el siguiente código:

```
# Definimos el incremento dx como pi/50
dx = pi/50

# Inicializamos x en -2*pi
x = -2*pi

# Creamos una lista vacía para almacenar los puntos
lista = []
```

```
# Mientras x sea menor o igual a 2*pi
while (x <= 2*pi):

# Añadimos un vector a la lista. El vector tiene x como su primera
# componente, el seno de x como su segunda componente, y 0 como su
# tercera componente.
        lista.append(vector(x, sin(x), 0))

        # Incrementamos x en dx
        x = x + dx

# Dibujamos una curva a través de los puntos en la lista color rojo
circle = curve(pos=lista, color=color.red)
```

La Figura 15 presenta la función seno planteada en el Ejemplo 1.3.

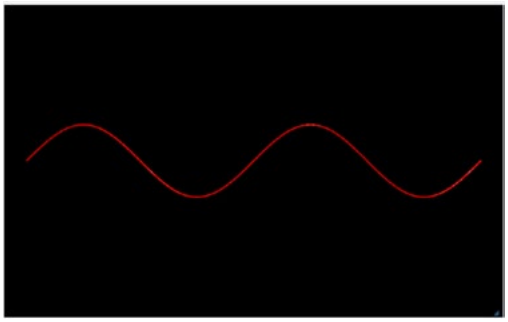


Figura 15. Dibujo de la función seno por medio de un objeto curve

Retener un número limitado de puntos

Al crear una curva o al agregar un punto a una curva, se puede especificar cuántos puntos se deben retener (Qiu, 2020). La siguiente declaración especifica que solo se deben conservar los 150 puntos más recientes a medida que agrega puntos a la curva:

c = curve(retain=150)

La siguiente fórmula especifica que, si ya hay 30 puntos en la curva, se debe eliminar el más antiguo y agregar el nuevo:

c.append(pos=vector(2,-1,0),retain=30)

Atributos de una curva

A continuación se muestra un resumen de los atributos de los puntos individuales o de la curva en su conjunto:

- *pos*: una posición de vector o lista de posiciones utilizadas para crear una curva. Una vez creada la curva, se deben utilizar métodos de curva como *c.point (N)* para leer los datos de posición.
- *radius*: radio de la sección transversal de este segmento de la curva; si se especifica para la curva como un todo, especifica el radio de cualquier punto para el que no se proporcionó un radio específico. El radio predeterminado es 0, lo que hace una curva delgada.
- *color*: color de un punto; si se especifica para la curva como un todo, especifica el color de los puntos para los que no se dio un color específico.
- *emissive*: si es *True*, las luces locales y distantes se ignoran y el brillo se rige por el color del objeto. El efecto es que el objeto parece estar brillando. El valor predeterminado para *emissive* es *False*.
- *visible*: si es *False*, el punto no se muestra; si es *False* para la curva en su conjunto, no se muestran puntos.
- *retain*: especifique cuántos puntos conservar.

Una curva también tiene los atributos *size, axis, up, shininess* y *emissive* (Scherer *et al.*, s.f.).

Mover, reorientar y cambiar el tamaño de una curva (uso de *origin*)
Las posiciones de los puntos son relativas; un punto cuya posición es el vector (2, 1, 0), normalmente se muestra en el vector de ubicación (2, 1, 0). Sin embargo, la posición de un punto es relativa al valor de origen propio de una curva, que por defecto es vector (0, 0, 0). Si cambia el valor de *origin* de la curva al vector (10, 6, 5), el punto se muestra en el vector de ubicación (12, 7, 5); es decir, el punto se muestra 2 a la derecha, 6 arriba y 0 delante del vector de posición de la curva (10, 6, 5). Otra forma de decir esto es que la ubicación de la pantalla es la suma vectorial: vector (10, 6, 5) + vector (2, 1, 0). Esto significa que se puede mover rápida y eficazmente toda la curva con solo cambiar el valor de origen de la curva. El valor *pos* de un punto individual no cambia; es solo que el punto se muestra en una posición desplazada. Como resultado, mover una curva completa es muy rápido.

Del mismo modo, los cambios de *size, axis, up, shininess* y *emissive* cambian de forma inmediata y rápida el tamaño de la curva, su orientación

en el espacio y la apariencia de su superficie. Cuando especifica la posición de un punto, es relativa a un origen en el vector (0,0,0) y con el vector de eje estándar (1,0,0). Cambiar el tamaño no cambia el radio; simplemente acerca o aleja los puntos (Scherer *et al.*, s.f.).

Rotar una curva

Al igual que con otros objetos, la forma de cambiar la orientación de toda la curva es cambiar el eje de la curva. También se puede rotar toda la curva alrededor de un eje específico. Si no se especifica un origen, la rotación se producirá alrededor del origen de la curva:

Métodos de curva

La modificación de los puntos existentes no se realiza directamente, sino mediante el uso de un poderoso conjunto de funciones, y este cambio hace posible mostrar curvas mucho más rápidamente (Scherer *et al.*, s.f.).

Lo siguiente es útil para modificar una curva llamada «*c*»:

- *c.npoints*: el número total de puntos actualmente en la curva.
- *c.append (…)*: agrega un punto o varios puntos al final.
- *c.unshift (…)*: pone un punto o muchos al inicio.
- *c.splice (…)*: inserta un punto o varios en cualquier lugar.
- *c.modify (n,v1)*: Modifica el punto *n = 0,1,2,3….* Por ejemplo, *c.modify (0,v1)* modifica el punto 1 por el punto indicado por *v1, modify (1,v1)*, modifica el punto 2 por el punto indicado por *v1* y así sucesivamente.
- *c.clear ()*: elimina todos los puntos.
- *data = c.pop (n)*: obtenga los datos para el punto *n* y elimínelos. Puede usar valores negativos, donde –1 es el último punto, –2 el próximo al último, etc.
- *data = c.shift ()*: obtiene los datos del primer punto y elimínalo.
- *data = c.point (n)*: obtiene los datos para el enésimo punto.
- *data = c.slice (start,end)*: obtiene los datos para una lista de puntos.

Ondas armónicas

La descripción de una onda unidimensional (que se propaga a en el eje x) con rapidez v es:

$$y(x,t) = f(x \mp vt)$$

Cuando el signo que precede la magnitud de la velocidad es positivo, la onda viaja hacia la izquierda; y cuando el signo es negativo, la onda viaja hacia la derecha.

Considere una onda senoidal o cosenoidal, conocidas como ondas armónicas simples u ondas armónicas, de la siguiente manera:

$$y(x,t) = Asen(\omega t \mp kx)$$

Donde k es una constante positiva llamada «número de propagación» o «número de onda angular» y cuya unidad es m^{-1}. La función seno varía entre +1 y –1 de manera que el máximo valor de $y(x,t)$ es A. Este valor máximo de la perturbación se conoce como la amplitud de la onda (Serway y Jewett, 2014). Puede demostrarse fácilmente si se hace:

$$\lambda = \frac{2\pi}{k}$$

En cualquier tiempo t, se verifica que:

$$y(x,t) = y(x \pm \lambda,t)$$

El período espacial λ se conoce como la «longitud de onda». Un cambio en x en la cantidad λ deja intacto a $y(x,t)$. De igual forma, si se hace:

$$T = \frac{2\pi}{\omega}$$

En cualquier posición x, se comprueba que:

$$y(x,t) = y(x,t \pm T)$$

El período temporal T se conoce simplemente como «periodo». Un cambio en t en la cantidad T deja intacto a $y(x,t)$. La constante ω se le conoce como frecuencia angular:

$$\omega = \frac{2\pi}{T} = 2\pi f$$

En donde f es la frecuencia (ciclos de la señal armónica por unidad de tiempo):

$$f = \frac{1}{T}$$

La velocidad de propagación de una onda se obtiene por medio de la expresión:

$$v = \lambda f = \frac{\omega}{k}$$

La Figura 16 representa la descripción de una onda armónica (Luda, s. f.).

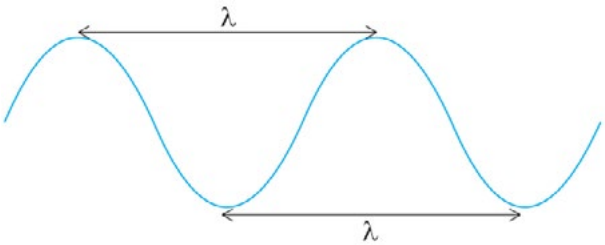


Figura 16. Representación gráfica de una onda armónica

El código en VPython para dibujar una onda armónica se presenta a continuación:

```
# Importamos las funciones necesarias de vpython y math
from vpython import *

# Configuración de la escena
scene.width      = 400
scene.height     = 200
scene.range      = 5
scene.center     = vector(5,0,0)
scene.title      = “Onda Armónica\n”
scene.autoscale  = False
scene.caption    = “Datos\n\n”

global A,f,longitudOnda,k,w

# Crear una curva para la onda
onda = curve(color = color.red)
```