

INSTITUTO TECNOLÓGICO DE NUEVO LAREDO

CON LA CIENCIA POR LA HUMANIDAD

INGENIERÍA EN SISTEMAS COMPUTACIONALES



Programación móvil 2

Portafolio de evidencias



DOCENTE

Ing. Humberto Peña Valle M.T.I.

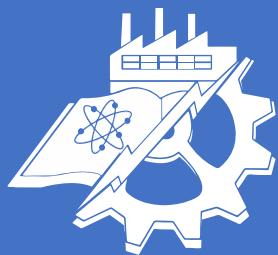


INTEGRANTES

Jessica Janet Grajeda Castellanos	17100229
Juan Felipe Garza Sánchez	17100218
Miguel Ángel Méndez Cruz	17100254

CONTENIDO

Portafolio de evidencias	1
Tema 1	3
Tema 2	105
Tema 3	282
Tema 4 y 5	314



INSTITUTO TECNOLÓGICO DE NUEVO LAREDO

CON LA CIENCIA POR LA HUMANIDAD

INGENIERÍA EN SISTEMAS COMPUTACIONALES



Programación móvil 2

Tema 1



DOCENTE

Ing. Humberto Peña Valle M.T.I.



INTEGRANTES

Jessica Janet Grajeda Castellanos 17100229

Juan Felipe Garza Sánchez 17100218

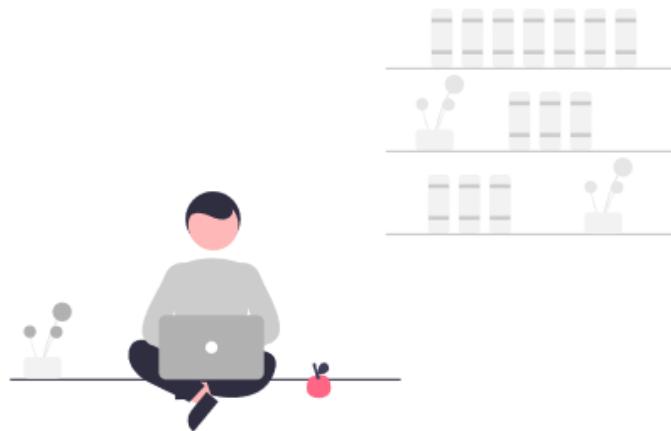
Miguel Ángel Méndez Cruz 17100254

ÍNDICE

Explicación general del contenido del tema	5
1.1 Investigacion	6
1.1.1 Macintosh Operating System	6
1.1.2 Xcode	33
1.1.3 Swift	49
1.1.4 Cocoa, Cocoa Touch y Foundation	72
1.2 Documentar los temas vistos en clase	78
1.2.1 Resumen del documental	79
1.2.2 Versiones de IOS	81
1.2.3 Guía de uso de una MAC	86
Conclusiones y/o recomendaciones	100
Bibliografía/Linkografía	101

EXPLICACIÓN GENERAL DEL CONTENIDO DEL TEMA

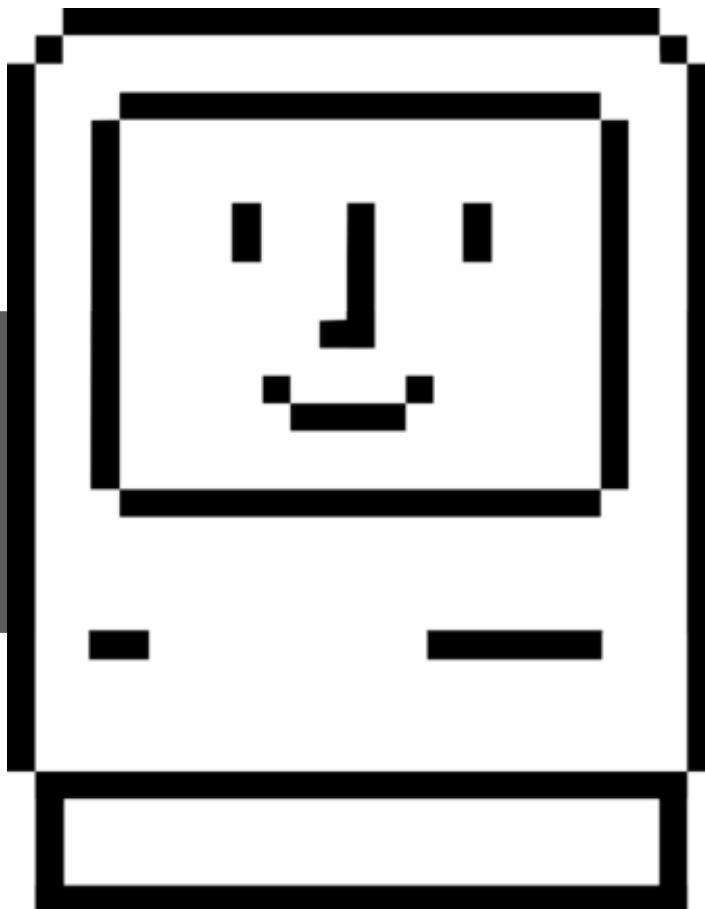
En este tema introductorio a la materia de Programación Móvil 2 veremos contenido como la historia de la compañía Apple, como fue que se dio su creación además de otros acontecimientos ocurridos a lo largo de su historia. También hablaremos acerca de los productos creados por esta compañía como las MacBook o iPhone, así como las versiones de sus sistemas operativos macOS y iOS. Además, se tocarán temas como el entorno de desarrollo integrado llamado Xcode, el cual sirve para programar software para los sistemas operativos de productos Apple. También veremos el lenguaje de programación Swift, los frameworks Cocoa, Cocoa Touch y Foundation. Por último, veremos una guía rápida para comenzar a utilizar un equipo Mac.



La investigación tiene como propósito una introducción a lo que el desarrollo de aplicaciones iOS respecta, pues desde un inicio abarcaremos el origen del mismo sistema operativo, historia de la compañía y entre otros acontecimientos importantes, también, abarcar el cómo es que se deben de manejar distintas herramientas y seguimiento de procedimientos que van desde la elaboración de la aplicación hasta la publicación de una aplicación en la App Store.

1.1 INVESTIGACIÓN

1.1.1 MACINTOSH OPERATING



CLASSIC MAC OS

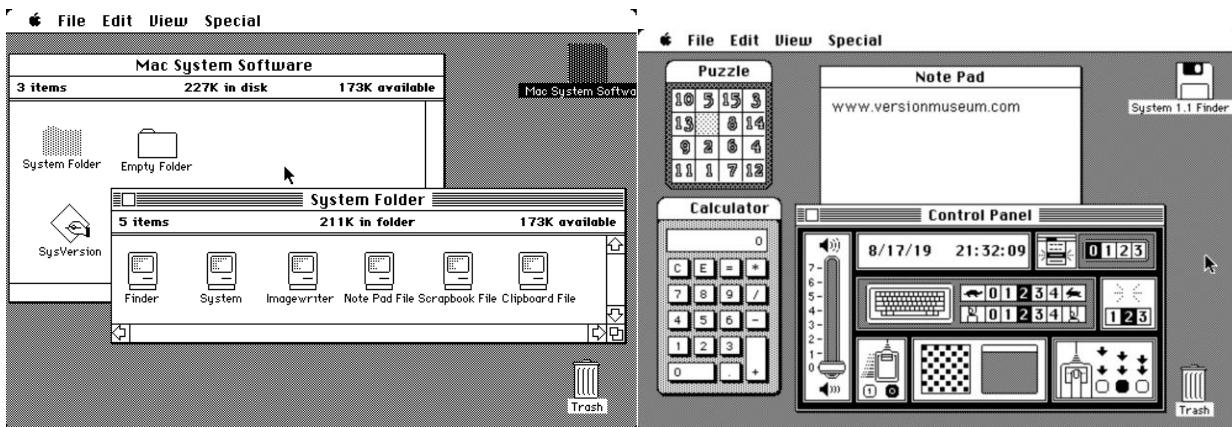
La versión clásica de Mac OS es en realidad una serie de versiones del sistema operativo Macintosh, desarrollado para la familia de computadoras Macintosh. Aunque el sistema operativo Macintosh no fue el primero en tener una interfaz gráfica (Este fue la Alto Executive desarrollada por Xerox PARC para la computadora Xerox Alto) sí fue la primera en ser exitosa. La razón fue por su costo, el cual era bastante menor a comparación de sus competidores, por ejemplo, la Xerox Alto costaba en su momento \$32,000 USD, la Xerox Star costó \$16,000 USD mientras que la Apple Lisa tenía un costo de \$10,000 USD. Por esto se volvió muy popular entre los entusiastas de la tecnología, además de que tenía una GUI mucho más atractiva que la de sus competidores.

El nombre Macintosh (Luego recortado a solo “Mac”) fue derivado del nombre McIntosh, este nombre era una referencia al nombre de la manzana favorita del creador del proyecto, Jef Raskin. Raskin nombró al proyecto como una fruta porque pensó que dar nombres femeninos a las computadoras era sexista. Sin embargo, Apple no tenía los derechos del nombre, este lo tenía una empresa fabricante de equipos de audio llamada McIntosh Laboratory. Esto provocó que Jef Raskin cambiara el nombre ligeramente a Macintosh.



VERSIONES

1984 — System 1



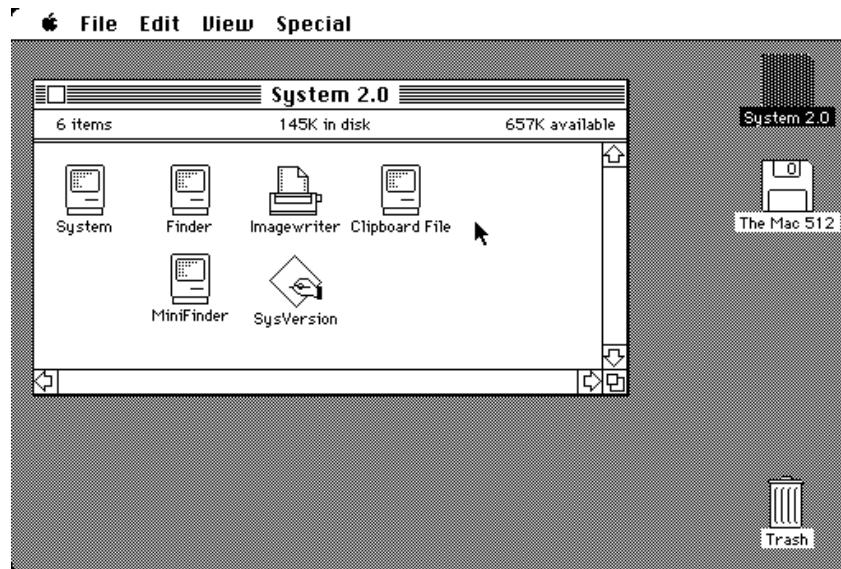
Esta primera versión del sistema operativo fue lanzada el 24 de enero de 1984 en conjunto de la computadora Macintosh 128K, este no tenía un nombre oficial, esto porque fue basado parcialmente en el sistema operativo llamado Lisa OS también desarrollado

por Apple para la computadora Lisa en 1983. Gracias a un acuerdo que llegaron Apple y Xerox, el cual consistía en que Apple dejaría que Xerox comprase acciones a un precio accesible, esta versión utilizaba conceptos de la computadora Xerox Alto.

System 1 tenía características como un escritorio, ventanas, iconos, un mouse, menús, scrollbars, carpetas, documentos, aplicaciones, la basura y software del sistema, además de otras características como:

- MFS (Macintosh File System) fue el único formato de sistema de archivos admitido. No admitía una estructura de directorios anidada. En cambio, las carpetas se almacenaron en el archivo de escritorio del Finder y no estaban disponibles fuera del Finder
- No había ningún comando para crear una carpeta nueva. Había una carpeta proporcionada por defecto que se podía duplicar si fuera necesario.
- Los cuadros de diálogo Abrir y Guardar no dieron ningún reconocimiento a las carpetas
- Solo podía ejecutar una aplicación a la vez
- No tenía soporte para colores
- Tuvo una actualización más, System 1.1 que incluía una optimización a su código, haciendo que los tiempos de arranque fuesen más rápidos

1985 — System 2



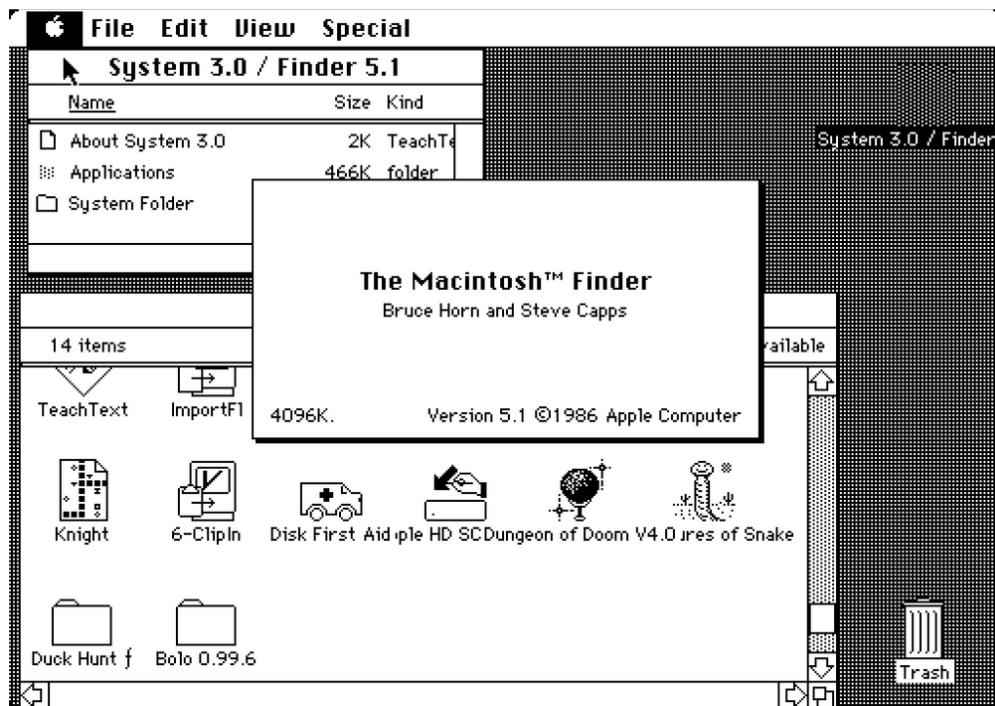
Esta versión fue lanzada en abril de 1985, casi un año después de que Apple lanzara su anterior actualización de software para Macintosh. La mejora más significativa que proporcionó esta versión fue el rendimiento. El tiempo de arranque se redujo en un 20%.

Otras características de esta versión son las siguientes:

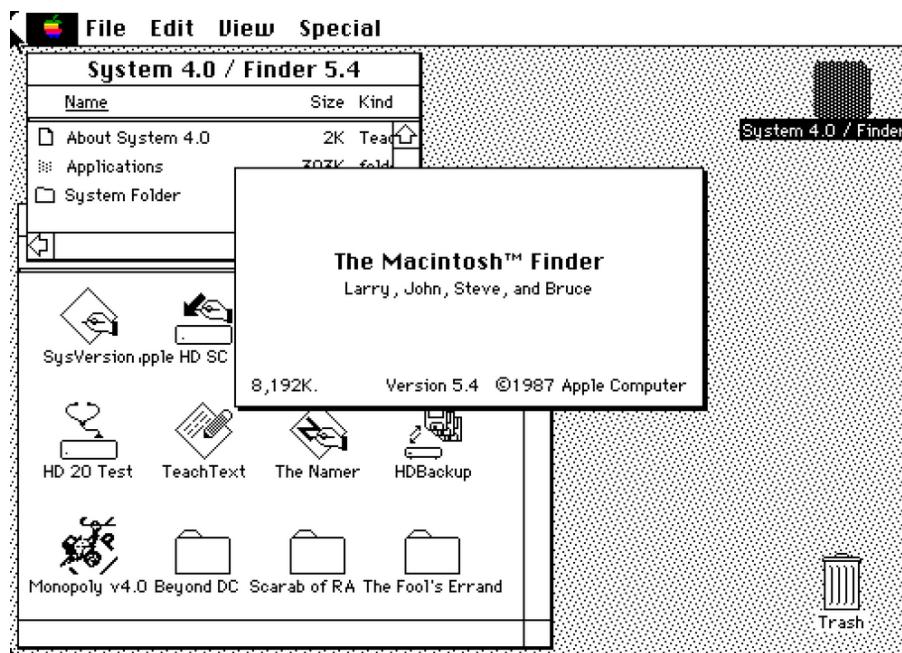
- Los discos ahora se pueden expulsar arrastrando sus iconos a la Papelera, en lugar de seleccionar el comando Expulsar disco y luego arrastrar el ícono a la Papelera.
- Se agregó una vista de lista no jerárquica, donde los elementos dentro de una carpeta se enumeraban en una lista vertical con pequeños íconos.

1986 — System 3

Lanzado en enero de 1986, su cambio más notable fue el cambio de MFS a HFS para el sistema de archivos, ahora las carpetas eran reales y las carpetas se podían anidar dentro de las carpetas. Se agregaron cuadros de zoom en el extremo derecho de las barras de título de la ventana; al hacer clic, la ventana cambiaría de tamaño para ajustarse al contenido de esa carpeta, si es posible; al hacer clic nuevamente, la ventana volvería a su tamaño anterior. El ícono de la papelera se abultaba cuando se colocaba algo en él y las líneas apuntaban en la dirección opuesta. (Como ")" en lugar de "(.".). También se mejoró la velocidad del Finder mucho más.



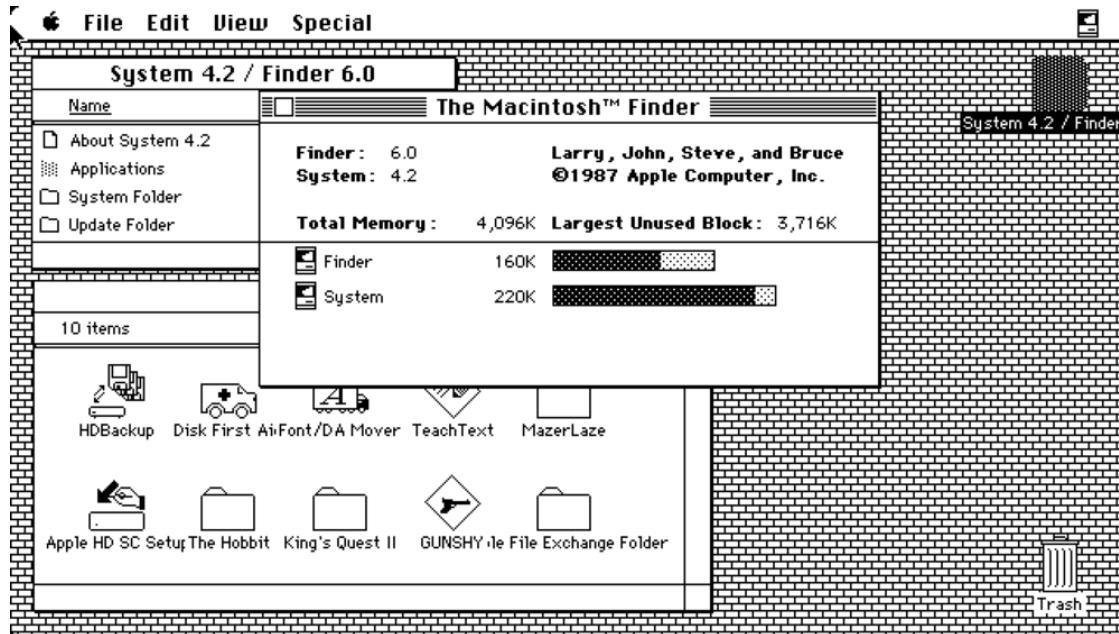
1987 — System 4



Esta versión fue lanzada en enero de 1987, no tuvo tantos cambios a comparación de System 3 pero si se solucionaron muchos bugs de la versión anterior, además de que hubo actualización de código para que este fuese más eficiente

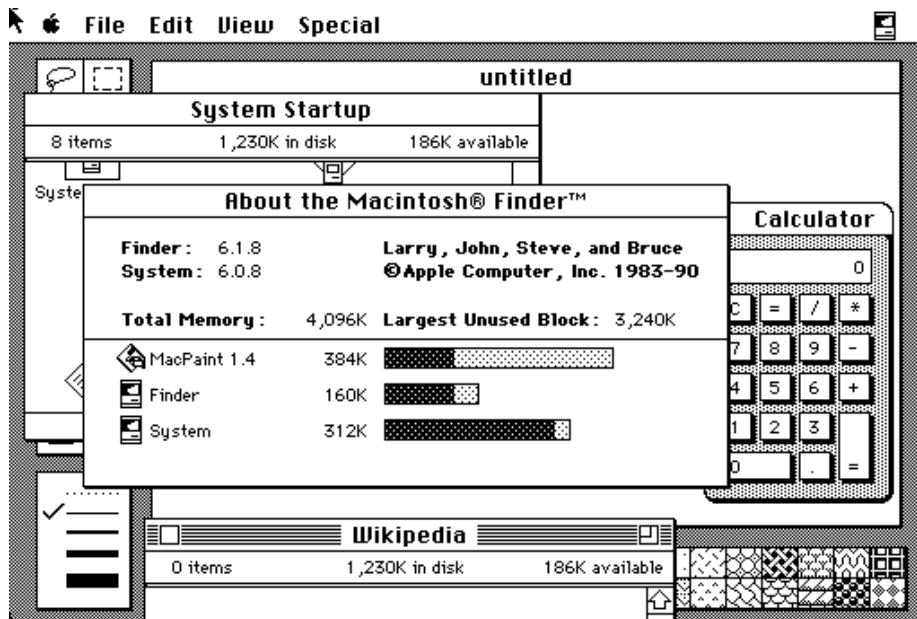
1987 — System 5

System 5 era un nombre comercial para un paquete de software que fue lanzado en octubre del 1987, este contenía System 4.2, Finder versión 6.0, MultiFinder 1.0 y LaserWriter versión 5.0.



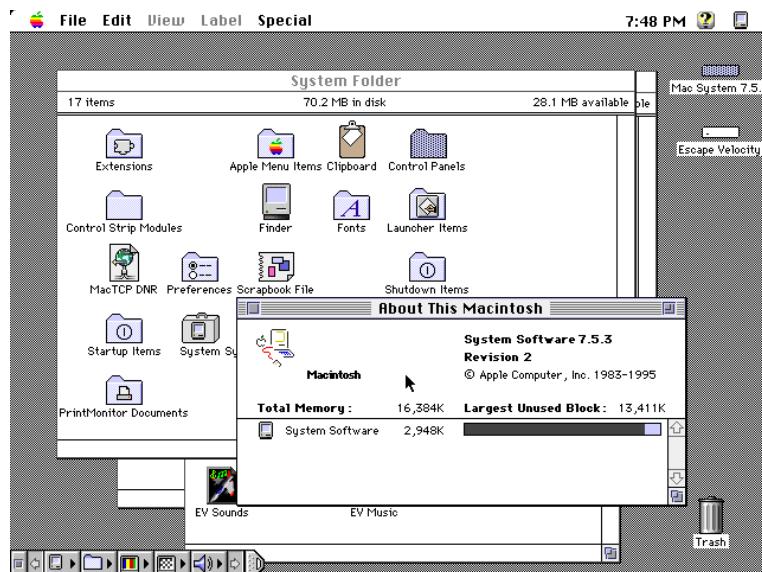
1988 — System 6

Lanzado en abril de 1988, System 6 agregó soporte de color, aunque el Finder en sí todavía no usaba color, incluso en máquinas con capacidad de color. System 6 fue el primero que incluyó la multitarea cooperativa de forma predeterminada. Es ampliamente considerado como el mejor sistema Mac para máquinas más antiguas (que no son PowerPC), y hay grupos de aficionados que navegan por la web utilizando System 6 y Mac antiguas para ejecutarlo.



1990 — System 7

System 7 fue lanzado el 13 de mayo de 1991. Ofrecía una serie de mejoras del sistema que antes no existían o que solo estaban disponibles como extensiones opcionales del sistema operativo.



Esta versión obtuvo las siguientes características:

- Se implementó la multitarea cooperativa incorporada. Anteriormente, esta función estaba disponible a través de MultiFinder en System 6, o no estaba disponible en absoluto. Debido a que se podía ejecutar más de una aplicación a la vez, los Desk Accessories quedaron obsoletos y System 7 no los trata de manera diferente a otras aplicaciones.
- La papelera de reciclaje ahora era una carpeta real en lugar del estado especial que tenía anteriormente. Esto permitió que los elementos se tiraran a la papelera en diferentes volúmenes, cada uno con su propia papelera.

También hubo una gran cantidad de cambios arquitectónicos para hacer que el sistema operativo sea más coherente y estable. Apple se jactó en su lanzamiento de que System 7 era "sólido como una roca", y aunque fue una gran mejora con respecto a los sistemas anteriores, la afirmación fue bastante hiperbólica. Esta afirmación se volvió algo vacía durante los años siguientes, ya que la estabilidad del sistema se degradó terriblemente a medida que crecía la complejidad. Las versiones posteriores de System 7 eran notoriamente poco confiables, a menudo congelaban toda la máquina y corrompián los sistemas de archivos después de errores inofensivos.

A partir de la actualización 7.6 se decidió cambiar el nombre, dejando atrás el "System" para utilizar el nombre "Mac OS" y así tener un nombre más comercial para poder vender licencias del sistema operativo a terceros, proyecto que después fue abandonado por Apple.

1997 — Mac OS 8

lanzado el 26 de julio de 1997, fue la primera versión que incluyó un video introductorio.



Mac OS 8 renovó el Finder. El Finder finalmente fue multiproceso, lo que significa que podía hacer más de una cosa al mismo tiempo. El subproceso múltiple también significó

que las computadoras con más de un procesador experimentarían un mejor rendimiento del Finder. La apariencia general del Finder se renovó para que pareciera más tridimensional. La apariencia del Finder también se hizo mucho más personalizable. El uso compartido de Web personal permitió a los usuarios alojar páginas web en sus computadoras.

En cuanto a cambios estéticos, a las carpetas se les dio un aspecto tridimensional en azul, las ventanas ahora estaban teñidas de platino y la barra de menú ahora era plateada, en lugar del blanco puro original. Los botones aparecieron en relieve (en lugar de un simple switch en blanco y negro), y las barras de desplazamiento también aparecieron en un nuevo aspecto 3D más enriquecido.

1999 — Mac OS 9

Mac OS 9, la última versión principal del Mac OS clásico, se lanzó el 23 de octubre de 1999.



Se agregaron múltiples usuarios en Mac OS 9, lo que permitió a los usuarios iniciar sesión y tener su propia configuración única. También se implementó AppleTalk sobre TCP / IP. La Actualización de software permitía a los usuarios obtener actualizaciones de software fuera de Internet e informaba a los usuarios de las nuevas actualizaciones a medida que aparecían.

En mayo de 2002, el CEO de Apple, Steve Jobs, pronunció un "funeral" simulado para Mac OS 9 durante su discurso de apertura, vestido de negro y dirigiéndose a un ataúd que contenía una caja minorista de gran tamaño. El propósito era anunciar que Apple había detenido todo el desarrollo de OS 9. Mac OS 9.2.2 era la versión final de Mac OS 9 y del clásico Mac OS.

OS X

Lo que se convertiría en macOS se originó en NeXT, una empresa fundada por Steve Jobs tras su salida de Apple en 1985. Ahí, se desarrolló el sistema operativo NeXTSTEP derivado de Unix y luego se lanzó en 1989.

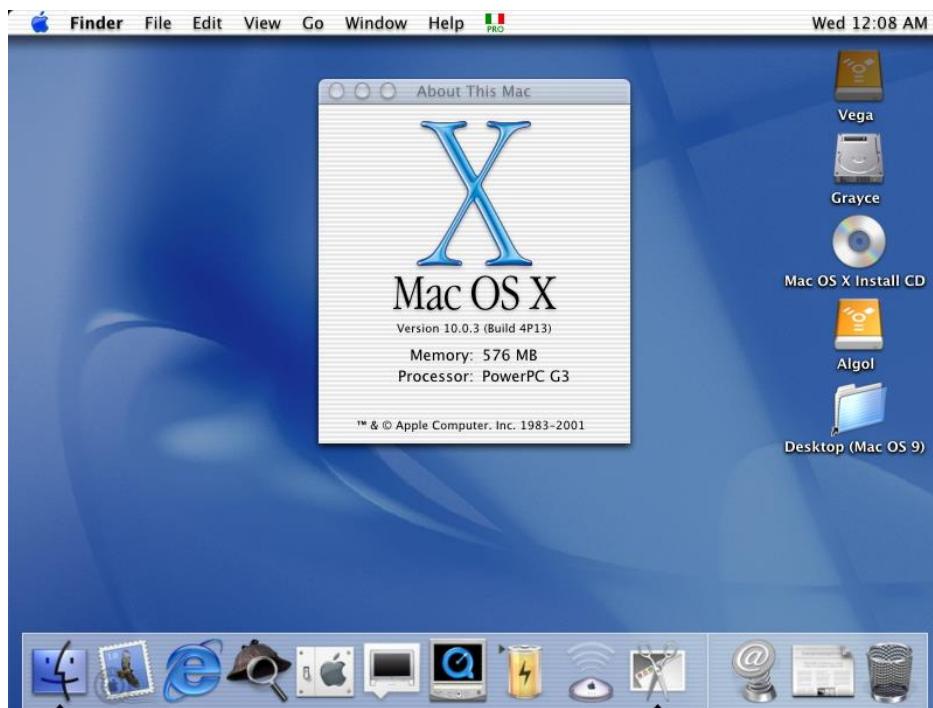


A principios de la década de 1990, Apple había intentado crear un sistema operativo de "próxima generación" para superar a su sistema operativo, Mac OS, a través de los proyectos Taligent, Copland y Gershwin, pero todos fueron abandonados. Esto llevó a Apple a comprar NeXT en 1996, lo que permitió que NeXTSTEP sirviera como base para el sistema operativo de próxima generación de Apple.

VERSIONES

Mac OS X 10.0 – Cheetah

Esta versión tuvo un cambio radical a comparación de Mac OS 9, fue lanzado el 24 de marzo del 2001.



Este introdujo una nueva interfaz del usuario llamado Aqua, que mostraba barras de título con un aspecto de metal pulido, botones de colores brillantes en las ventanas e iconos fotorrealistas. Esta versión presentó la base que aún le da a las Mac modernas su apariencia icónica. También se incluyeron nuevas aplicaciones, como Mail, Address Book yTextEdit.

Mac OS X 10.1 – Puma

OS X Puma fue lanzado el 25 de septiembre del 2001. Brindaba una navegación e inicio de sesión más rápidos, así como opciones configurables, como un Dock móvil y nuevos menús del sistema con controles de volumen, duración de la batería y acceso AirPort en la barra de menú. iTunes se incluyó con OS X 10.1 con capacidades de grabación de CD, reproducción de DVD y soporte para cámaras digitales y reproductores MP3 de terceros.



Mac OS X 10.2 – Jaguar

Fue lanzado el 24 de agosto del 2002. Si bien las versiones anteriores de OS X tenían nombres de gatos de la jungla, no se usaron con fines de marketing hasta OS X Jaguar. Anunciado en la Macworld Expo 2002, presentaba una aplicación de correo actualizada que colocaba spam en una carpeta de correo no deseado. iChat se convirtió en la aplicación de mensajería instantánea predeterminada de Apple, con compatibilidad integrada con AOL Instant Messenger. Junto con un nuevo Finder que ofrece una búsqueda refinada, OS X 10.2 incluyó acceso universal por primera vez y permitió a los usuarios ampliar el contenido de la pantalla o iniciar aplicaciones con dictado de voz.



Mac OS X 10.3 – Panther

Apple lanzó esta versión el 24 de octubre del 2003. Incluyó una función útil que permite a los usuarios ver instantáneamente todas las ventanas abiertas a la vez, ver las ventanas de un programa actual o simplemente ver archivos en el escritorio. iChat AV permite a los usuarios comunicarse con audio y video, así como con texto, y Safari se convirtió en el navegador web predeterminado de Mac después de que dejó de ser compatible con Internet Explorer para Mac.



Mac OS X 10.4 – Tiger

Tiger fue lanzado el 29 de abril del 2005. Incluía un cliente de búsqueda universal, Spotlight, que permitía a los usuarios buscar archivos, correos electrónicos, contactos, imágenes, calendarios y aplicaciones en todo el sistema desde la barra de menús. El tablero presentaba widgets para el clima, información de vuelos, tickers de acciones y más. Safari incluía funciones RSS e iChat admitía hasta cuatro participantes en una videoconferencia o 10 participantes en una audioconferencia.



Tiger se convirtió en el primer sistema operativo en admitir la arquitectura Apple-Intel después de la transición de Apple a los procesadores Intel x86, y se convirtió en la versión de más larga ejecución de Mac OS X antes del lanzamiento de Leopard 30 meses después.

Mac OS X 10.5 – Leopard

Apple lanzó Leopard el 26 de octubre del 2007 y se convirtió en un momento de redefinición para su software, ya que se señaló que era la "actualización más grande de OS X". Presentaba una apariencia modernizada con un Dock reflectante tridimensional, una barra de menú semitransparente, sombras más grandes para ventanas activas y nuevos íconos de alta resolución.



Mac OS X 10.6 – Snow Leopard

Snow Leopard se lanzó el 28 de agosto del 2009. Esta actualización incluyó todas las aplicaciones de Apple reescritas en código de 64 bits; Grand Central Dispatch, una nueva forma para que los desarrolladores de software escriban aplicaciones que aprovechan los procesadores multinúcleo; y OpenCL, un estándar abierto basado en C que permite

a los desarrolladores aprovechar el poder de la GPU para tareas que van más allá de los gráficos.



Mac OS X 10.7 – Lion

Apple lanzó OS X Lion el 20 de julio del 2010. Admite oficialmente gestos multitáctiles con gestos y respuestas adicionales, como desplazamiento con banda elástica, zoom de página e imagen y deslizamiento de pantalla completa. Las aplicaciones podrían ser de pantalla completa por primera vez, y la introducción de Mission Control combinó Exposé, Dashboard, Spaces y aplicaciones de pantalla completa para brindar a los usuarios un lugar para ver y navegar todo lo que se ejecuta en su Mac.



OS X 10.8 – Mountain Lion

Fue lanzado el 25 de julio del 2012, y desde entonces eliminó el prefijo "Mac" de todas las referencias de OS X en su sitio web. Correo sincronizado con iCloud, calendarios, contactos, recordatorios, documentos, notas y más entre Mac, iPads, iPhones y iPod touch. También se trajeron aplicaciones de dispositivos iOS: Recordatorios, Notas y Mensajes, que llevaron iMessage a la Mac por primera vez; El Centro de notificaciones alertó a los usuarios sobre nuevos correos electrónicos, mensajes, actualizaciones de software o alertas de calendario; El dictado convertía palabras en texto; y un nuevo botón Compartir permitió compartir fotos, videos, archivos y enlaces con Mail, Messages y AirDrop.



OS X 10.9 – Mavericks

Fue lanzado el 22 de octubre del 2013. Reemplazó su convención de nombres de grandes felinos con lugares en el estado natal de Apple, California; por lo tanto, Mavericks recibió su nombre de un lugar popular para practicar surf.



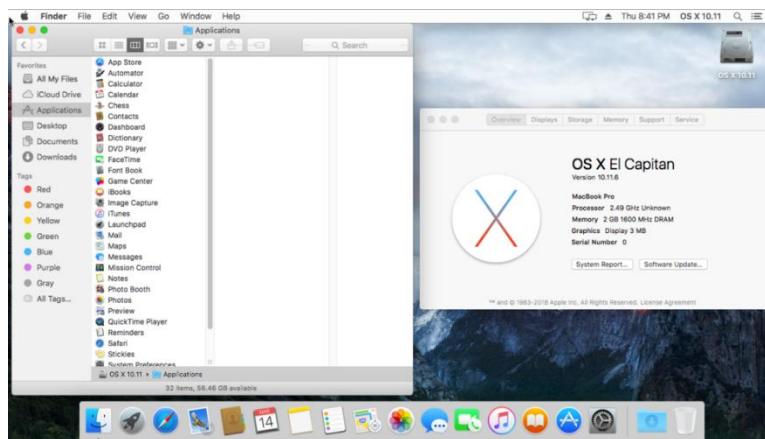
OS X 10.10 – Yosemite

Lanzado el 16 de octubre del 2014. Para igualar la revisión del diseño de iOS 7, OS X Yosemite descartó la interfaz esquemática que había utilizado durante más de 10 años. La actualización incluyó un diseño gráfico plano, efectos de translucidez borrosos, un Dock bidimensional que repetía el utilizado en Tiger, íconos actualizados, esquemas de colores claros y oscuros, y el primer reemplazo del tipo de letra del sistema predeterminado de Lucida Grande a Helvetica Neue. Con Continuity, las Mac ahora pueden recibir y realizar llamadas desde un iPhone en la misma red WiFi. Las Mac también podían enviar y recibir mensajes de texto SMS, y iCloud sincronizaba todos los iMessages y mensajes de texto entre una Mac y un dispositivo iOS.



OS X 10.11 – El Capitan

Lanzada el 30 de septiembre del 2015, esta versión se enfoco en estabilidad, performance y seguridad



macOS 10.12 – Sierra

MacOS Sierra fue lanzado el 20 de septiembre del 2016. Sierra trajo una variedad de cambios, más notablemente la llegada de Siri al escritorio Mac. Los usuarios pueden acceder a Siri en la esquina superior derecha de la barra de menú entre los íconos del menú Centro de notificaciones y Spotlight, un ícono de Dock, el ícono de la barra de menú o una tecla de acceso rápido. El asistente activado por voz podría ayudar a los usuarios a encontrar archivos, obtener acceso rápido a información como el clima o realizar búsquedas, con resultados de búsqueda que se pueden guardar y anclar en el área de notificaciones. Otros cambios en Sierra incluyeron un Portapapeles universal, que permitía copiar texto de un dispositivo Apple a otro; iCloud Desktop and Documents, para sincronizar datos entre Mac utilizando la nube; Desbloqueo automático, que permitió al Apple Watch desbloquear una cuenta macOS; y Apple Pay en la web. Las fotos recibieron una nueva función de Recuerdos que creaba automáticamente colecciones

seleccionadas de las fotos y videos favoritos de los usuarios. Sierra también entregó más de 60 correcciones de seguridad.



macOS 10.13 – High Sierra

Lanzado el 25 de septiembre del 2017, macOS High Sierra es una actualización iterativa de macOS Sierra, y su nombre es una oda a las montañas de Sierra Nevada de California. Esta versión presenta Apple File System para administrar de manera eficiente los datos para un mejor rendimiento, así como un nuevo estándar de la industria para video 4K: codificación de video de alta eficiencia, también conocida como HEVC o H.265. Una aplicación de Fotos actualizada facilita la búsqueda y organización de fotos y contiene herramientas de edición avanzadas. Safari se vuelve más rápido y contiene nuevas funciones como la Prevención de seguimiento inteligente para eliminar los datos de seguimiento entre sitios de los anunciantes. Safari también desactiva automáticamente esos molestos videos de reproducción automática con audio para una experiencia de navegación más silenciosa.



macOS 10.14 – Mojave

Apple lanzó esta versión el 24 de septiembre del 2018. Su característica principal es un nuevo modo oscuro que convierte la interfaz de usuario general en gris oscuro. (Un nuevo fondo de pantalla interactivo también puede adaptarse a la hora del día en función de la ubicación del usuario). La nueva función Stacks organiza escritorios desordenados apilando automáticamente archivos similares en grupos, y la vista de galería, que recuerda a Cover Flow, permite a los usuarios hojear visualmente los archivos. Los paneles de vista previa muestran metadatos de archivos, y Vista rápida permite a los usuarios rotar, recortar y marcar archivos PDF y recortar clips de vídeo sin abrir aplicaciones nativas. Las aplicaciones News, Stocks, Voice Memos y Home de iOS llegan al escritorio; Group FaceTime permite hasta 32 usuarios en un chat de video simultáneo; y la Mac App Store tiene un diseño renovado. Las mejoras de seguridad a través de la Prevención de seguimiento inteligente de Safari bloquean el seguimiento del usuario sin permiso. La herramienta de captura de pantalla contiene nuevos controles en pantalla para marcar imágenes y permite a los usuarios grabar videos. Continuity Camera permite a los usuarios tomar una foto o escanear un documento con un iPhone o iPad y que aparezca instantáneamente en su Mac. Las herramientas de marcado integradas para Finder, Quick Look y Screenshots también hacen que agregar comentarios sea rápido y fácil.



macOS 10.15 – Catalina

Lanzado el 7 de octubre de 2019

En macOS Catalina, Apple ha eliminado la aplicación iTunes que ha sido un elemento básico del sistema operativo Mac desde 2001. iTunes se dividió en tres aplicaciones: Música, Podcasts y TV.

Las nuevas aplicaciones son similares en función a iTunes ahora, pero están divididas por función. Todavía puede administrar sus dispositivos en Catalina, pero ahora se hace a través del Finder en lugar de a través de una aplicación. La sincronización de medios se puede realizar mediante las aplicaciones Apple TV, Podcasts o Música.



macOS 11 – Big Sur

Lanzado el 12 de noviembre del 2020 actualiza el diseño de la interfaz de usuario, descrito por Apple como el mayor cambio desde la introducción de Mac OS X. Sus cambios incluyen translucidez en varios lugares y una nueva paleta de colores. Todas las aplicaciones estándar, así como el Dock y la barra de menú, se rediseñaron y simplificaron, y sus íconos ahora tienen formas cuadradas redondeadas como las aplicaciones de iOS y iPadOS.

Soporte para los procesadores de Apple: Apple tiene nuevos procesadores propios para Mac, los M1, y macOS 11 Big Sur es la primera versión del sistema operativo diseñada para sacarles el máximo provecho. Eso sí, macOS todavía soportará a los Mac con chips de Intel "durante los próximos años".

Soporte para aplicaciones de iOS e iPadOS: MacOS Big Sur podrá ejecutar de forma nativa aplicaciones de iOS e iPadOS, aunque únicamente en los Mac con los nuevos procesadores M1. Los desarrolladores no tendrán que hacer modificaciones, simplemente Mac podrá instalarlas y usarlas como si fuera un iPhone o un iPad.



OS X SERVER

Esta versión del sistema operativo añade funcionalidad de servidor y herramientas de administración del sistema a macOS y proporciona herramientas para administrar tanto computadoras basadas en macOS como dispositivos basados en iOS. Este proporciona servicios de red como un agente de transferencia de correo, servidores AFP y SMB, un servidor LDAP y un servidor de nombres de dominio, así como aplicaciones de servidor que incluyen un servidor web, una base de datos y un servidor de calendario.



Las versiones de Mac OS X Server anteriores a la versión 10.7 "Lion" se llegaron a vender como sistemas operativos de servidor completos e independientes. Pero a partir de Mac OS X 10.7 "Lion", Mac OS X Server (y sus sucesores OS X Server y macOS Server) se han ofrecido como paquetes de software complementarios, vendidos a través de la App Store, que se instalan sobre la correspondiente Instalación del sistema operativo.

VERSIONES

Mac OS X Server 1.0 (Rhapsody)

Lanzado el 16 de marzo del 1999, Mac OS X Server 1.0 - 1.2v3 se basó en Rhapsody, un híbrido de OPENSTEP de NeXT Computer y Mac OS 8.5.1. La GUI parecía una mezcla de la apariencia platinada de Mac OS 8 con la interfaz basada en OPENSTEP. Incluía una capa de tiempo de ejecución llamada Blue Box para ejecutar aplicaciones heredadas basadas en Mac OS dentro de una ventana separada.

Mac OS X Server 10.0 (Cheetah)

Lanzamiento: 21 de mayo de 2001

Mac OS X Server 10.0 incluía la nueva interfaz de usuario Aqua, Apache, PHP, MySQL, Tomcat, compatibilidad con WebDAV, Macintosh Manager y NetBoot.

Mac OS X Server 10.1 (Puma)

Lanzamiento: 25 de septiembre de 2001

Mac OS X Server 10.1 presentó un rendimiento mejorado, mayor estabilidad del sistema y menores tiempos de transferencia de archivos en comparación con Mac OS X Server 10.0. Se agregó soporte para configuraciones de almacenamiento RAID 0 y RAID 1, y Mac OS 9.2.1 en NetBoot. [6]

Mac OS X Server 10.2 (Jaguar)

Publicado: 23 de agosto de 2002

La versión 10.2 de Mac OS X Server incluye la administración de archivos y usuarios de Open Directory actualizada, que con esta versión se basa en LDAP, comenzando la desaprobación de la arquitectura NetInfo originada en NeXT. La nueva interfaz de Workgroup Manager mejoró significativamente la configuración. El lanzamiento también vio importantes actualizaciones para NetBoot y NetInstall. Se proporcionan muchos servicios de red comunes, como NTP, SNMP, servidor web (Apache), servidor de correo (Postfix y Cyrus), LDAP (OpenLDAP), AFP y servidor de impresión. La inclusión de Samba versión 3 permite una estrecha integración con clientes y servidores de Windows. También se incluyen MySQL v4.0.16 y PHP v4.3.7.

Mac OS X Server 10.3 (Panther)

Lanzamiento: 24 de octubre de 2003

La versión 10.3 de Mac OS X Server incluye la administración de archivos y usuarios de Open Directory actualizada, que con esta versión se basa en LDAP, comenzando la desaprobación de la arquitectura NetInfo originada en NeXT. La nueva interfaz de Workgroup Manager mejoró significativamente la configuración. Se proporcionan muchos servicios de red comunes, como NTP, SNMP, servidor web (Apache), servidor de correo (Postfix y Cyrus), LDAP (OpenLDAP), AFP y servidor de impresión. La inclusión de Samba versión 3 permite una estrecha integración con clientes y servidores de Windows. También se incluyen MySQL v4.0.16 y PHP v4.3.7. [Cita requerida]

Mac OS X Server 10.4 (Tiger)

Lanzamiento: 29 de abril de 2005

La versión 10.4 agrega compatibilidad con aplicaciones de 64 bits, listas de control de acceso, Xgrid, agregación de enlaces, filtrado de correo no deseado, detección de virus (ClamAV), asistente de configuración de puerta de enlace y servidores

Mac OS X Server 10.5 (servidor Leopard)

El servidor Mac OS X Leopard que ejecuta Server Admin en el escritorio

Lanzamiento: 26 de octubre de 2007. Leopard Server se vendió una licencia de clientes ilimitados. Mac OS X Server versión 10.5.x "Leopard" fue la última versión principal de Mac OS X Server que admitió servidores y estaciones de trabajo basados en PowerPC, como Apple Xserve G5 y Power Mac G5.

Mac OS X Server 10.6 (servidor Snow Leopard)

Publicado: 28 de agosto de 2009

Snow Leopard Server se vendió por 499 dólares e incluía licencias de cliente ilimitadas

Mac OS X 10.7 (Lion Server)

Lanzamiento: 20 de julio de 2011

Al publicar la versión preliminar para desarrolladores de Mac OS X Lion en febrero de 2011, Apple indicó que, a partir de Lion, Mac OS X Server se incluiría con el sistema operativo y no se comercializaría como un producto separado.

OS X 10.8 (servidor de Mountain Lion)

Lanzamiento: 25 de julio de 2012.

Al igual que Lion, Mountain Lion no tenía una edición de servidor separada. Un paquete OS X Server estaba disponible para Mountain Lion en Mac App Store

OS X 10.9 (servidor Mavericks)

Lanzamiento: 22 de octubre de 2013.

No hay una edición de servidor separada de Mavericks, al igual que no había una edición de servidor separada de Mountain Lion. Hay un paquete, disponible en la Mac App Store que incluye una aplicación de administración de servidor llamada Server, así como otras herramientas administrativas adicionales para administrar perfiles de cliente y Xsan

OS X 10.10 (Yosemite Server 4.0)

Lanzamiento: 16 de octubre de 2014.

Aquellos inscritos en los programas para desarrolladores de Mac o iOS reciben un código para descargar OS X Server de forma gratuita.

OS X 10.11 (servidor 5.0)

Lanzamiento: 16 de septiembre de 2015.

La versión 5.0.3 de OS X Server funciona con OS X Yosemite 10.10.5 y OS X El Capitan 10.11.

OS X 10.11 (servidor 5.1)

Lanzamiento: 21 de marzo de 2016.

OS X Server 5.1 requiere 10.11.4 El Capitan, ya que las versiones anteriores de OS X Server no funcionarán en 10.11.4

macOS 10.12 (servidor 5.2)

Lanzamiento: 20 de septiembre de 2016.

La versión 5.2 de macOS Server funciona con OS X El Capitan 10.11 o macOS Sierra 10.12.

macOS 10.12 (servidor 5.3)

Lanzamiento: 17 de marzo de 2017.

La versión 5.3 de macOS Server solo funciona en macOS Sierra (10.12.4) y posteriores.

macOS 10.13 (servidor 5.4)

Lanzamiento: 25 de septiembre de 2017.

La versión 5.4 de macOS Server solo funciona en macOS High Sierra (10.13) y posteriores.

macOS 10.13.3 (servidor 5.5)

Lanzamiento: 23 de enero de 2018.

La versión 5.5 de macOS Server solo funciona en macOS High Sierra (10.13.3) y posteriores.

macOS 10.13.5 (servidor 5.6)

Lanzamiento: 24 de abril de 2018.

La versión 5.6 de macOS Server solo funciona en macOS High Sierra (10.13.5) y posteriores.

macOS 10.14 (servidor 5.7)

Lanzamiento: 28 de septiembre de 2018.

La versión 5.7 de macOS Server solo funciona en macOS Mojave (10.14) y posteriores.

Con esta versión, Apple dejó de incluir servicios de código abierto como Calendar Server, Contacts Server, Mail Server, DNS, DHCP, VPN Server y sitios web con macOS Server. Los servicios incluidos ahora están limitados a Profile Manager, Open Directory y Xsan.

macOS 10.14 (servidor 5.8)

Lanzamiento: 25 de marzo de 2019.

La versión 5.8 de macOS Server solo funciona en macOS Mojave (10.14.4) y posteriores. Profile Manager admite nuevas restricciones, cargas útiles y comandos.

macOS 10.15 (servidor 5.9)

Lanzamiento: 8 de octubre de 2019.

La versión 5.9 de macOS Server solo funciona en macOS Catalina (10.15) y posteriores.

macOS 10.15 (servidor 5.10)

Lanzamiento: 1 de abril de 2020.

La versión 5.10 de macOS Server solo funciona en macOS Catalina (10.15) y posteriores.

macOS 11 (servidor 5.11)

Lanzamientos: 15 de diciembre de 2020.

La versión 5.11 de macOS Server solo funciona en macOS Big Sur (11) y posteriores.

TABLA COMPARATIVA

A continuación, se presenta una tabla que compara los siguientes sistemas operativos:

- macOS Big Sur
- Microsoft Windows 10
- Linux Ubuntu 20

	macOS Big Sur	Microsoft Windows 10	Linux Ubuntu 20
Creador o compañía	Apple	Microsoft	Canonical Ltd.
Fecha de lanzamiento	12 de noviembre del 2020	29 de julio del 2015	23 de abril del 2020
Costo por licencia	Gratis al comprar una computadora Apple compatible	\$139 - \$199 USD	Gratis
Tipo de núcleo (CPU)	Intel x86-64, ARM64	Intel, Amd, Qualcomm Snapdragon	Amd64 o Intel 64
Tipo de kernel	XNU	Windows NT kernel	Linux Kernel
Arquitectura soportada	<ul style="list-style-type: none"> • X86-64 • ARM64 	<ul style="list-style-type: none"> • IA-32 • x86-64 • ARMv7 • ARM64 	<ul style="list-style-type: none"> • Intel x86 • AMD64 • 64bit ARM • IBM Power Systems • IBM Z/Architecture
Procesador grafico	Aqua	Windows Integrated adapter	X11 System de X.Org

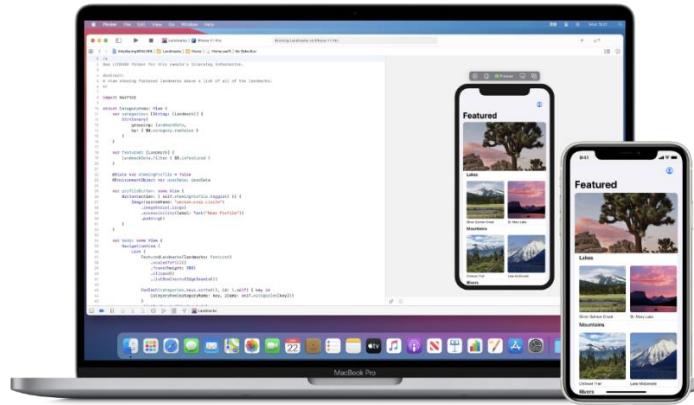
% de usuarios en el mundo	16.91%	76.26%	1.91%
% de usuarios en Mexico	13.95%	78.52%	2.63%
Sistemas de archivos soportados	NTFS (solo lectura), HFS+, FAT32, exFAT	FAT, FAT32, NTFS, exFAT	Ext4, FAT32, NTFS
Firewall	Sí, integrado	Sí, integrado	ufw - Uncomplicated Firewall

1.1.2 XCODE



DEFINICIÓN

XCode es un IDE que provee un conjunto de herramientas para desarrollar aplicaciones para dispositivos Apple, ofreciendo diseño de interfaces, manejo de código, testeo y debugging de aplicaciones.



HISTORIA

XCode tiene un nombre muy peculiar debido a su historia, la cual empieza dentro una empresa llamada NeXT Computer que creó dos aplicaciones llamadas Project builder e Interface builder que tenían como objetivo la creación de aplicaciones para su sistema operativo NeXTSTEP.



En 1997 Apple decide comprar la compañía NeXT computer y empezar a convertir el sistema operativo NeXSTEP a lo que hoy se conoce como OS X, de igual forma adaptar Project builder e Interface builder a su nuevo sistema operativo. No es hasta la versión OS X Panther (10.3) que Apple decide remplazar el nombre de Project builder a XCode en donde la X hace referencia al sistema operativo de OS X y la palabra code a la finalidad de la aplicación. Posteriormente en el lanzamiento de OS X 10.6 se decide

integrar la aplicación Interface builder a XCode ya que trabajaban de una manera totalmente independiente de XCode y en ocasiones se presentaban conflictos entre los dos.

¿Por qué no llamarlo simplemente iCode?

El nombre XCode se establece desde antes que los productos empezaran a inicializarse con la letra “i”, sin embargo, ésta no es la principal razón de que su nombre no fuese iCode, ya que Apple no tiene los derechos como para nombrar a todos sus productos con una letra “i” en sus iniciales, como lo es en el caso de iCode, la cual es una compañía China que ya posee los derechos de ese nombre.

VERSIONES

1.x Series: Xcode 1.0 se lanzó en otoño de 2003. Xcode 1.0 se basaba en Project Builder, pero tenía una interfaz de usuario (UI) actualizada, soporte de compilación distribuido e indexación de Code Sense.

Lenguajes soportados: C, Objective-C, C++, Objective-C++.

2.x Series: Xcode 2.0 fue lanzado con Mac OS X v10.4 "Tiger". Incluía el lenguaje de programación visual Quartz Composer, una mejor indexación de Code Sense para Java y compatibilidad con Ant. También incluyó la herramienta Apple Reference Library, que permite buscar y leer documentación en línea desde el sitio web de Apple y la documentación instalada en una computadora local.

Xcode 2.1 podría crear archivos binarios universales. Admitía encabezados precompilados compartidos, objetivos de pruebas unitarias, puntos de interrupción condicionales y puntos de observación.

Lenguajes Soportados: C, Objective-C, C++, Objective-C++.

3.x Series: Xcode 3.0 se lanzó con Mac OS X v10.5 "Leopard". Los cambios notables desde 2.1 incluyen la herramienta de depuración DTrace (ahora llamada Instrumentos), soporte de refactorización, documentación sensible al contexto y Objective-C 2.0 con recolección de basura.

Xcode 3.1 fue una versión actualizada de las herramientas de desarrollo para Mac OS X, y era la misma versión incluida con el iPhone SDK. Podría apuntar a plataformas que no sean Mac OS X, incluido iPhone OS 2.0. Otra característica nueva desde Xcode 3.0 es que el soporte SCM de Xcode ahora incluye Subversión 1.5.

Lenguajes soportados: C, Objective-C, C++, Objective-C++.

4.x Series: En junio de 2010, se anunció la versión 4 de Xcode durante el discurso del Estado de la Unión de Herramientas para Desarrolladores. La versión 4 de las herramientas de desarrollo consolida las herramientas de edición de Xcode y el Interface Builder en una sola aplicación, entre otras mejoras. Apple lanzó la versión final de Xcode 4.0 el 9 de marzo de 2011.

El 29 de agosto de 2011, Xcode 4.1 se puso a disposición para Mac OS X Snow Leopard para los miembros de los programas pagados para desarrolladores de Mac o iOS.

Xcode 4.1 fue la última versión que incluyó GNU Compiler Collection (GCC) en lugar de solo LLVM GCC o Clang.

Xcode 4.4 incluye soporte para la síntesis automática de propiedades declaradas, nuevas características de Objective-C como sintaxis literal y subíndice, localización mejorada.

Lenguajes soportados: Objective-C, C++, Objective-C++ y en la versión 4.2 se podía migrar código de aplicación hacia el lenguaje ARC.

5.x Series: El 18 de septiembre de 2013, se lanzó Xcode 5.0. Se envió con los SDK de Mountain Lion para iOS 7 y OS X 10.8. Sin embargo, el soporte para OS X 10.9 Mavericks solo estaba disponible en las versiones beta. Xcode 5.0 también agregó una versión de Clang que genera código ARM de 64 bits para iOS 7. Apple eliminó el soporte para construir binarios Cocoa recolectados de basura en Xcode 5.1.

Lenguajes Soportados: Lenguaje C, Lenguaje ARC y C++ (en anteriores versiones al 5.1 había un problema, Python y Ruby gem se utilizaban para hacer opciones de línea de comando, pero provocaban un problema al compilar).

6.x Series: El 2 de junio de 2014, Apple anunció la versión 6 de Xcode. Una de las características más notables fue la compatibilidad con Swift, un lenguaje de programación completamente nuevo desarrollado por Apple. Xcode 6 también incluyó funciones como áreas de juegos y herramientas de depuración en vivo. El 17 de septiembre de 2014, al mismo tiempo, se lanzaron iOS 8 y Xcode 6. Xcode se puede descargar en la Mac App Store.

Lenguajes Soportados: Swift(read-eval-print loop) reemplazando a C y a Objective-C, ARC usado para la administración de memoria.

7.x Series: El 8 de junio de 2015, en la Apple Worldwide Developers Conference, se anunció la versión 7 de Xcode. Introdujo soporte para Swift 2 y Metal para OS X, y agregó soporte para la implementación en dispositivos iOS sin una cuenta de desarrollador de Apple. Xcode 7 fue lanzado el 16 de septiembre de 2015

Lenguajes soportados: Swift 2, Objective-C, C, C++.

8.x Series: El 13 de junio de 2016, en la Conferencia Mundial de Desarrolladores de Apple, se anunció la versión 8 de Xcode; se lanzó una versión beta el mismo día. Introdujo soporte para Swift 3. Xcode 8 fue lanzado el 13 de septiembre de 2016.

Lenguajes soportados: Swift 3, Objective-C, C++, C, Python

9.x Series: El 5 de junio de 2017, en la Conferencia Mundial de Desarrolladores de Apple, se anunció la versión 9 de Xcode; se lanzó una versión beta el mismo día. Introdujo soporte para Swift 4 y Metal 2. También introdujo la depuración remota en dispositivos iOS y tvOS de forma inalámbrica, a través de Wi-Fi. Xcode 9 se lanzó públicamente el 19 de septiembre de 2017.

Lenguajes soportados: Swift 4, Objective-C, C++, Python.

10.x Series: El 4 de junio de 2018, se anunció la versión 10 de Xcode; se lanzó una versión beta el mismo día. Xcode 10 introdujo soporte para el Modo Oscuro anunciado para macOS Mojave, las plataformas de colaboración Bitbucket y GitLab (además de GitHub ya compatible), modelos de aprendizaje automático de entrenamiento de patios de recreo y las nuevas funciones en Swift 4.2 y Metal 2.1, así como mejoras al editor y al sistema de construcción del proyecto. Xcode 10 también dejó de admitir la creación de aplicaciones macOS de 32 bits y ya no admite la integración de Subversion. Xcode 10 se lanzó públicamente el 17 de septiembre de 2018.

Lenguajes soportados: Swift 4.2, Swift 4, Swift 5

11.x Series: El 3 de junio de 2019, se anunció la versión 11 de Xcode; se lanzó una versión beta el mismo día. Xcode 11 introdujo soporte para las nuevas funciones en Swift 5.1, así como el nuevo marco SwiftUI. También es compatible con la creación de aplicaciones para iPad que se ejecutan en macOS; incluye soporte integrado para Swift Package Manager; y contiene más mejoras en el editor, incluido un "minimap" que ofrece una descripción general de un archivo de código fuente con navegación rápida. Xcode 11 requiere macOS 10.14 o posterior y Xcode 11.4 requiere 10.15 o posterior. Xcode 11 se lanzó públicamente el 20 de septiembre de 2019.

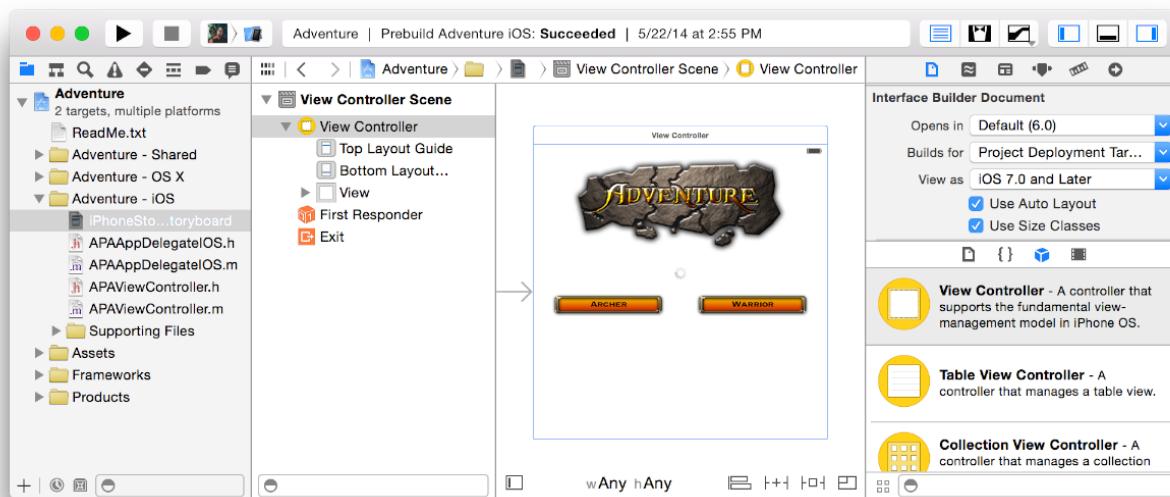
Lenguajes soportados: Swift 5.1, Swift 4.2, Swift 4.

12.x Series: El 22 de junio de 2020, en la Conferencia Mundial de Desarrolladores de Apple, se anunció la versión 12 de Xcode; se lanzó una versión beta el mismo día. Xcode 12 introdujo soporte para Swift 5.3 y requiere macOS 10.15.4 o posterior. Xcode 12 se lanzó públicamente el 16 de septiembre de 2020.

Lenguajes soportados: Swift 5.3, Swift 4.2, Swift 4.

NIB, XIB Y STORY BOARD

El archivo de interface de usuario tiene una extensión de tipo. storyboard o. xib, en donde un archivo. xib usualmente especifica un controlador de vista o un menú, mientras que un storyboard es un poco más robusto puesto que especifica un conjunto de vistas de controles. A diferencia de. xib un storyboard puede contener muchas vistas y transiciones entre ellos.



XCode maneja el contenido de los archivos .xib y .storyboard en un formato XML para que posteriormente al compilar el proyecto transformar esos archivos en archivos binarios llamados .nibs, en donde dichos archivos .nibs son cargados de ser inicializados para crear nuevas vistas.

APP STORE

La App Store es una plataforma de distribución digital desarrollada y mantenida por Apple, la cual permite a los usuarios con dispositivos Apple buscar y descargar aplicaciones creadas con el kit de desarrollo de software de iOS.

App Store ha estado disponible desde el 10 de julio del 2008 con un total 500 aplicaciones disponibles, dicho número ha crecido significativamente a millones en la actualidad.



Para cargar una aplicación en la App Store será necesario considerar lo siguiente:

- El servicio de Apple Developer program para el desarrollo de las aplicaciones.
- Un ordenador Mac en conjunto de XCode para convertir la aplicación en un archivo binario.
- Se debe adquirir una membresía que permita publicar aplicaciones dentro de la App Store, la cual tiene un costo de 99 dólares anuales para la particular y 299 dólares para la Enterprise.
- Una vez adquirida la membresía, debes asegurar que tu aplicación tenga los certificados oficiales de Apple firmados (certificado de desarrollo y distribución), los cuales son generados desde el software de XCode.
- Se debe registrar un dispositivo Apple en la cuenta del desarrollador, ya que será necesario para que se pueda probar la aplicación.
- Por medio de iTunes connect se hará la publicación de la aplicación así que será necesario esta plataforma.
- Si se requiere distribuir una app de pago en la App Store será necesario revisar un apartado llamado "Agreements, tax and baking" de iTunes connect.
 - Se tiene que elegir un contrato que corresponda a tu aplicación dentro de iTunes connect.
- Se necesita un registro personal para iTunes connect que contenga toda la información para el despliegue y gestión de la app, básicamente se llena información acerca de la aplicación, tales como capturas de pantalla, público al que va dirigido, restricciones de edad, etc.
- Se necesita una revisión por parte del equipo encargado de Apple y así lograr el visto bueno de ellos para publicarla.
- Si la app llega a ser rechazada el equipo de Apple te hará saber las causas del por qué no fue posible su publicación y qué hacer para corregirlo.

ESTRUCTURA DE EMPAQUETADO

Dentro de una aplicación empaquetada se guarda todo lo que una aplicación necesita para funcionar correctamente.

Hay que considerar también que existen tres tipos de aplicaciones iOS:

1. Aplicaciones nativas

Fueron creadas con Objective-C o Swift.

2. Aplicaciones híbridas

Usan frameworks como Xamarin, Cordova, Flutter, etc. Acompañados de Objective-C y Swift.

3. Aplicaciones basadas en la web

Son aplicaciones web responsivas creadas especialmente para trabajar con dispositivos móviles.

Tipos de archivos que se encuentran dentro del empaquetado:

- **Archivo Info.plist**

Requerido ya que información listada en un archivo estructurado que contiene configuración de la aplicación.

- **Ejecutable**

Contiene la aplicación principal o punto de entrada y/o cualquier otra aplicación enlazada.

- **Archivos de recursos**

Son archivos que viven fuera del archivo ejecutable, a este tipo de archivos se les suele relacionar con imágenes, sonidos, archivos de cadena, archivos de configuración, etc.

Bundle container

Directorio que consiste en todos los archivos que vienen con la aplicación cuando se instala desde la App Store u otra fuente.

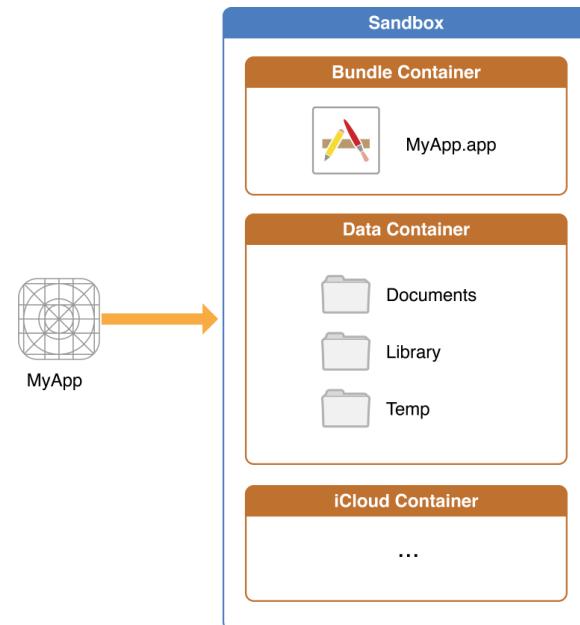
- **iTunesMetadata.plist**

Es un archivo usado para proveer información a iTunes acerca de una aplicación.

- **iTunesArtWork**

Archivo que contiene la imagen que es usada para representar la aplicación en iTunes.

- META-INF
Es una carpeta que contiene dos archivos de la metadata de la aplicación.
- .app
Carpeta que mantiene todos los componentes de la aplicación.
- Aplicación binaria
El archivo contiene la aplicación ejecutable.
- Info.plist
Archivo de manifiesto para la aplicación, contiene todo acerca de dispositivos soportados, bundle ID, nombre a mostrar, etc.
- Icono de la aplicación
- Imágenes de lanzamiento
Usadas para mostrarse cuando la aplicación es lanzada y se encuentra cargando la interfaz gráfica.
- Storyboard/nib files
Estos archivos contienen información acerca de las vistas.



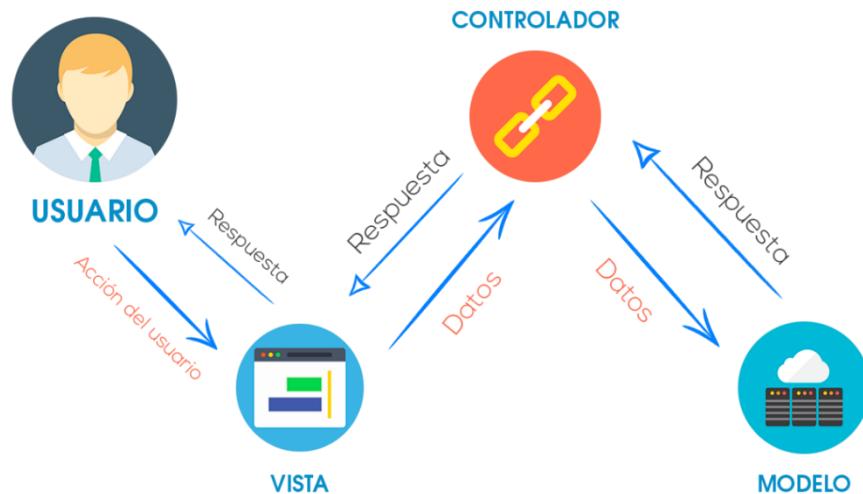
Data container

Es donde se almacena los datos locales y consiste en archivos que el desarrollador desea que sean guardados en el dispositivo del usuario.

MODELO VISTA CONTROLADOR

Es un patrón de arquitectura de software basado en tres componentes principales los cuales son: vistas, modelos y controladores. Esto con el fin de lograr una separación de la lógica de la aplicación y la vista.

Actualmente es una arquitectura muy utilizada e importante puesto a que puede realizar componentes gráficos básicos hasta sistemas robustos pensados para empresas.



Este patrón tiene la facilidad y ventaja de que al hacer un cambio dentro del proyecto no llegue a afectar a otra parte del mismo.

Modelo

Es el encargado de los datos y generalmente consulta a una base de datos, sin embargo, esta capa es totalmente independiente a el almacenamiento de los datos.

Aquí es donde las reglas o funcionalidades del sistema se definen, llevando también un conjunto de registro de las vistas y controladores del sistema.

Controlador

Es el encargado de recibir las órdenes del usuario y se los hace pasar al modelo de una forma que pueda entender dichas instrucciones del usuario.

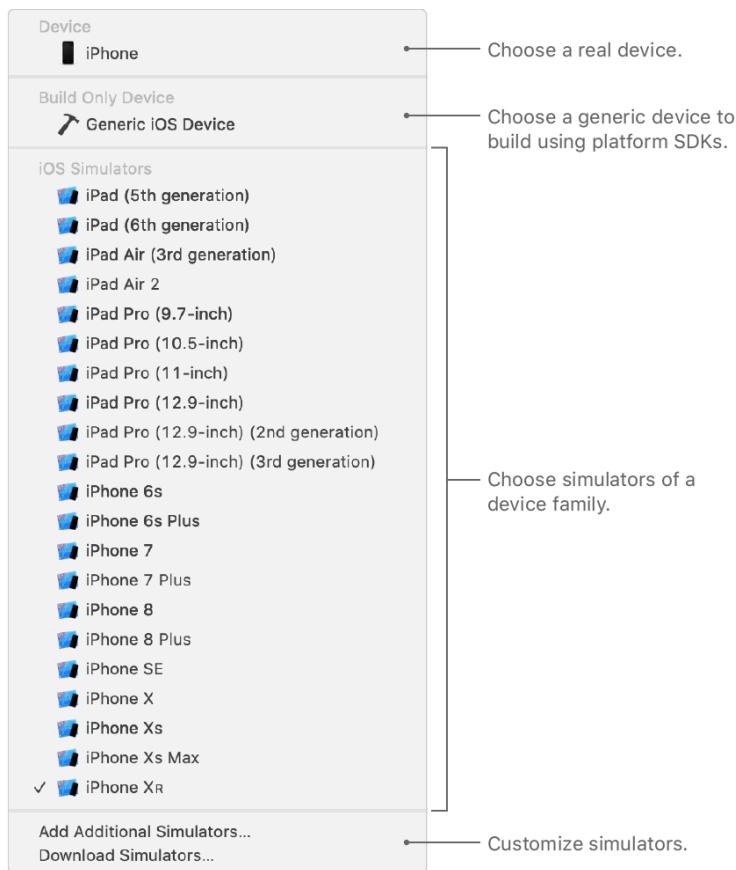
Vistas

Son el conjunto de representaciones visuales de resultado, todo lo que tenga que ver con la interfaz gráfica vendría aquí.

SIMULADORES SOPORTADOS

Para las aplicaciones iOS, tvOS y watchOS, puede elegir un dispositivo simulado, en Simuladores, en el menú de destino de ejecución junto al menú de esquema en la barra de herramientas.

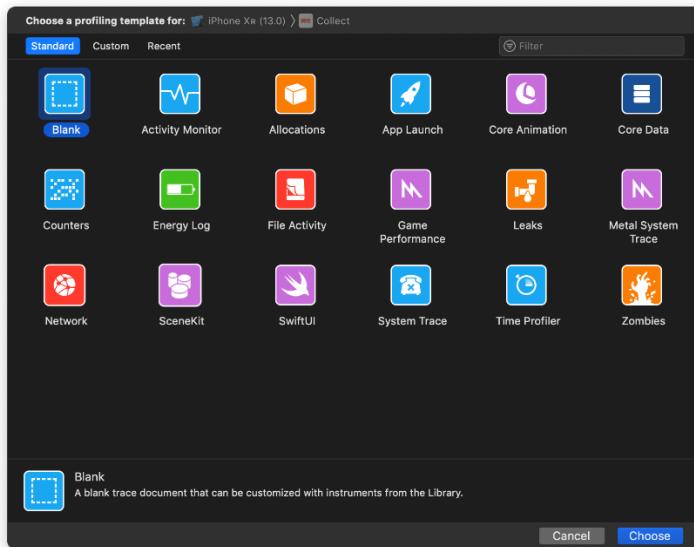
Para agregar simuladores adicionales de una familia de productos que ejecutan versiones anteriores del sistema operativo, elija Agregar simuladores adicionales.



INSTRUMENTS

Instruments es una herramienta de prueba y análisis de rendimiento potente y flexible que forma parte del conjunto de herramientas de Xcode. Está diseñado para ayudarlo a perfeccionar sus aplicaciones, procesos y dispositivos iOS, watchOS, tvOS y macOS con el fin de comprender y optimizar mejor su comportamiento y rendimiento. La incorporación de Instruments en su flujo de trabajo desde el comienzo del proceso de desarrollo de la aplicación puede ahorrarle tiempo más adelante al ayudarlo a encontrar problemas al principio del ciclo de desarrollo.

En Instruments, utiliza herramientas especializadas, conocidas como instrumentos, para rastrear diferentes aspectos de sus aplicaciones, procesos y dispositivos a lo largo del tiempo. Instruments recopila datos a medida que crea perfiles y le presenta los resultados en detalle para su análisis.



A diferencia de otras herramientas de rendimiento y depuración, Instruments le permite recopilar tipos de datos muy dispares y verlos uno al lado del otro. Esto facilita la identificación de tendencias que de otro modo podrían pasarse por alto. Por ejemplo, su aplicación puede mostrar un gran crecimiento de memoria debido a múltiples conexiones de red abiertas. Al usar los instrumentos Allocations y Connections juntos, puede identificar las conexiones que no se cierran y, por lo tanto, dan como resultado un rápido crecimiento de la memoria.

Al utilizar Instruments de forma eficaz, puede:

- Examinar el comportamiento de una o más aplicaciones o procesos.
- Examinar funciones específicas del dispositivo, como Wi-Fi y Bluetooth
- Realice la creación de perfiles en un simulador o en un dispositivo físico
- Localice problemas en su código fuente
- Realice un análisis de rendimiento en su aplicación
- Encuentra problemas de memoria en tu aplicación, como fugas, memoria abandonada y zombis.
- Identifique formas de optimizar su aplicación para una mayor eficiencia energética
- Realice la resolución de problemas generales a nivel del sistema
- Guardar configuraciones de instrumentos como plantillas

Aunque está integrado y se puede usar con Xcode, Instruments es una aplicación separada, que se puede usar de forma independiente según sea necesario.

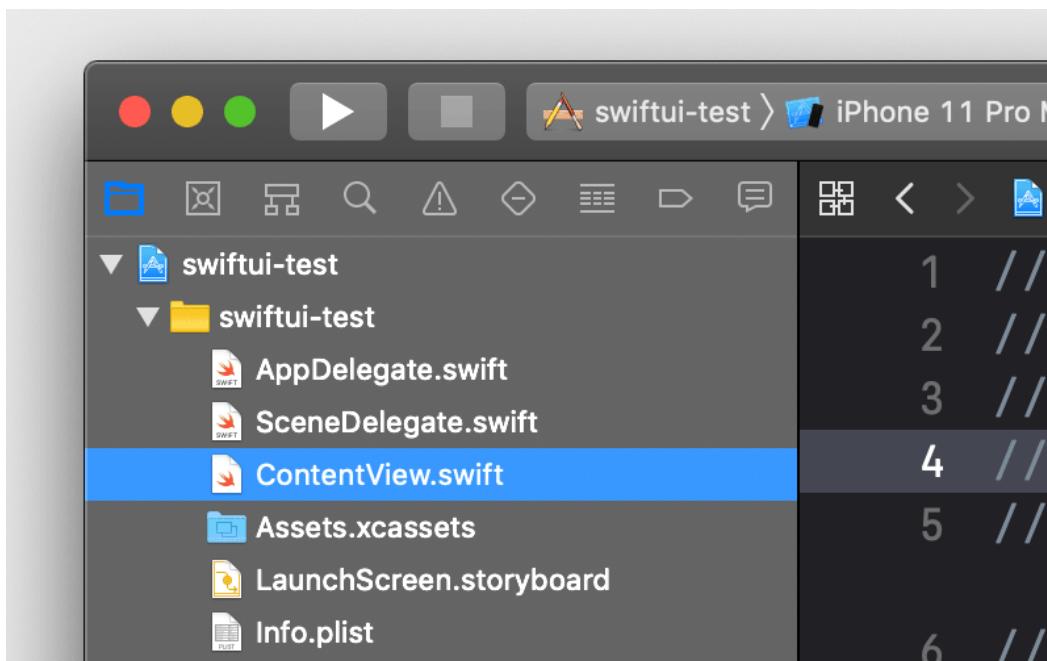
PROGRAMA DE DESARROLLADORES DE APPLE

Al inscribirse en Apple Developer Program, las personas y las organizaciones reciben todo lo que necesitan para desarrollar apps y poder distribuirlas. Los miembros pueden distribuir apps en App Store para iPhone, iPad, Mac, Apple Watch, Apple TV y iMessage. También pueden distribuir software fuera de Mac App Store, así como ofrecer apps personalizadas a empresas específicas y apps exclusivas a sus empleados a través de Apple Business Manager.

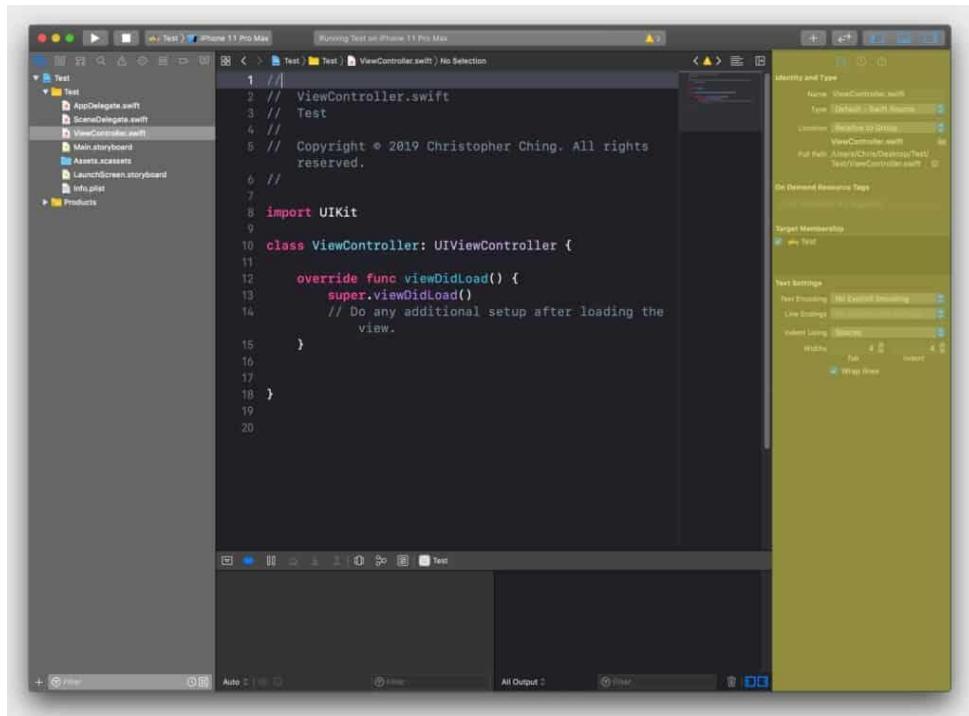
VENTANA DE UN PROYECTO

Panel de navegación

Es aquí donde se ven todos los archivos asociados al proyecto, siempre será visible esta pestaña cuando se crea un nuevo proyecto en Xcode:



Panel de utilerías

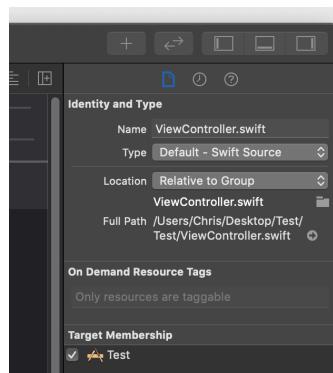


Este está en el lado derecho de la pantalla, este panel de utilidades tiene una variedad amplia de Inspectores que nos ayudarán a ver detalles del archivo, elemento o pieza de código que estemos viendo

Algunos inspectores que tenemos son:

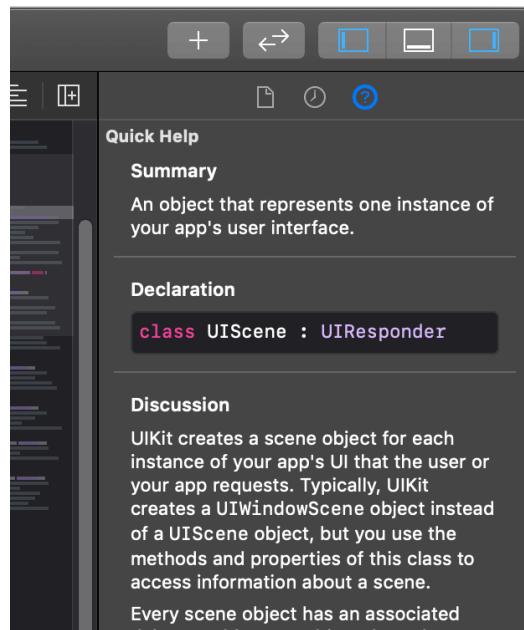
- Inspector de archivos

Este nos ayuda ver los detalles del archivo que este seleccionado en el navegador de archivos



- Inspector de ayuda rápida

Este inspector te muestra parte de la documentación del método, clase o keyword en la que se encuentra el cursor actualmente



Panel de depuración

El panel de depuración muestra una consola y el estado de distintas variables cuando una aplicación está en ejecución.



1.1.3 SWIFT



DEFINICIÓN

Swift es un lenguaje de programación multi paradigma propietario de Apple para el desarrollo de aplicaciones de IOS, Mac, Apple TV y Apple Watch.

HISTORIA

Swift fue presentado en 2014 por la empresa Apple en su famoso evento Worldwide Developers Conference, sin embargo, la historia comienza en 2010 gracias Chris Lattner con ayuda de otros programadores en Apple. La idea principal del lenguaje de programación Swift fue tomar lo mejor de lenguajes de programación ya existentes, tales como Objective-C, Rust, Haskell, Ruby, Python, C#, CLU y entre otros.



El nombre Swift hace referencia a un tipo de pájaro y que su vez su adjetivo significa rápido, haciendo referencia a que el código que genera es a menudo más rápido en tiempo de ejecución que el generado por el compilador de Objective-C.

La primera versión de Swift fue la 1.0 la cual tuvo una actualización significativamente notoria a Swift 2.0 en la Worldwide Developers Conference de 2015, para después en su versión 2.2 declararse un proyecto de código abierto para la comunidad.

En la versión 3.0 de Swift hubo una evolución significativa en la sintaxis permitiendo así también sobrepasar la popularidad de Objective C en 2018.

Swift en su versión 4.0 ha presentado cambios severos en donde se pueden actualizar códigos escritos en versiones antiguas de Swift por medio de la funcionalidad de migración que provee XCode.

Versión	Lanzamiento
Swift 1.0	2014- 09-09
Swift 1.1	2014-10-22
Swift 1.2	2015-04-08
Swift 2.0	2015-09-21
Swift 3.0	2016-09-13
Swift 4.0	2017-09-19
Swift 4.1	2018-03-29
Swift 4.2	2018-09-17

PARADIGMAS DE PROGRAMACIÓN SOPORTADOS

Swift se considera un lenguaje de programación multiparadigma pues esta soportada programación orientada a protocolos, orientada a objetos, programación funcional y programación imperativa.

Programación orientada a objetos

Es un paradigma de la programación la cual se define como una manera de programar específica en donde se organiza el código en unidades denominadas clases y de las cuales se crean objetos que se relacionan entre sí para conseguir objetivos de aplicaciones.



Se basa en los siguientes pilares:

- Abstracción

Consiste en el proceso de ocultar lo innecesario y mostrar solo los detalles relevantes.

- Encapsulación

Práctica que envuelve la información de código que se manipula y lo guarda de manera segura.

- Herencia

Mecanismo por el cual los objetos adquieren comportamientos y propiedades de otros objetos.

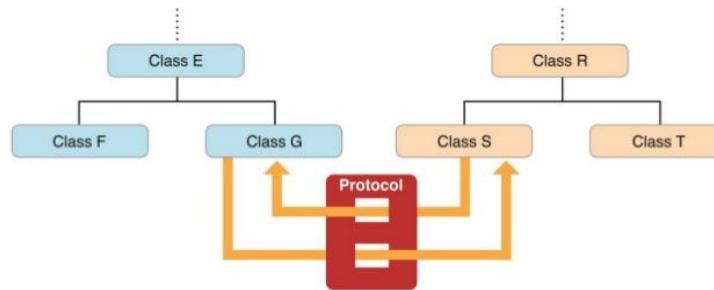
- Polimorfismo

Es la manera de hacer un proceso de varias formas para abarcar distintos casos de uso.

Programación orientada a protocolos

Apple asegura que Swift es el primer lenguaje de programación orientado a protocolos la cual tiene como fin el combatir los problemas que conlleva una programación orientada a objetos, como ejemplo están el problema de herencia múltiple, que las clases son tipos de referencia y dificultades para tests unitarios en un gran acoplamiento entre clases.

Principalmente consiste en buscar otra forma de acceder a la abstracción de métodos y propiedades, pero a través de tipos de datos por valor y que no funcionan por referencia. El manejo de este tipo de datos es más eficiente en memoria ya que evitamos ciclos de retención, referencias perdidas, etc.



Dato que sirve y está en ámbito, está, dato que no, se borra. Para conseguir esto se sustituyen clases por structs, que en Swift son idénticos en todos los aspectos salvo en tres: las clases son datos por referencia y los structs son tipos de datos por valor (con entidad única y que residen como tales en la propia variable o constante). Además, una clase soporta herencia, pero un struct no, por lo que la clase obliga a inicializar sus propiedades cuando se crea su implementación, pero un struct no.

TIPOS DE LICENCIAMIENTO

Apple developer

Es completamente gratuita y únicamente requiere registrarse en la página web de Apple para así poder tener acceso a documentación técnica, herramientas de desarrollo y un simulador de XCode para probar aplicaciones

iOS University Program

Completamente gratuita, aunque solo aplica para determinadas universidades que se encuentran incluidas por Apple. El propósito de este licenciamiento es la formación de alumnos en las herramientas de Apple.

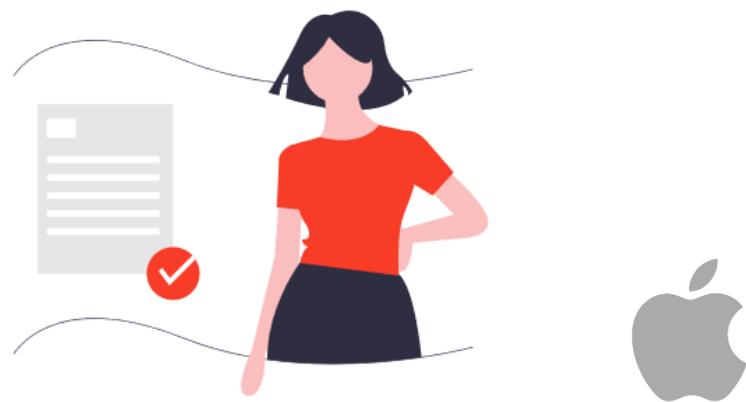
El licenciamiento ofrece probar aplicaciones en nuestros dispositivos.

iOS Developer Program

Tiene un costo de 99 dólares al año y ofrece adicionalmente documentación técnica, herramientas de desarrollo, descarga de firmwares, instalar aplicaciones en nuestros dispositivos registrándolos previamente en la web de Apple y buscar en la App Store nuestras aplicaciones y centralizarlas con unos beneficios del 70% sobre el previo de venta que establezcamos.

iOS Developer Enterprise Program

Tiene un costo de 299 dólares al año y está orientada al desarrollo de aplicaciones tipo in-house o bien aplicaciones corporativas privadas ya que nos ofrece publicar de forma privada a la empresa aplicaciones que puedan ser instaladas en los dispositivos de los empleados.



PLAYGROUND

Es una herramienta educacional para el entorno de desarrollo Swift desarrollado y lanzada por Apple en su evento Worldwide Developers Conference en 2016.

Este proyecto fue presentado como una aplicación para iPad en conjunto de iOS 10 para posteriormente lanzar en febrero de 2020 una versión para macOS, adicional de ello Apple publica lecciones de aprendizaje para desarrollar habilidades de programación y debugging.

Nos permite visualizar una ventana que nos dice el resultado de cada una de las líneas que introducimos en el programa y cada vez que editamos nuestro código, este recompila, evalúa y nos muestra el resultado en adición a ello, también nos permite visualizar graficas de rendimiento en nuestro código.

Esta herramienta tiene una gran versatilidad como herramienta y son perfectas para la formación pues es muy fácil probar y practicar dentro del entorno de Swift.

Playground se conforma de un paquete de archivos dentro de una extensión. playground que incluye varios elementos:

- Código Swift que se está programando.
- Carpeta de recursos para acceder dentro de nuestro contenedor.
- Carpeta de documentación en HTML y CSS que permite unir código Swift a documentación en un mismo documento Playground del editor.
- Timeline de resultados.



SINTAXIS

Comentarios

Hay dos maneras de escribir comentarios dentro del lenguaje de programación Swift, los cuales pueden ser de una sola línea o de más de una línea.

El comentario de una sola línea se define agregando dos slash y el comentario de varias líneas se hace por medio del encapsulado del comentario dentro de dos slash y dos asteriscos.

```
● ● ●  
// Esto es un comentario de una línea  
/*  
    Esto es un comentario de varias líneas  
*/
```

Punto y coma

El punto y coma se puede ser opcional solo sí la línea abarca solo una sentencia, de lo contrario será obligatoria.

```
● ● ●  
  
let variable = 1  
let variable2 = 2; print(variable2);
```

Nomenclatura e identificadores

Identificador es dar a las variables, constantes, métodos, funciones, enumeración, que especifica el nombre de la estructura, clase y otros protocolos

Consideraciones:

- Entre mayúsculas y minúsculas, Minombre con mi_nombre son dos identificadores diferentes.
- Los identificadores primer personaje puede (_) o las letras comienzan con un guión bajo, pero no puede ser un número.
- Los identificadores pueden ser otros caracteres de subrayado (_), letras o números.

Unicode se llama el sistema de codificación unificado, que contiene la codificación de texto de Asia, como el chino, japonés, coreano y otros personajes, e incluso emoticonos.

LITERALES

Literales numéricicos enteros

Tenemos 4 diferentes formas de escribir literales numéricicos enteros: decimales, binarios, octal y hexadecimal.

```
● ● ●  
let decimalEntero = 17  
let binarioEntero = 0b10001  
let octalEntero = 0o21  
let hexadecimalEntero = 0x11
```

Literales numéricicos decimales

Tenemos tres formas de interpretarlos: decimales, con exponente y hexadecimal.

```
● ● ●  
let decimalDouble = 12.1875  
let exponenteDouble = 1.21875e1  
let hexadecimalDouble = 0xC.3p0
```

Los literales numéricos pueden tener un tipo de formato para que sea más sencillo de leer, en donde los dos enteros y los dos flotantes pueden estar separados con ceros y contener guiones bajos para ayudar a la lectura del mismo.

```
● ● ●  
let paddedDouble = 000123.456  
let unMillon = 1_000_000  
let unPocoMasDeUnMillion = 1_000_000.000_000_1
```

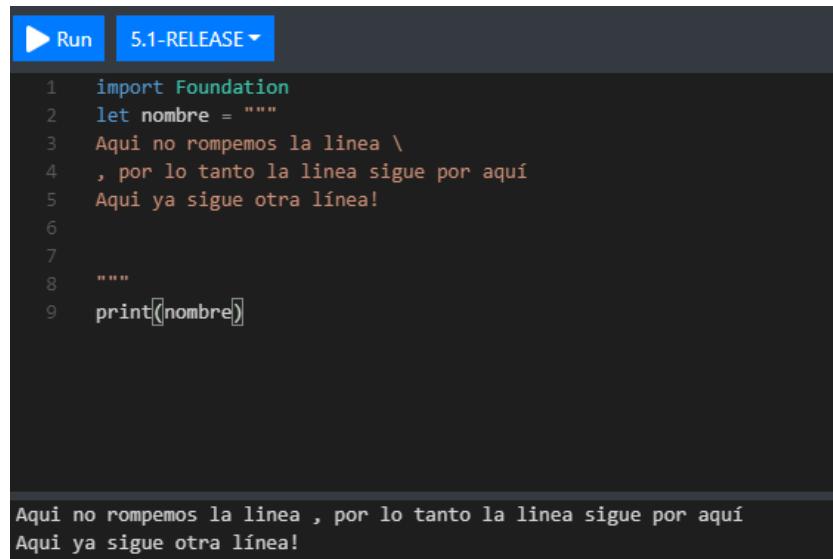
Literal multilínea

Al contrario que otros lenguajes de programación Swift es capaz de romper líneas sin el uso de los caracteres “\n”, esto se puede hacer usando el encapsulamiento del texto dentro de tres comillas dobles.

```
● ● ●  
let ejemplo = """  
Hola, soy Felipe y esto  
es el uso de la literal  
multilinea ;)  
"""
```

Ignorar el rompimiento de líneas

Para lograr esto habría que colocar el carácter especial “\” al final de cada línea para ignorar el rompimiento de línea.



The screenshot shows a Jupyter Notebook cell with the following code:

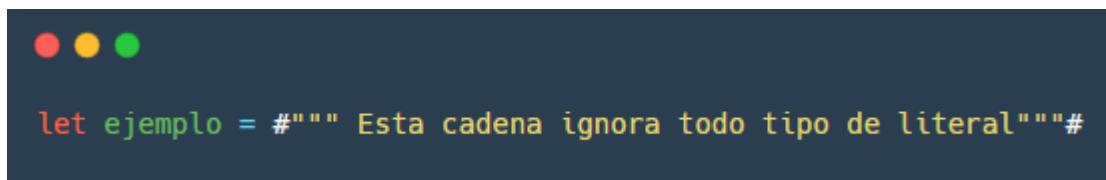
```
1 import Foundation
2 let nombre = """
3 Aqui no rompemos la linea \
4 , por lo tanto la linea sigue por aquí
5 Aqui ya sigue otra linea!
6
7 """
8
9 print(nombre)
```

The output of the code is displayed below the cell:

```
Aqui no rompemos la linea , por lo tanto la linea sigue por aquí
Aqui ya sigue otra linea!
```

Raw string

Es la forma en que nosotros podemos ignorar todo tipo de literal dentro de un String y tomarlo como un simple carácter, para ello será necesario encapsular nuestra cadena dentro de 2 caracteres “#”.



The screenshot shows a Jupyter Notebook cell with the following code:

```
let ejemplo = #"" Esta cadena ignora todo tipo de literal""#
```

TIPOS DE DATOS

Character

Representa un solo carácter.



```
let genero: Character = "H"
```

Bool

Permite saber dos posibles valores: verdadero o falso.



```
let haceFrio: Bool = true
```

String

Cadena de caracteres tratada como una matriz de ellos.



```
let nombre: String = "Felipe"
```

Numéricos

Tipo	Rango
Int8	-127 A 127
UInt8	0 A 255
Int16	-32768 A 32768
UInt16	0 A 65535
Int32	-2147483648 A 2147483647
UInt32	0 A 4294967295
Int64	-9223372036854775808 A 9223372036854775807
UInt64	0 A 18446744073709551615
Flotador	1.2E-38 A 3.4E + 38 (6 dígitos)
Doble	2.3E-1.7E + 308 A 308 (15 dígitos)

Alias

Define un nombre alternativo para un tipo de dato ya existente.

```
● ● ●

import Foundation
typealias entero16 = UInt16

let n : entero16 = 2
print(n)
```

Inferencia de tipos

Swift hace uso de la inferencia para así permitir al compilador decidir el tipo de dato de una expresión en particular y automáticamente cuando se compile el código va a examinar los valores que se le ha dado a dicha expresión.

Declaración de variables y constantes

Tanto variables como constantes, deben ser declaradas antes de ser utilizadas, para declarar constantes se utiliza la palabra reservada “let”, y para declarar variables se hace uso de la palabra reservada “var”.

```
● ● ●

let miConstante = "Hola, soy una constante"
var miVariable = "Hola, soy una variable"
```

Se pueden declarar múltiples variables y múltiples constantes en una misma línea separándolas por medio de comas.

```
● ● ●

let c1 = 1, c2 = 2, c3 = 3
var v1 = 1, v2 = 2, v3 = 3
```

Se puede colocar el tipo de dato si así se requiere a nuestra variable o constante, esto gracias a la anotación de tipo de dato la cual se logra poniendo nos puntos seguido del nombre de nuestra variable y colocando posteriormente el tipo de dato que queremos especificar explícitamente a nuestra variable o constante.

```
● ● ●

let edad: Int = 21
```

El nombrar variables y constantes puedes contener cualquier tipo de carácter UNICODE, sin embargo, se deben seguir las siguientes normas:

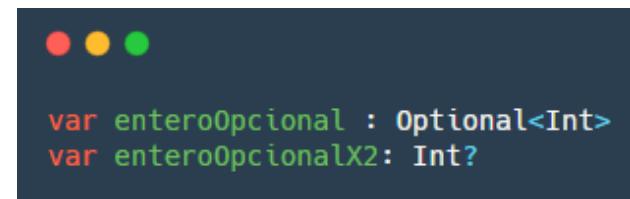
- No puede empezar con un número.
- No se pueden declarar dos identificadores con el mismo nombre.
- Los nombres de variables no pueden contener espacios en blanco.

Tipo de dato optional

El tipo optional permite definir si un valor puede o no puede estar presente dentro de una variable.

Dentro del lenguaje de programación Swift cualquier tipo de dato puede ser declarado explícitamente a opcional y al ser este declarado como opcional en automático se le provee un valor inicial de “nil” a dicha propiedad.

Existen dos maneras distintas de definir un tipo de dato opcional:



```
var enteroOpcional : Optional<Int>
var enteroOpcionalX2: Int?
```

La primera se define con la palabra reservada `Optional` y encapsulando entre una etiqueta el tipo de dato, mientras que la otra forma simplemente se agrega un signo de interrogación después del tipo de dato.

SECUENCIAS DE ESCAPE

Las combinaciones de caracteres que consisten en una barra diagonal inversa (\) seguida de una letra o una combinación de dígitos se denominan "secuencias de escape" y tienen como porpósito especificar accubres cini retirbis de carro y movimientos de tabulación. También se emplean para proporcionar representaciones literales de caracteres no imprimibles y de caracteres que normalmente tienen significados especiales, como las comillas dobles (").

- “\0” Para un carácter nulo.
- “\” Backslash.
- “\t” Tab horizontal.
- “\n” Salto de línea.
- “\r” Retorno.
- “\” “ Comillas dobles.
- “\” “Comilla siempre.
- “\{n\}” Valor escalar UNICODE, en donde n es un valor hexadecimal numérico.

Interpolación de cadena

Swift permite construir cadenas en combinación de variables, constantes y expresiones por medio de esta literal.

La manera de implementar esta técnica es encapsular nuestra valor o expresión de nuestro interés dentro de un paréntesis seguido de slash invertido, en cualquier parte de nuestra cadena de texto.

Esto permite un manejo sencillo a la hora de crear nuestras cadenas de texto que necesiten concatenar valores y/o expresiones.

```
● ● ●  
let nombre = "Felie"  
print("Hola, mi nombre es \(nombre) ")
```

OPERADORES ARITMÉTICOS

Swift soporta los siguientes operadores aritméticos:

4. “+” Adición
5. “-” Sustracción
6. “*” Multiplicación
7. “/” División.
8. “%” Resto.
9. “++” Incremento.
10. “--” Decremento.

OPERADORES RELACIONALES

Swift soporta los siguientes operadores relacionales:

- “==” Igualdad.
- “!=” Distinto.
- “>” Mayor que.
- “<” Menor que.
- “>=” Mayor o igual que.
- “<=” Menor o igual que.

OPERADORES LÓGICOS

Swift soporta los siguientes operadores lógicos:

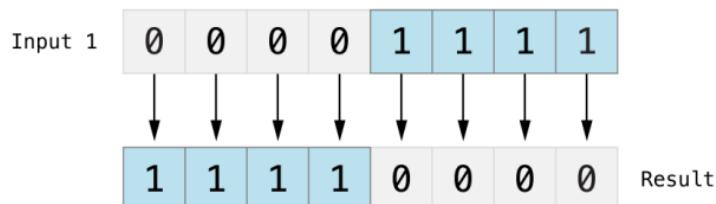
- “**&&**” La lógica y, en donde ambos lados deben ser verdaderos para que se cumpla la condición.
- “**||**” La lógica o, en donde al menos uno de los lados debe ser verdadero para que se cumpla la condición.
- “**!**” Operador lógico de negación, en donde invierte el valor actual de un booleano.
-

BIT A BIT (BITWISE)

Los operadores de este tipo nos permiten manipular de manera individual una fila de bits de datos que se encuentran dentro de una estructura de datos. Swift soporta todos los operadores que se encuentra en C.

Operador NOT (~)

Nos permite invertir todos los bits en un número.

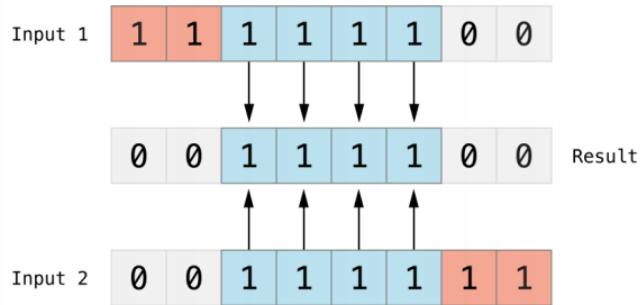


El ejemplo anterior quedaría de la siguiente manera:

```
let bitsIniciales: UInt8 = 0b00001111
let bitsInvertidos = ~bitsIniciales // Esto equivale a 11110000
```

Operador AND (&)

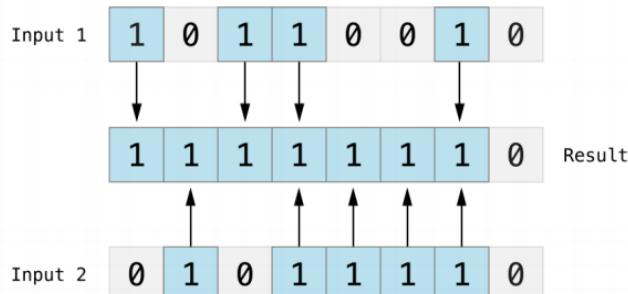
Combina los bits de dos números de tal manera que retorna un nuevo número en donde los bits se establecerán como 1 solo si los bits son iguales a 1 en las dos entradas.



```
let primerosSeisBits: UInt8 = 0b11111100
let ultimosSeisBits: UInt8 = 0b00111111
let cuatroBitsIntermedios = primerosSeisBits & ultimosSeisBits // Esto equivale a
00111100
```

Operador OR (|)

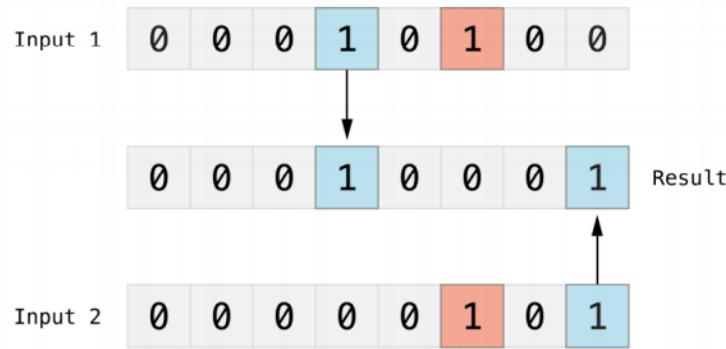
Compara los bits de dos números, en donde el número a regresar es un nuevo número en donde los bits que tienen en cualquiera de las dos entradas un uno se colocan como 1.



```
let primerosBits: UInt8 = 0b10110010
let segundosBits: UInt8 = 0b01011110
let bitsCombinados = primerosBits | segundosBits // Esto equivale a D11111110
```

Operador XOR (^)

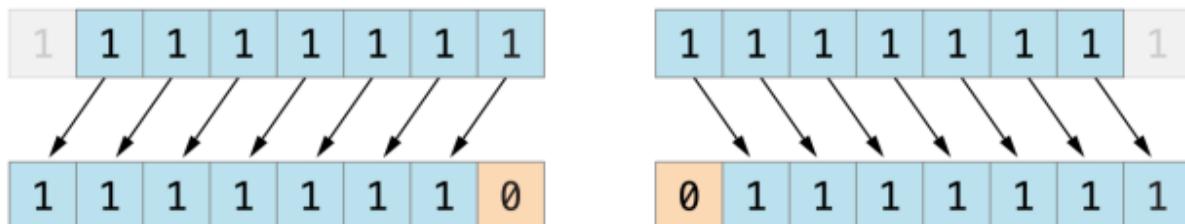
Compara los bits de dos números de tal forma que retorna un nuevo número en donde los bits se establecen como 1 cuando las entradas de bits son distintas y cuando no se establecen como 0.



```
let primerosBits: UInt8 = 0b00010100
let segundosBits: UInt8 = 0b00000101
let bitsResultantes = firstBits ^ otherBits // equivale a 00010001
```

Operador Bitwise Left y Right Shift

Mueven todos los bits de un numero hacia la derecha o a la izquierda por un cierto número de lugares de acuerdo a las reglas definidas.



```
let shiftBits: UInt8 = 4 // 00000100
shiftBits << 1 // 00001000
shiftBits << 2 // 00100000
```

OPERADORES DE ASIGNACIÓN

El operador de asignación permite inicializar o actualizar el valor de una variable.

Operadores de asignación soportados:

- “**=**” Operador de asignación simple, especifica el operando derecho asignado a la izquierda del operando.
- “**+ =**” Añadiendo después de la asignación, los lados izquierdo y derecho del operando antes de añadir asignan a la izquierda del operando.
- “**- =**” Después de asignación de resta, los operandos izquierdo y derecho en ambos lados del operando de la izquierda después de la sustracción asignados.
- “*** =**” Entonces multiplicar la asignación, los lados izquierdo y derecho de los operandos se multiplican antes de la asignación al operando izquierdo.
- “**/ =**” Dividido después de la asignación, los lados izquierdo y derecho del operando divisoria después de la asignación al operando de la izquierda.
- “**% =**” El resto después de la asignación, los lados izquierdo y derecho del operando resto después de la asignación al operando de la izquierda.
- “**<< =**” Izquierda en modo bit después de la asignación.
- “**>> =**” Derecha en modo bit después de la asignación.
- “**y =**” Y a nivel de bits de asignación después de la operación.
- “**^ =**” Bit a bit OR exclusiva del operador y luego asignado.
- “**| =**” OR bit a bit después de la asignación.

OPERADORES DE INTERVALOS

Operadores de intervalos cerrados

El operador de rango cerrado (`a ... b`) define un rango que va desde `a` hasta `b`, e incluye los valores `a` y `b`. El valor de `a` no debe ser mayor que `b`.

El operador de rango cerrado es útil cuando se realiza una iteración sobre un rango en el que desea utilizar todos los valores, como en un bucle `for-in`:

```
● ● ●

for index in 1...5 {

    print("\(index) multiplicado por 5 es \(index * 5)")

}
```

Entreabierto operador de intervalo

El operador de rango medio abierto (`a .. <b`) define un rango que se extiende de `a` hasta `b`, pero no incluye `b`. De ahí viene su nombre, se le llama rango semiabierto porque contiene su primer valor, pero no su valor final.

Al igual que con el operador de rango cerrado, el valor de `a` no debe ser mayor que `b`. Dicho esto, si el valor de `a` es igual a `b`, pues básicamente no existe rango alguno y por ende el rango resultante estará vacío.

```
● ● ●

let names = ["Fleipe", "Jessica", "Angel"]

let count = names.count

for index in 0..<count {

    print("La persona número \(index + 1) se llama \(names[index])")

}
```

Los rangos semiabiertos son particularmente útiles cuando se trabaja con listas basadas en cero (que comienzan por cero), como en los arreglos de datos, donde es útil contar hasta (pero sin incluir) la longitud de la lista:

Operador de Rango Unilateral

El operador de rango cerrado tiene una forma alternativa para los rangos que continúan lo más lejos posible en una dirección, por ejemplo, un rango que incluye todos los elementos de un arreglo desde el índice 2 hasta el último elemento.

En estos casos, puede omitirse el valor de un lado del operador de rango. Este tipo de rango se denomina rango de un solo lado o unilaterales porque el operador tiene un valor en un solo lado. Por ejemplo:

```
● ● ●

let names = ["Fleipe", "Jessica", "Angel"]

print("Nombres a partir del segundo elemento:")

for name in names[2...] {

    print(name)

}

print("\nNombres hasta el segundo elemento:")

for name in names[...2] {

    print(name)

}
```

Condicional ternaria

Swift maneja un operador ternario la cual es definida con un signo de interrogación en conjunto del carácter de dos puntos para definir otro caso.

La estructura sería la siguiente:

```
● ● ●

var condicion: Bool = true

condicion ? print("primer caso") : print("segundo caso")
```

DESICIONES

Utilizados para ejecutar distintas partes del código dependiendo de ciertas condiciones.

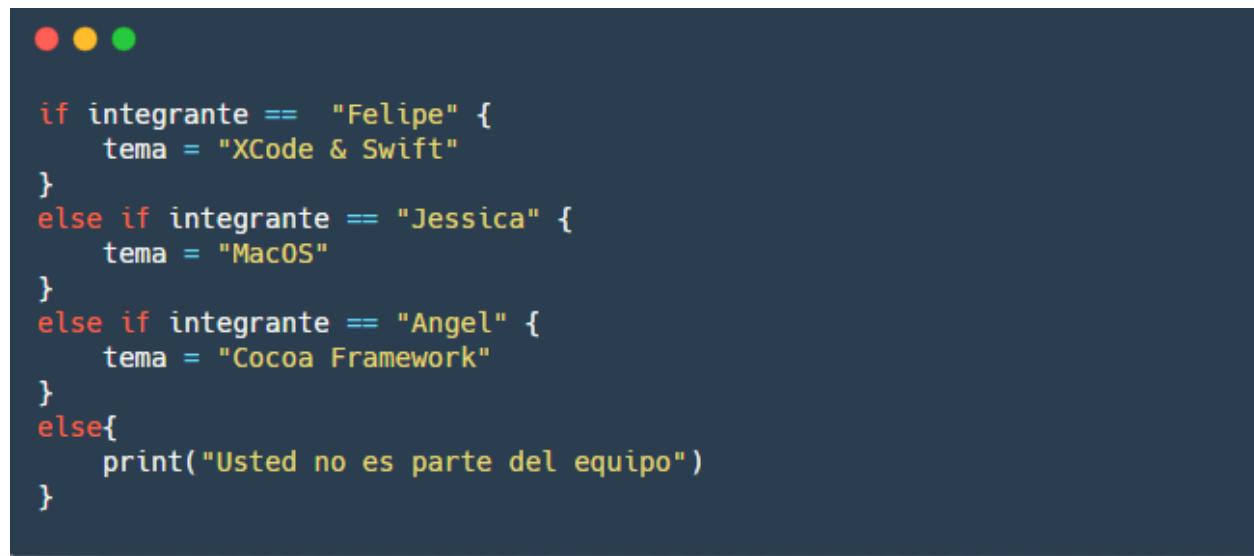
Swift incorpora como lenguaje de programación las siguientes:

If

Ejecuta un conjunto de instrucciones si una condición es verdadera, en caso de que no se cumpla dicha condición también puede agregarse la instrucción else que ejecutará un conjunto de instrucciones si la condición no se cumple.

En caso de que se desee anidar varias condicionales para posibles casos, se hace uso de la palabra reservada else if.

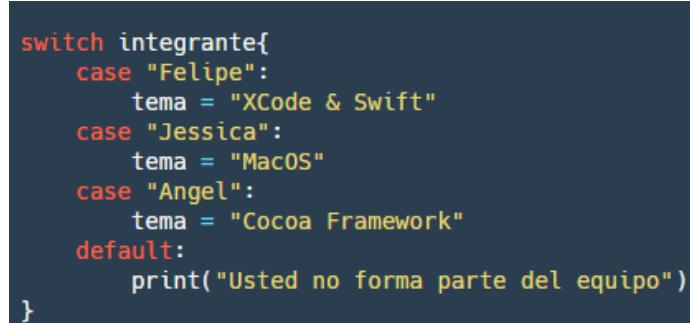
Ejemplo:



```
if integrante == "Felipe" {
    tema = "XCode & Swift"
}
else if integrante == "Jessica" {
    tema = "MacOS"
}
else if integrante == "Angel" {
    tema = "Cocoa Framework"
}
else{
    print("Usted no es parte del equipo")
}
```

Switch

Consiste en un conjunto de posibles casos los cuales se definen con la palabra reservada case seguido del posible valor. En caso de que el valor no abarque ninguno de los posibles casos se puede asignar un caso por defecto para abarcarlo.



```
switch integrante{
    case "Felipe":
        tema = "XCode & Swift"
    case "Jessica":
        tema = "MacOS"
    case "Angel":
        tema = "Cocoa Framework"
    default:
        print("Usted no forma parte del equipo")
}
```

Algo que hay que destacar es que no se puede dejar un caso sin instrucciones, pues marcaría error.

The screenshot shows a code editor interface for Swift 5.1. The code is as follows:

```
1 import Foundation
2
3
4 var tema: String = ""
5 var integrante: String = "Felipe"
6
7 switch integrante{
8     case "Felipe":
9
10    case "Jessica":
11        tema = "MacOS"
12    case "Angel":
13        tema = "Cocoa Framework"
14    default:
15        print("Usted no forma parte del equipo")
16
17
18 /tmp/CD24A77B-AE03-4CA8-98F4-9EA50C12F9B0.MN5zGv/main.swift:10:2: error: 'case' label in
| case "Felipe":
```

The editor highlights the line "case "Felipe":" in red, indicating a syntax error. The status bar at the top shows "Run" and "5.1-RELEASE".

Para solventar este caso podría hacerse uso de la palabra reservada break.

The screenshot shows the same code editor with the fix applied. The code now includes a "break" statement after the first case label:

```
3
4 var tema: String = ""
5 var integrante: String = "Felipe"
6
7 switch integrante{
8     case "Felipe":
9         break
10    case "Jessica":
11        tema = "MacOS"
12    case "Angel":
13        tema = "Cocoa Framework"
14    default:
15        print("Usted no forma parte del equipo")
16
17
18 print(tema)
```

CICLOS

For-In loop

Se utiliza para iterar sobre una secuencia de elementos dentro de un array, rangos de números, caracteres dentro de un string o incluso sobre un diccionario de datos.

Ejemplo recorriendo un array:

```
● ● ●  
let equipo = ["Felipe", "Jessica", "Angel"]  
for integrante in equipo {  
    print("Hello, \(integrante)!")  
}
```

Ejemplo con un rango de números:

```
● ● ●  
for index in 1...5 {  
    print("Número actual: \(index)")  
}
```

Ejemplo recorriendo un diccionario:

```
● ● ●  
let integrantes = ["Felipe": 21, "Jessica" : 21, "Angel": 21]  
for (nombre, edad) in integrantes {  
    print("\(nombre) tiene una edad de \(edad)")  
}
```

While loop

Se utiliza para ejecutar un conjunto de instrucciones constantemente dependiendo si una condición se cumple.

```
● ● ●  
while condicion {  
    //Instrucciones  
}
```

Repeat while

A diferencia del ciclo while, repeat while realiza primero el conjunto de instrucciones correspondientes y después pregunta si la condición se cumple.



```
repeat {
    //Instrucciones que se ejecutan al menos una vez
} while condicion
```

1.1.4 Cocoa, Cocoa Touch y Foundation



CONCEPTOS

- **FRAMEWORK**

Es una plataforma para desarrollar aplicaciones de software. Proporciona una base sobre la cual los desarrolladores de software pueden crear programas para una plataforma específica. Por ejemplo, un framework puede incluir clases y funciones predefinidas que se pueden usar para procesar entradas, administrar dispositivos de hardware e interactuar con el software del sistema. Esto agiliza el proceso de desarrollo, ya que los programadores no necesitan reinventar la rueda cada vez que desarrollan una nueva aplicación.

- **API**

Significa "Interfaz de programación de aplicaciones". Una API es un conjunto de comandos, funciones, protocolos y objetos que los programadores pueden usar para crear software o interactuar con un sistema externo. Proporciona a los desarrolladores comandos estándar para realizar operaciones comunes para que no tengan que escribir el código desde cero.

DEFINICIÓN COMPLETA

- Cocoa

Cocoa es un conjunto de frameworks orientados a objetos que proporciona un entorno de ejecución para aplicaciones que se ejecutan en OS X e iOS. Cocoa es el entorno de aplicaciones preeminente para OS X. (Carbon es un entorno alternativo en OS X, pero es un marco de compatibilidad con interfaces programáticas procedimentales destinadas a admitir bases de código OS X existentes). La mayoría de las aplicaciones que ve en OS X, incluidos Mail y Safari, son aplicaciones Cocoa. Un entorno de desarrollo integrado llamado Xcode admite el desarrollo de aplicaciones para ambas plataformas. La combinación de este entorno de desarrollo y Cocoa facilita la creación de una aplicación completa y bien factorizada.

- Cocoa Touch

Cocoa Touch es un framework de interfaz de usuario proporcionado por Apple para crear aplicaciones de software para productos como iPhone, iPad y iPod Touch. Está escrito principalmente en lenguaje Objective-C y se basa en Mac OS X. Cocoa Touch se desarrolló en base a la arquitectura Modelo-Vista-Controlador. Las interfaces de programación de aplicaciones de alto nivel disponibles en Cocoa Touch ayudan a hacer posible la animación, la creación de redes y la adición de la apariencia y el comportamiento de la plataforma nativa a las aplicaciones desarrolladas con menos desarrollo de código.

- Foundation

El framework Foundation proporciona una capa base de funcionalidad para aplicaciones y marcos, que incluyen almacenamiento y persistencia de datos, procesamiento de texto, cálculos de fecha y hora, clasificación y filtrado, y redes. Las clases, protocolos y tipos de datos definidos por Foundation se utilizan en los SDK de macOS, iOS, watchOS y tvOS.

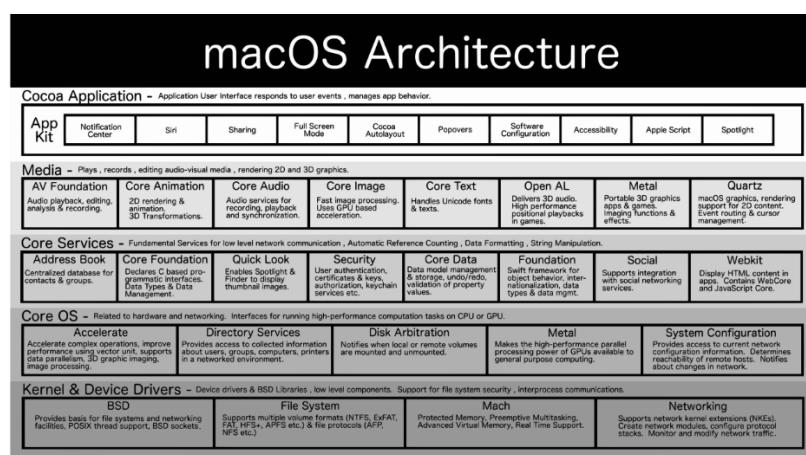
HISTORIA

Foundation fue desarrollado originalmente por la compañía NeXT como parte del sistema operativo NeXTStep. Cuando NeXT fue adquirida por Apple, Inc., las clases Foundation se convirtieron rápidamente en la base de Mac OS X y luego en el kit de desarrollo para iPhone. Debido al hecho de que Foundation comenzó como parte de NeXTstep, las clases que componen este framework comienzan con las letras "NS". También, cuando NeXT fue comprada por Apple y posteriormente se pusieron a trabajar en el sistema operativo Rhapsody que sería el sucesor directo de OpenStep. La base de OpenStep de bibliotecas y soporte binario se denominó Yellow Box.

Rhapsody evolucionó a Mac OS X y Yellow Box se convirtió en Cocoa. Por lo tanto, las clases de Cocoa comienzan con las letras NS, como NSString o NSArray. Estos representan el término propietario del framework OpenStep. Cocoa recibió este nombre debido a que no tenían el tiempo para registrar una marca nueva, por lo que se decidió usar uno de los nombres ya registrados por Apple, Cocoa, aunque este fue utilizado para un proyecto multimedia educacional para niños. Despues fue adaptado a Cocoa Touch, el cual es utilizado para desarrollar software ejecutable en iOS, watchOS y tvOS. Cocoa Touch permite el uso de hardware y funciones que no son posibles en las computadoras macOS, por esto lleva el nombre Touch.

FRAMEWORKS QUE CONTIENEN

- ARQUITECTURA



○ CORE AUDIO

Core Audio es la infraestructura de audio digital de iOS y OS X. Incluye un conjunto de marcos de software diseñados para manejar las necesidades de audio en sus aplicaciones. Core Audio está estrechamente integrado en iOS y OS X para un alto rendimiento y baja latencia.

○ CORE DATA

Core Data es un framework que utiliza para administrar los objetos de la capa del modelo en su aplicación. Proporciona soluciones generalizadas y automatizadas para tareas comunes asociadas con el ciclo de vida de los objetos y la gestión de gráficos de objetos, incluida la persistencia.

Los datos básicos suelen reducir entre un 50 y un 70 por ciento la cantidad de código que escribe para admitir la capa del modelo. Esto se debe principalmente a las siguientes funciones integradas que no se tiene que implementar, probar ni optimizar:

- Seguimiento de cambios y gestión integrada de deshacer y rehacer más allá de la edición de texto básica.
- Mantenimiento de la propagación del cambio, incluido el mantenimiento de la coherencia de las relaciones entre los objetos.
- Carga diferida de objetos, futuros parcialmente materializados (fallas) e intercambio de datos de copia en escritura para reducir los gastos generales.

○ CORE IMAGE

Core Image es una tecnología de análisis y procesamiento de imágenes diseñada para proporcionar procesamiento casi en tiempo real para imágenes fijas y de video. Opera con tipos de datos de imagen de los marcos Core Graphics, Core Video e Image I / O, utilizando una ruta de renderización de GPU o CPU. Core Image oculta los detalles del procesamiento de gráficos de bajo nivel al proporcionar una interfaz de programación de aplicaciones (API) fácil de usar. No necesita conocer los detalles de OpenGL, OpenGL ES o Metal para aprovechar la potencia de la GPU, ni tampoco necesita saber nada sobre Grand Central Dispatch (GCD) para beneficiarse del procesamiento multinúcleo. Core Image maneja los detalles.

○ CORE ANIMATION

Core Animation es una infraestructura de animación y representación de gráficos disponible en iOS y OS X que se utiliza para animar las vistas y otros elementos visuales de una aplicación. Con Core Animation, la mayor parte del trabajo necesario para dibujar cada fotograma de una animación está hecho por el programador. Todo lo que tiene que hacer es configurar algunos parámetros de animación (como los puntos de inicio y finalización) y decirle a Core Animation que comience. Core Animation hace el resto,

entregando la mayor parte del trabajo de dibujo real al hardware de gráficos integrado para acelerar el renderizado. Esta aceleración automática de gráficos da como resultado altas velocidades de fotogramas y animaciones fluidas sin sobrecargar la CPU y ralentizar su aplicación. Al escribir una aplicación para iOS se está utilizando Core Animation aun sin saberlo. En OS X es posible aprovechar Core Animation con poco esfuerzo.

○ BONJOUR

Bonjour es la propuesta de Apple para redes de configuración cero sobre IP. Bonjour surge del trabajo del Grupo de Trabajo ZEROCONF, parte del Grupo de Trabajo de Ingeniería de Internet (IETF). Los requisitos del grupo de trabajo ZEROCONF y las soluciones propuestas para redes de configuración cero sobre IP cubren básicamente tres áreas:

- Direccionamiento (asignación de direcciones IP a hosts)
- Nombrar (usar nombres para referirse a hosts en lugar de direcciones IP)
- Descubrimiento de servicios (encontrar servicios en la red automáticamente)

Bonjour tiene una solución de configuración cero para las tres áreas, como se describe en las siguientes cuatro secciones.

Bonjour permite a los proveedores de servicios, fabricantes de hardware y programadores de aplicaciones admitir un único protocolo de red, IP, al tiempo que abre nuevos caminos en la facilidad de uso.

Los usuarios de la red ya no tienen que asignar direcciones IP, asignar nombres de host o incluso escribir nombres para acceder a los servicios de la red. Los usuarios simplemente preguntan qué servicios de red están disponibles y eligen de la lista.

En muchos sentidos, este tipo de navegación es aún más potente para las aplicaciones que para los usuarios. Las aplicaciones pueden detectar automáticamente los servicios que necesitan u otras aplicaciones con las que pueden interactuar, lo que permite la conexión, la comunicación y el intercambio de datos automáticos, sin requerir la intervención del usuario.

○ CORE LOCATION

Core Location proporciona servicios que determinan la ubicación geográfica, la altitud y la orientación de un dispositivo, o su posición en relación con un dispositivo iBeacon cercano. El framework recopila datos utilizando todos los componentes disponibles en el dispositivo, incluidos Wi-Fi, GPS, Bluetooth, magnetómetro, barómetro y hardware celular.

Utiliza instancias de la clase `CLLocationManager` para configurar, iniciar y detener los servicios de ubicación principal. Un objeto de administrador de ubicación admite las siguientes actividades relacionadas con la ubicación:

- Actualizaciones de ubicación estándar y significativas. Realice un seguimiento de los cambios grandes o pequeños en la ubicación actual del usuario con un grado de precisión configurable.
- Seguimiento de la región. Monitoree distintas regiones de interés y genere eventos de ubicación cuando el usuario ingrese o salga de esas regiones.
- Alcance de baliza. Detecta y localiza balizas cercanas.
- Encabezados de la brújula. Informe los cambios de rumbo desde la brújula a bordo.

○ UIKIT

El marco UIKit proporciona la infraestructura necesaria para las aplicaciones iOS o tvOS. Proporciona la arquitectura de ventana y vista para implementar su interfaz, la infraestructura de manejo de eventos para entregar Multi-Touch y otros tipos de entrada a su aplicación, y el ciclo de ejecución principal necesario para administrar las interacciones entre el usuario, el sistema y su aplicación. Otras características ofrecidas por el marco incluyen soporte de animación, soporte de documentos, soporte de dibujo e impresión, información sobre el dispositivo actual, administración y visualización de texto, soporte de búsqueda, soporte de accesibilidad, soporte de extensión de aplicaciones y administración de recursos.

1.2 DOCUMENTAR LOS TEMAS VISTOS EN CLASE



1.2.1. RESUMEN DEL DOCUMENTAL

Este documental trata de dos personajes icónicos para el mundo de la tecnología moderna, Bill Gates y Steve Jobs. En un principio podemos ver a un joven Bill Gates aún como estudiante de Harvard interesado por la programación de una computadora, utilizando rollos de papel perforados, mientras tanto Steve Jobs regresaba de un viaje espiritual y visitó a su amigo Steve Wozniak.

La altair 8800 fue la primera computadora pequeña para ser usada por una persona, sin embargo, no existía forma fácil de hacerla funcionar. Steve Jobs vio una oportunidad aquí y decide darle un lenguaje de programación a esa máquina por medio de patrones que forman comandos hechos en hojas de papel perforadas. Se lleva dicho programa a la compañía creadora de la altair 8800 para así venderlo.

Steve Wozniak consigue crear una computadora como la altair 8800 pero a diferencia de ella le agrego un teclado, un monitor y 16 veces más memoria.

Entonces Steve Jobs vio un gran futuro para esa computadora y llegó a un acuerdo con su amigo Wozniak, él crearía los productos innovadores y Jobs tendría la empresa. Fue así como se fundó la compañía Apple y Steve Jobs introduce el diseño de la computadora personal a como la conocemos el día de hoy.

Mientras tanto, Bill Gates decide abandonar Harvard y emprende creando su compañía llamada Microsoft.

La Apple II fue la que causo el impacto necesario dentro de la empresa Apple para llevarla al éxito, pues era demasiado fácil de utilizar (Lanzada en 1997).

Bill Gates descubre la Apple 2 y se da cuenta que no es lo suficientemente compatible como para ejecutar ciertos programas, entonces junto a su equipo de trabajo le añade una placa a la Apple 2 para que así pueda tener esa compatibilidad que le faltaba.

Prácticamente Bill Gates permitió que la Apple 2 diera el salto necesario para que fuese una computadora de negocios.

Bill Gates ve una oportunidad en IBM para ofrecer un sistema operativo el cual no tenían y tuvieron que adquirirlo por medio de una compañía pequeña. Dicho sistema operativo se llamó Microsoft DOS.

Posteriormente Bill Gates negocia el sistema operativo para tener permiso de distribuirlo a terceros.

IBM dominó el mercado de las computadoras personales gracias a el sistema operativo DOS por lo que Steve Jobs decide buscar ideas en Xerox y es ahí donde descubren la

interfaz gráfica y el mouse los cuales permitirán revolucionar totalmente la manera en que se manejan las computadoras.

Steve Jobs decide mostrarle a Gates dicho descubrimiento encontrado en Xerox para que participe en el desarrollo de la interfaz de usuario, sin embargo, Bill Gates decide tomar provecho de la oportunidad y decide crear su propio sistema operativo con interfaz gráfica para así sacar del mercado a Apple, lo que provocó realmente fue que hizo caer las acciones de Apple a la mitad, pues las ventas esperadas no fueron las correspondientes por problemas de lentitud con la Macintosh. Esto provoca que el gerente de Apple sacara a Steve Jobs del equipo.

A mediados de los años 90 Bill Gates fue tachado de mantener un monopolio por parte de su compañía, para ese entonces Steve Jobs ya está de vuelta en Apple. Steve Jobs logró que Gates ayudara a Apple para que así tuviera competencia directa con Microsoft y así Gates dejase de ser tachado de monopolista, dando una situación donde ambos salían ganando. Gates fundó Bill & Melinda Gates Foundation y Jobs siguió innovando la industria de las computadoras y revolucionó la telefonía móvil en el mundo con su creación llamada iPhone. Steve Jobs falleció en 2011 por cáncer de páncreas.

1.2.2 Versiones de IOS

iPhone OS 1.0 *Alpine*

- ▶ Novedoso por integrar en un solo dispositivo de bolsillo
 - ▶ Teléfono celular
 - ▶ Cámara fotográfica
 - ▶ Reproductor de música
 - ▶ Navegador de Internet
- ▶ Incluye un conjunto muy limitado de apps
 - ▶ Mail, Calendar, Photos, Clock, Text, Safari, Notes, YouTube, Calculator, Maps, Settings, Camera, Stocks y Phone
 - ▶ No contaba con el App Store ni iTunes Store App

Programación Móvil II

Ing. Humberto Peña Valle M.T.I.

iPhone OS 2.0 *Big Bear*



- ▶ Surge el 11 de Julio de 2008
- ▶ Se incorpora el uso de "*third-party apps*"
 - ▶ Con ello surge la *App Store*
- ▶ Se integra la funcionalidad de GPS y soporte a la red de telecomunicaciones 3G
- ▶ Introduce al precursor de *iCloud*, llamado *MobileMe*

Programación Móvil II

Ing. Humberto Peña Valle M.T.I.

iPhone OS 3.0 *Kirkwood*



Programación Móvil II

- ▶ Se incorporan las herramientas de copiar/pegar
- ▶ Soporte a MMS
- ▶ Spotlight
- ▶ Conectividad hacia otros dispositivos por WiFi
- ▶ Soporte a notificaciones de inserción (push notifications), para apps de terceros

Ing. Humberto Peña Valle M.T.I.

iOS 4 *Apex*



Programación Móvil II

- ▶ Se elimina la palabra "Phone" del nombre
- ▶ Corre en iPods, iPhones y los novedosos iPads
- ▶ Introduce FaceTime
 - ▶ Primer aplicación para dispositivo móvil capaz de realizar videollamadas (solo entre usuarios de iPhone 4)
- ▶ Es la primer versión Multitarea

Ing. Humberto Peña Valle M.T.I.

iOS 5 *Telluride*



Programación Móvil II

- ▶ Se agregaron algunas novedades muy bien aceptadas
 - ▶ Centro de notificaciones
 - ▶ iMessage
 - ▶ Siri
 - ▶ iCloud
- ▶ Se mejoró la funcionalidad del bloqueo de pantalla

Ing. Humberto Peña Valle M.T.I.

iOS 6 Sundance



Programación Móvil II

Ing. Humberto Peña Valle M.T.I.

- ▶ Apple lanza su propio servicio de mapeo (*Apple Maps*)
- ▶ Surge *Passbook* para almacenar tarjetas de pago, cupones, boletos de avión, etc.
- ▶ Desaparece *YouTube* como aplicación preinstalada

iOS 7 Innsbruck



Programación Móvil II

Ing. Humberto Peña Valle M.T.I.

- ▶ Se implementó un diseño aún más simple
 - ▶ Íconos más planos
 - ▶ Tipo de letra *Helvetica*
- ▶ Efecto de ventana de vidrio en el *Centro de Control*
 - ▶ Lámpara
 - ▶ *Bluetooth*
 - ▶ *AirDrop* (novedad en esa versión)
- ▶ Renueva la aplicación *Photos*
- ▶ Surge *iTunes Radio*
- ▶ Se introduce *Touch ID*

iOS 8 Okemo



Programación Móvil II

Ing. Humberto Peña Valle M.T.I.

- ▶ Apple otorga mayor control sobre iOS a los desarrolladores
- ▶ Permite utilizar teclados y *widgets* de terceros
- ▶ Se lanza *Testflight*
- ▶ *Research Kit*
- ▶ *Health Kit*
- ▶ *Home Kit*
- ▶ Se introduce *Continuity*, para eliminar la barrera entre iOS y OS X
- ▶ *Apple Pay* en la versión iOS 8.1
- ▶ *Apple Music* en la versión iOS 8.4

iOS 9 Monarch



Programación Móvil II

Ing. Humberto Peña Valle M.T.I.

- ▶ Hacer más inteligente a Siri
 - ▶ Incorpora el uso del panel de búsqueda Spotlight
- ▶ Apple Music
- ▶ 3D Touch
- ▶ Se redujeron drásticamente los tamaños del software descargable

iOS 10 Whitetail



Programación Móvil II

Ing. Humberto Peña Valle M.T.I.

- ▶ Se mejora el bloqueo de pantalla
- ▶ Se actualiza la apariencia de las aplicaciones *News* y *Music*
- ▶ Lo más destacado es la apertura, por primera vez, del SDK para *Siri*, *iMessages* y *Maps*

iOS 11 Tigris



Programación Móvil II

Ing. Humberto Peña Valle M.T.I.

- ▶ Se combinan la pantalla de bloqueo y el centro de notificaciones, permitiendo ver las notificaciones desde la pantalla de bloqueo
- ▶ Se rediseña el centro de control
- ▶ Se introduce *Face ID*
- ▶ Siri mejora la inteligencia artificial aprendiendo sobre los intereses y el uso que cada usuario le da al dispositivo
- ▶ Siri permite el ingreso de búsquedas por texto

iOS 12 Hope



Programación Móvil II

Ing. Humberto Peña Valle M.T.I.

- ▶ La realidad aumentada (AR) mejora en cuanto a manejo de gráficos 3D y reconocimiento de elementos y rostros de la vida real, a través de la herramienta ARKit
- ▶ Mejora la aplicación Fotos y se agregan nuevas secciones y características en sus búsquedas
- ▶ Siri se integra mejor a aplicaciones de terceros para ejecución de las mismas y sugerencias
- ▶ Se introduce Siri Shortcuts, que permite crear atajos para automatizar tareas en iOS

iOS 13 Yukón



Programación Móvil II

Ing. Humberto Peña Valle M.T.I.

- ▶ Modo oscuro de forma nativa, tanto en el sistema operativo como en las aplicaciones de Apple, lo cual está disponible para el desarrollo también
- ▶ Integración de *QuickPath*, que permite deslizar el dedo por el teclado para escribir palabras
- ▶ Integración de *Memoji Stickers*, basados en la cara del usuario con diferentes expresiones y exportables para ser usados en varias apps
- ▶ Más herramientas para editar imágenes y videos
- ▶ Siri lee los mensajes al usar los AirPods



iOS 14

14

- "Widgets" hacen su primera aparición en el sistema operativo de Apple
- "Biblioteca de aplicaciones" que agrupa automáticamente las aplicaciones de manera inteligente.
- Las llamadas, FaceTime y apps de terceros aparecen en una nueva notificación compacta que no tapa toda la pantalla.
- Ahora puedes seguir viendo un video o continuar una conversación por FaceTime mientras usas otra app
- Nueva app Traducir, diseñada para traducir conversaciones en 11 idiomas.
- La app Casa te ayuda a administrar tu casa inteligente mucho mejor y, más importante aún, de manera segura.
- CarPlay revolucionó la relación entre el dispositivo iPhone y el auto. Ahora se puede abrir y encender el auto con el iPhone.

1.2.3 Guía de uso de una MAC

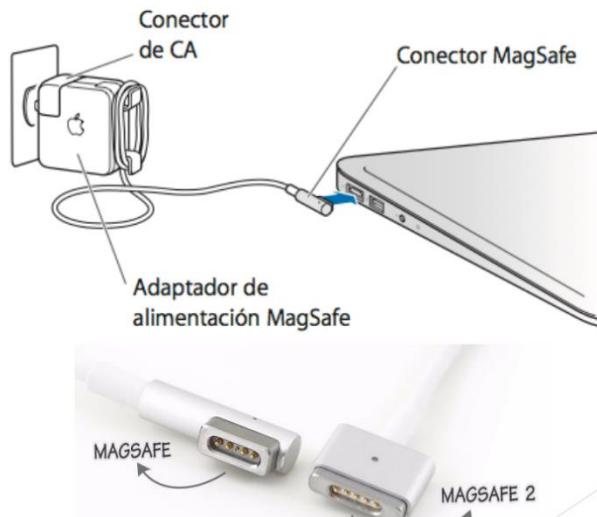


MagSafe

- ▶ **Mag** (magnéticamente) **Safe** (seguro)
- ▶ Adaptador de corriente directa
- ▶ Cuenta con un conector magnético
 - ▶ Asegura desconectar el cable de corriente si experimenta alguna tensión excesiva
 - ▶ Ayuda a evitar que se deshilachen o se debiliten los cables con el pasar del tiempo
 - ▶ Ayuda al conector a encontrar su posición correcta, para lograr una conexión segura y rápida

Ing. Humberto Peña Valle M.T.I.

MagSafe



Ing. Humberto Peña Valle M.T.I.

FireWire



Programación Móvil II

- ▶ También conocido como el estándar *IEEE 1394*
- ▶ Conectividad para dispositivos compatibles IEEE 1394
 - ▶ Tecnología en fase de obsolescencia
- ▶ Diseñado para altas tasas de transferencia
 - ▶ Otras computadoras
 - ▶ Dispositivos de almacenamiento
 - ▶ Interfaces de audio
 - ▶ Cámaras de video

Ing. Humberto Peña Valle M.T.I.

FireWire



Programación Móvil II

Ing. Humberto Peña Valle M.T.I.

FireWire



Ing. Humberto Peña Valle M.T.I.

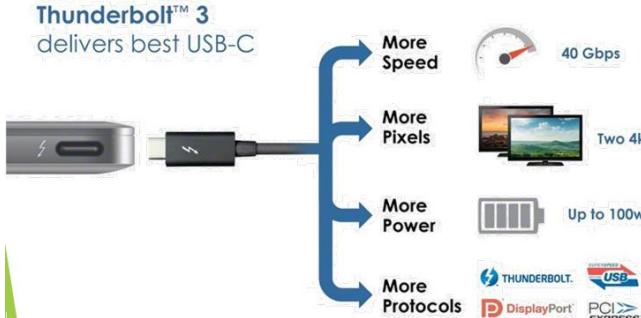
Thunderbolt



- ▶ Actualmente es el esquema de transferencia más veloz
- ▶ A diferencia de *FireWire* o USB, puede manejar conexiones de datos y video simultáneamente
- ▶ 20 veces más rápido que USB 2.0 y 12 veces más rápido que *FireWire*
- ▶ Puede conectar (directo o mediante adaptador)
 - ▶ Monitores *Thunderbolt*
 - ▶ Dispositivos *FireWire*
 - ▶ Ethernet gigabit
 - ▶ DVI, HDMI, VGA

Ing. Humberto Peña Valle M.T.I.

Thunderbolt



Ing. Humberto Peña Valle M.T.I.

USB-C

- ▶ Se utiliza para
 - ▶ Carga eléctrica
 - ▶ Transferencia de datos
 - ▶ Transferencia de video
- ▶ Puede conectar
 - ▶ iPhones, iPads, iPods para carga y sincronización
 - ▶ Cámaras y dispositivos de almacenamiento para transferir datos

Ing. Humberto Peña Valle M.T.I.

USB-C



Img. Humberto Peña Valle M.T.I.

Lightning

- ▶ Conector propietario de *Apple* para *iPad*, *iPad mini*, *iPhone*, *iPod touch*
 - ▶ Carga eléctrica
 - ▶ Transferencia de datos
- ▶ El otro extremo del cable es USB



Img. Humberto Peña Valle M.T.I.

Primeros pasos como usuario Mac Tour (asistente de configuración)

- ▶ Idioma
 - ▶ 30 idiomas diferentes
 - ▶ Al seleccionar un idioma, la Mac habla en dicho idioma
- ▶ País
- ▶ Teclado
- ▶ Conexión WiFi
- ▶ Transferencia de información de otro equipo
 - ▶ Cuentas de usuario
 - ▶ Archivos
- ▶ Zona horaria
- ▶ Contraseña

ing. Humberto Peña Valle M.T.I.

Apple ID

- ▶ Cuenta para acceder a los servicios de *Apple*
 - ▶ *FaceTime*
 - ▶ *App Store*
 - ▶ *Apple Music*
 - ▶ *iMessage*
 - ▶ *iBooks*
 - ▶ *iCloud*
 - ▶ *iTunes*

ing. Humberto Peña Valle M.T.I.

Apple ID

- ▶ Se solicita asociar a cada cuenta en el primer inicio de sesión
- ▶ Se puede crear, utilizar uno existente o saltar este paso



ing. Humberto Peña Valle M.T.I.

Repositorio de contraseñas

- ▶ Posterior al inicio de sesión con Apple ID, se ofrece el servicio *iCloud Keychain*
- ▶ Almacena las contraseñas en iCloud para poder utilizarlas en todos los equipos asociados a un Apple ID
- ▶ Se configura con una contraseña de 4 dígitos y el número de teléfono para aumentar la seguridad mediante el envío de SMS

ing. Humberto Peña Valle M.T.I.

Familiarizarse con el nuevo sistema operativo



Ing. Humberto Peña Valle M.T.I.

Dock

- ▶ Todo ícono es un acceso directo, excepto *Finder* y *Trash Can*
- ▶ Puede ajustar la posición de cada ícono, excepto *Finder* y *Trash Can* que se encuentran siempre en los extremos
- ▶ Las aplicaciones se ubican del lado izquierdo del divisor

Ing. Humberto Peña Valle M.T.I.

Dock

- ▶ Carpetas, documentos y ventanas minimizadas se encuentran del lado derecho
- ▶ El *Dock* puede reubicarse en los lados de la pantalla, así como puede ocultarse o escalarse
- ▶ Muestra un indicador debajo de las aplicaciones abiertas

Ing. Humberto Peña Valle M.T.I.

Menu Bar

- ▶ Ubicada siempre en la parte alta de la pantalla
- ▶ Se ajusta basado en lo que se encuentre utilizando
 - ▶ El nombre de la aplicación es la primera opción del menú
- ▶ En la extrema derecha de la barra se encuentran accesos a algunas herramientas o utilerías
- ▶ La mayoría de las funciones de la aplicación en uso, deben encontrarse en alguna opción del menú
- ▶ Una de las herramientas de mayor utilidad se representa con un ícono de lupa, cuyo nombre es *spotlight*

Ing. Humberto Peña Valle M.T.I.

Spotlight

- ▶ Se ubica en la esquina superior derecha de la pantalla
- ▶ Puede encontrar aplicaciones y documentos en la computadora, aún si la palabra escrita se encuentra solo dentro del documento
- ▶ Puede abrir las aplicaciones y documentos encontrados
- ▶ Puede llevar a cabo operaciones aritméticas

Ing. Humberto Peña Valle M.T.I.

Finder

- ▶ Es el lugar donde se encuentra todo lo que contiene la computadora
- ▶ Se utiliza para acceder y administrar carpetas, archivos, volúmenes y aplicaciones
- ▶ La barra lateral permite navegar en las áreas de la computadora
- ▶ La barra de herramientas tiene 4 botones para ajustar la vista de carpetas y archivos, así como un buscador
- ▶ *Finder* siempre se encuentra abierto y en ejecución
- ▶ No puede cerrarse como el resto de las aplicaciones

Ing. Humberto Peña Valle M.T.I.

Mission Control

- ▶ Ayuda a visualizar rápidamente
 - ▶ Todas las ventanas que están abiertas
 - ▶ Los programas a pantalla completa que se están ejecutando
 - ▶ Los escritorios abiertos
- ▶ Los escritorios son espacios de trabajo que el usuario diferencia lógicamente
- ▶ Cada escritorio puede contener un conjunto de ventanas y configuración del entorno diferente
- ▶ La carpeta *Desktop* o *Escritorio* de la computadora es única, independientemente de los escritorios creados

Ing. Humberto Peña Valle M.T.I.

Mac App Store

- ▶ Es la tienda de aplicaciones de *Apple* para *OS X* y *macOS*
- ▶ Se encuentran miles de Apps, tanto gratuitas como de pago

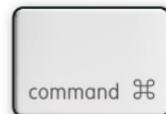
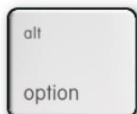
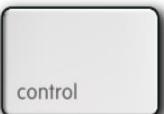
Ing. Humberto Peña Valle M.T.I.

Teclado



ing. Humberto Peña Valle M.T.I.

Teclas especiales



- ▶ **Control ^** (ctrl) se usa mucho menos que en una PC. Con los ratones de un solo botón, sirve para hacer click derecho (**control+click**)
- ▶ **Alt ⌘** (option) muy utilizada en Mac. Ofrece opciones diferentes, incluso en los menús de un programa
- ▶ **Comando ⌘** (command, cmd) es equivalente a las cosas que en una PC se hacen con la tecla **Control**

ing. Humberto Peña Valle M.T.I.

Combinaciones más habituales

► Command +

- ▶ **C** → Copiar
- ▶ **X** → Cortar
- ▶ **V** → Pegar
- ▶ **Z** → Deshacer
- ▶ **P** → Imprimir
- ▶ **H** → Ocultar ventana
- ▶ **Q** → Salir de un programa
- ▶ **Tab** → Cambiar de aplicación (similar a Windows)
- ▶ **Delete** → borra archivo o carpeta

Ing. Humberto Peña Valle M.T.I.

Combinaciones más habituales

► Command +

- ▶ **Barra espaciadora** → Abre spotlight
- ▶ **B** → Poner texto en negrita
- ▶ **I** → Poner texto en cursiva (Itálica)
- ▶ **U** → Subrayar texto
- ▶ **Flecha izq.** → Ir al inicio de una línea
- ▶ **Flecha der.** → Ir al fin de una línea

► Shift + Command +

- ▶ **3** → Captura de pantalla completa
- ▶ **4** → Captura de una ventana o área seleccionada

Ing. Humberto Peña Valle M.T.I.

CONCLUSIONES

Jessica Janet Grajeda Castellanos

Yo creo que haber estudiado la historia de Apple me da una visión más amplia de lo importante que esta compañía ha sido para la tecnología en la modernidad, ya que sin ella tal vez no conoceríamos a la computadora personal como la conocemos ahora. Además de los conceptos que aportó a la tecnología moderna, también presentó un lenguaje de programación el cual me parece muy interesante y quiero aprender más de este.

Juan Felipe Garza Sánchez

El conocer desde los orígenes de las distintas herramientas que nos provee Apple para el desarrollo de aplicaciones iOS como lo son el IDE oficial de desarrollo XCode, complementos para facilitar el aprendizaje durante el desarrollo en tiempo real, el lenguaje de programación oficial para el desarrollo de aplicaciones iOS, hasta el funcionamiento concreto de sus componentes como lo es la sintaxis del lenguaje de programación Swift, librerías que involucra, frameworks, tipos de archivos involucrados en el desarrollo, estructura de aplicaciones y procedimiento para la publicación de aplicaciones dentro de la App Store. Todo ello me ha permitido darme una idea de la forma de trabajar dentro de este entorno y así tener un punto de partida para mi aprendizaje dentro de la materia.

Sin duda alguna la investigación será de gran utilidad para las siguientes unidades, pues, ya no será mi primera experiencia averiguando el entorno de desarrollo, así como el despertar una curiosidad por mi parte para el profundizar en aquellos temas que probablemente no se lleguen a ver en la materia.

Miguel Ángel Méndez Cruz

Mi conclusión es que el sistema operativo de Apple me pareció muy interesante ya que en mi caso, nunca había trabajado con él o buscado especificaciones técnicas del mismo, pero me parece muy amigable para el usuario, tiene algunas coincidencias con el de windows pero aun así la simplicidad con la que se maneja y con la que trabaja lo hace un sistema operativo muy atractivo para todo tipo de usuarios, de igual manera en su forma móvil del iPhone (ios) tiene muchas herramientas que no yo sabía que tenían tanto sociales como del entorno

BIBLIOGRAFÍA | LINKOGRAFÍA

Apple Inc. (2018, 9 abril). About Files and Directories. Copyright 2018 Apple Inc. All Rights Reserved.

https://developer.apple.com/library/archive/documentation/FileManagement/Conceptual/FileSystemProgrammingGuide/Introduction/Introduction.html#/apple_ref/doc/uid/TP40010672-CH1-SW1

Sundell, J. (2019, 7 abril). String literals in Swift. [wiftbysundel.](http://www.swiftbysundell.com/articles/string-literals-in-swift/)

<https://www.swiftbysundell.com/articles/string-literals-in-swift/>

Apple Inc. (s. f.). Apple Developer Documentation. <https://apple.com/>. Recuperado 18 de febrero de 2021, de <https://developer.apple.com/documentation/xcode/>

Beginning Xcode Swift Edition Matthew Knott - 2007. Diving Deeper.

Dive into design patterns Alexander Shvets - Enero 16, 2019. Structural Design Patterns.

Patrones de Diseño de Software Miguel Ángel Sánchez - Nov 22, 2017

Alvarado, P. (2014, July 20). Qué Es Xcode, Para Qué Sirve y Cómo Descargar. IPadizate; iPadizate. <https://www.ipadizate.es/2014/07/20/xcode-93212/>

API (Application Program Interface) Definition. (n.d.). The Tech Terms Computer Dictionary. Retrieved February 19, 2021, from <https://techterms.com/definition/api>

Apple Developer Documentation. (n.d.-a). Apple Developer. Retrieved February 19, 2021, from <https://developer.apple.com/documentation/uikit>

Apple Developer Documentation. (n.d.-b). Apple Developer. Retrieved February 19, 2021, from <https://developer.apple.com/documentation/foundation>

Bonjour Concepts. (2013, April 23). Apple Developer.

https://developer.apple.com/library/archive/documentation/Cocoa/Conceptual/NetServices/Articles/about.html#/apple_ref/doc/uid/TP40002458-SW1

Cocoa (Touch). (2018, April 6). Apple Developer.

<https://developer.apple.com/library/archive/documentation/General/Conceptual/DevPedia-CocoaCore/Cocoa.html>

Dates in Mac history | KLJCharters Blog. (2013, October 30). KLJCharters Blog; <https://www.facebook.com/WordPresscom>. <https://kljcharters.wordpress.com/mac-dates/>

Documentation Archive. (n.d.). Apple Developer. Retrieved February 19, 2021, from <https://developer.apple.com/library/archive/navigation/>

Dormehl, L. (2020a, November 16). Today in Apple history: Steve Jobs secures Macintosh name | Cult of Mac. Cult of Mac; <https://www.facebook.com/cultofmac/>.
<https://www.cultofmac.com/454244/steve-jobs-naming-macintosh/>

Framework Definition. (n.d.). The Tech Terms Computer Dictionary. Retrieved February 12, 2021, from <https://techterms.com/definition/framework>

What Is Cocoa? (2013, September 18). Apple Developer.
<https://developer.apple.com/library/archive/documentation/Cocoa/Conceptual/CocoaFundamentals/WhatIsCocoa/WhatIsCocoa.html>

iOS 14 - Apple (MX). (n.d.). Apple (México). Retrieved February 13, 2021, from <https://www.apple.com/mx/ios/ios-14/>

Xcode - Soporte - Apple Developer. (n.d.). Apple Developer. Retrieved February 14, 2021, from <https://developer.apple.com/es/support/xcode/>

Xcode Release Notes. (2018, June 13). Apple Developer.
https://developer.apple.com/library/archive/releasenotes/DeveloperTools/RN-Xcode/Chapters/Introduction.html#/apple_ref/doc/uid/TP40001051-CH1-SW326

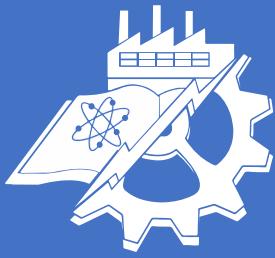
20 Years of Mac OS X Design History - 59 Images - Version Museum. (n.d.). Version Museum. Retrieved February 12, 2021, from <https://www.versionmuseum.com/history-of/mac-os-x>

Macworld Mac SECRETS : Pogue, David, 1963- : Free Download, Borrow, and Streaming : Internet Archive. (1999). Internet Archive; Foster City, CA : IDG Books Worldwide.
https://archive.org/details/mac_Macworld_Mac_Secrets_5th_Edition_1999/page/n329/mode/2up

System 1.0 / Finder 1.0 (Macintosh System Software). (n.d.). Wayback Machine. Retrieved February 12, 2021, from <https://web.archive.org/web/20161202033543/http://www.mac512.com/macwebpages/system10.htm>

uMac | University of Utah | Mac OS History. (n.d.). UMac | University of Utah | University of Utah - Mac Managers. Retrieved February 12, 2021, from https://www.macos.utah.edu/documentation/short_courses/mac_os_x_overview/history_and_evolution/mac_os_history.html

uMac | University of Utah | Mac OS X History and Technology. (n.d.). UMac | University of Utah | University of Utah - Mac Managers. Retrieved February 12, 2021, from https://www.macos.utah.edu/documentation/short_courses/mac_os_x_technology.html



INSTITUTO TECNOLÓGICO DE NUEVO LAREDO

CON LA CIENCIA POR LA HUMANIDAD

INGENIERÍA EN SISTEMAS COMPUTACIONALES



Programación móvil 2

Tema 2



DOCENTE

Ing. Humberto Peña Valle M.T.I.



INTEGRANTES

Juan Felipe Garza Sánchez	17100218
Jessica Janet Grajeda Castellanos	17100229
Miguel Ángel Méndez Cruz	17100254

ÍNDICE

Explicación general del contenido del tema	105
2.1 Elementos de la arquitectura Swift	106
2.2 Explicación, estructura y sintaxis en Swift	112
2.3 Prácticas	132
Conclusiones	278
Linkografía	279

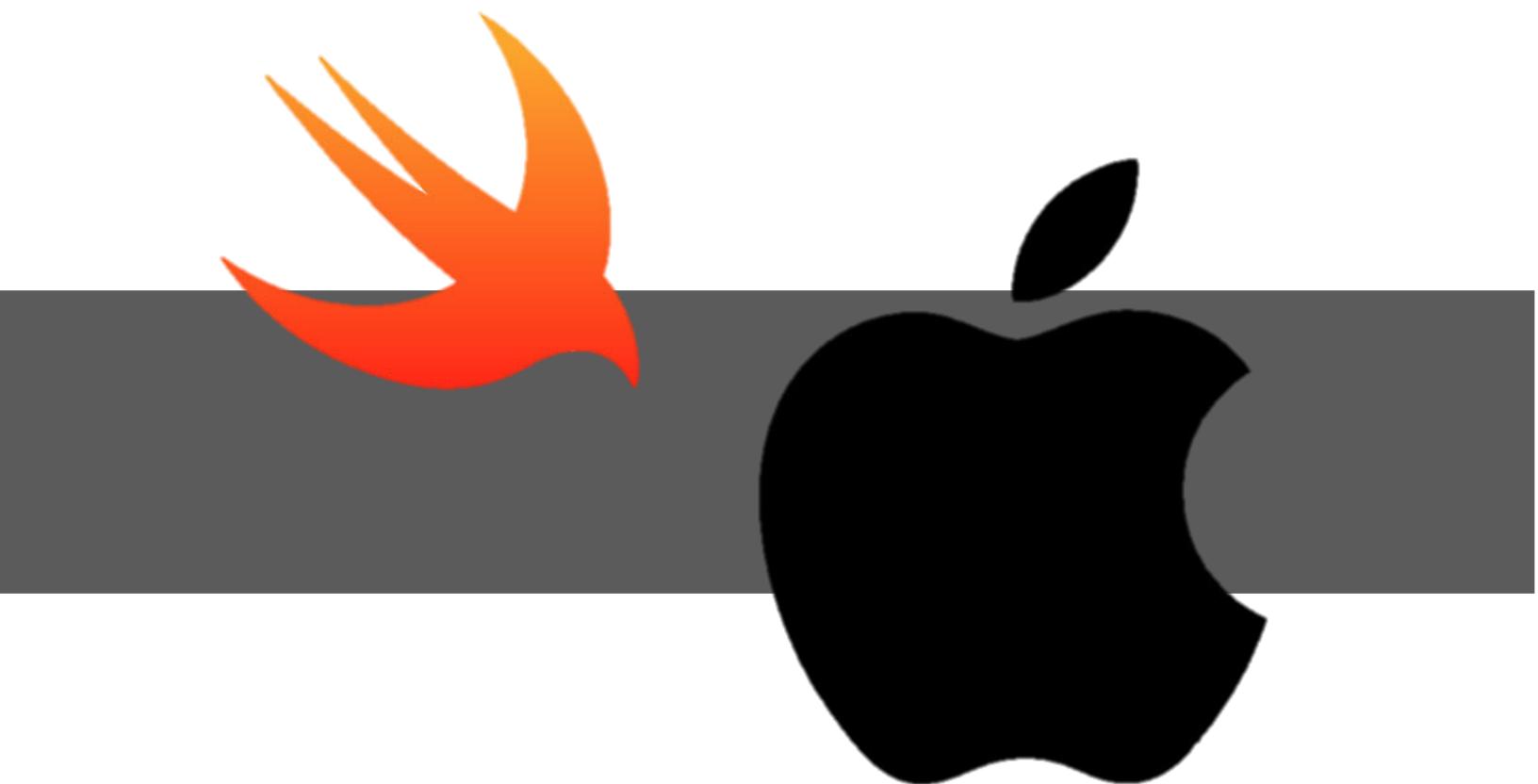
EXPLICACIÓN GENERAL DEL CONTENIDO DEL TEMA

En este tema lo principal que veremos es el lenguaje “Swift” el cual es un lenguaje de programación de propósito general el cual está desarrollado por Apple para sus dispositivos con los sistemas operativos iOS, OS X, watchOS, etc. Este lenguaje es rápido además de eficiente por lo que permite su rápido aprendizaje. En el tema anterior pudimos ver los básicos de este lenguaje de programación, sin embargo, ahora además de investigar acerca de tópicos un poco más avanzados, realizaremos diversas prácticas para reforzar nuestras habilidades con Swift.



Swift

2.1 INVESTIGACIÓN ARQUITECTURA SWIFT



2.1.1. Estructura de un archivo (código fuente)

La gran característica que mejor define Swift es que es un lenguaje que extrae lo mejor de los lenguajes modernos, que está construido sobre sí mismo, es muy sencillo de “personalizar” a nuestro gusto. Al ser Swift un lenguaje compilado, no interpretado, su compilación es uno de sus puntos fuertes, que le da la potencia en su funcionamiento. Todo archivo Swift termina con .swift, por ejemplo Clase.swift

A continuación, se muestra un archivo Swift de ejemplo:



```
1: import UIKit
2:
3: class myClass: myParent, myProtocol {
4:
5:     var myString: String = ""
6:     var myOtherString: String?
7:     var yetAnotherVariable: Float!
8:     let myAge: Int = 29
9:
10:    @IBOutlet weak var userOutput: UILabel!
11:    @IBOutlet var anotherUserOutput: UILabel!
12:
13:    class func myTypeMethod(aString: String) -> String {
14:        // Implement a Type method here
15:    }
16:
17:    func myInstanceMethod(myString: String, myURL: NSURL) -> NSDate? {
18:        // Implement the Instance method here
19:    }
20:
21:    override func myOverriddenInstanceMethod() {
22:        // Override the parent's method here
23:    }
24:
25:    @IBAction func myActionMethod(sender: AnyObject) {
26:        // React to an action in the user interface
27:    }
28:
29: }
```

Declaración import

Se utiliza esta declaración para incluir cualquier otro archivo que la aplicación necesitará utilizar.

Siguientes declaraciones

En este archivo de ejemplo se declaran clases, métodos, así como variables, no todos los archivos Swift pueden contenerlos, pero sí su mayoría. A continuación, se presentan algunas características que se pueden utilizar en archivos Swift que son diferentes a otros lenguajes

Constantes y variables

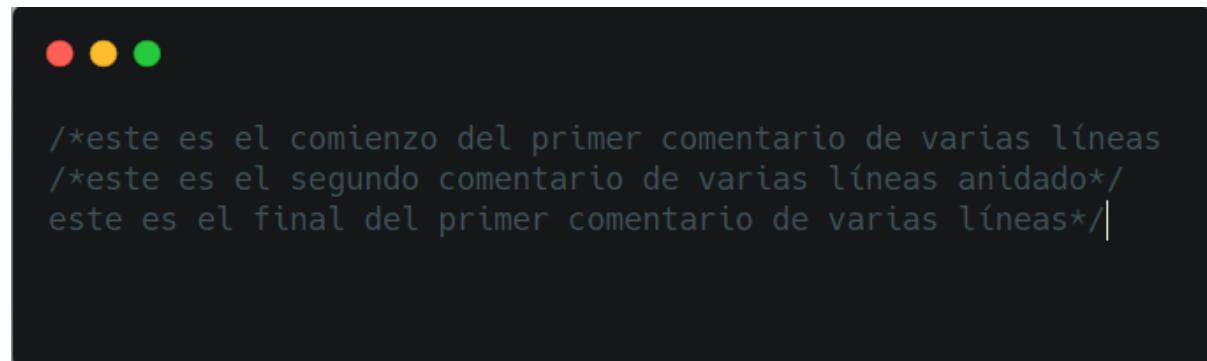
Es posible usar casi cualquier carácter para nombrar variables y constantes:



```
let π = 3.14159
let 世話 = "Cuidado"
let 鳩 = "America"
```

Comentarios multilínea anidados

Al contrario que en C, los comentarios multilínea en Swift pueden anidarse dentro de otros comentarios.



```
/*este es el comienzo del primer comentario de varias líneas
/*este es el segundo comentario de varias líneas anidado*/
este es el final del primer comentario de varias líneas*/
```

Punto y coma

El punto y coma al final de sentencia sigue existiendo y si lo añadimos el compilador no lanzará un error. El mismo compilador es el que se encarga de añadirlo si no está. Así que su uso es opcional y puede sernos útil cuando queremos juntar dos sentencias en una misma línea:



```
let America = "鳩" ; println(America)
//prints "鳩"
```

Operadores básicos

El operador del módulo % puede operar con números en coma flotante.
8 % 2.5 // Devuelve 0.5. También se encuentran operadores de rango a..b y a...b, útiles a la hora

de expresar secuencias. Y al contrario que en C y muchos otros lenguajes la asignación no devuelve valor alguno.

2.1.2. Estructura de un archivo IPA

IPA es el formato de archivo utilizado para las aplicaciones de iOS que se ejecutan en iPhones, iPads y dispositivos iPod Touch. El nombre significa "paquete de aplicación iOS". Los archivos IPA son equivalentes a los paquetes de aplicaciones ".app" que usa macOS. Usan el formato Zip común, a pesar de su extensión de archivo inusual.

Un archivo IPA válido siempre debe contener una carpeta de Payload nivel superior y, si está diseñado para su distribución a través de App Store, también tendrá algunos archivos de datos de iTunesMetadata .

Ejemplo de la API para iBooks 3.2:

```
$ unzip -l iBooks\ 3.2.ipa
Archive: iBooks 3.2.ipa
      Length      Date    Time     Name
-----  -----
          0  06-18-2014 10:43  Payload/iBooks.app/
  22732944  11-13-2013 13:11  Payload/iBooks.app/iBooks
        3978  11-07-2013 17:43  Payload/iBooks.app/Info.plist
          0  06-18-2014 10:42  Payload/iBooks.app/_CodeSignature/
  70449   11-13-2013 13:11  Payload/iBooks.app/_CodeSignature/CodeResources
          8  11-07-2013 17:43  Payload/iBooks.app/PkgInfo
          0  06-18-2014 10:43  Payload/iBooks.app/SC_Info/
          0  06-18-2014 10:42  Payload/iBooks.app/Settings.bundle/
  25226   11-07-2013 17:44  iTunesArtwork
  1883   06-18-2014 10:43  iTunesMetadata.plist
```

En el ejemplo anterior, el archivo Payload/iBooks.app/iBooks es el binario de la aplicación. iTunesArtwork es el ícono de la aplicación, como se muestra en la App Store. E iTunesMetadata.plist es un diccionario de información con una colección de metadatos sobre la aplicación, con su nombre, autor, fecha de lanzamiento, etc.

Los archivos IPA contienen archivos binarios que son específicos de iOS. Se pueden crear para dispositivos iOS de 32 bits (es decir, iPhone 5c y versiones posteriores), o para los dispositivos de 64 bits más recientes lanzados desde entonces. Los dispositivos iOS usan la arquitectura ARM, mientras que su PC con Windows o Mac probablemente usa x86. Esto significa que sin un binario x86 incorporado específico, su computadora no podrá ejecutar una aplicación iOS.

2.1.3. Espacios de nombres

Para evitar colisiones de nombres con otras bibliotecas y marcos, una clase Objective-C debe tener un nombre único. Esa es la razón por la que Apple usa prefijos en las clases Objective-C, como `UIView`, `CGRect` y `CALayer`. Los módulos Swift hacen que la necesidad de prefijos de clase sea obsoleta. Aunque los módulos son un importante paso adelante, no son tan flexibles. Swift actualmente no ofrece una solución para los tipos de espacios de nombres y las constantes dentro de los módulos. En Objective-C, cada constante tiene como prefijo dos o tres letras para evitar colisiones de nombres y el nombre de la constante describe su uso en el proyecto.

Los módulos Swift hacen que la necesidad de prefijos de clase sea obsoleta. En Objective-C, es una buena práctica utilizar un prefijo de clase para evitar colisiones de nombres con otras bibliotecas y marcos, incluidos los de Apple.

Aunque los módulos representan un importante paso adelante, no son tan flexibles como muchos de nosotros quisiéramos que fueran. Swift actualmente no ofrece una solución para los tipos de espacios de nombres y las constantes dentro de los módulos.

2.1.4. Módulos

Swift organiza el código en módulos. Cada módulo especifica un espacio de nombres e impone controles de acceso sobre qué partes de ese código se pueden usar fuera del mismo. Un programa puede tener todo su código en un solo módulo, o puede importar otros módulos como dependencias.

Además de los muchos de los módulos proporcionados por el sistema, como Darwin en macOS o Glibc en Linux, la mayoría de las dependencias requieren que el código se descargue y construya para poder ser utilizado.

Cuando hacemos uso del código de un módulo para resolver un problema este no está limitado a un caso en particular, por el contrario, este módulo o librería externa se puede reutilizar en otras situaciones. Por ejemplo, un módulo que nos ayuda con las peticiones de red se puede compartir entre una aplicación para compartir fotos y una aplicación de clima.

El uso de módulos nos ayuda a agilizar los tiempos de desarrollo, permitiendo construir nuestro proyecto (o ciertas áreas de este) sobre el código bien escrito (en la mayoría de los casos) de otros desarrolladores en lugar de volver a implementar la misma funcionalidad nosotros mismos y perder un tiempo para al final, muchas veces, ni siquiera obtener un mejor desempeño.

2.1.5. Instancias

Los métodos de instancia son funciones que pertenecen a instancias de una clase, estructura o enumeración en particular. Admiten la funcionalidad de esas instancias, ya sea proporcionando formas de acceder y modificar las propiedades de la instancia, o proporcionando funcionalidad relacionada con el propósito de la instancia. Los métodos de instancia tienen exactamente la misma sintaxis que las funciones, como se describe en Funciones. Un método de instancia tiene acceso implícito a todos los demás métodos de instancia y propiedades de ese tipo. Un método de instancia se puede llamar solo en una instancia específica del tipo al que pertenece. No se puede llamar de forma aislada sin una instancia existente.

2.2 EXPLICACIÓN, ESTRUCTURA Y SINTAXIS EN SWIFT

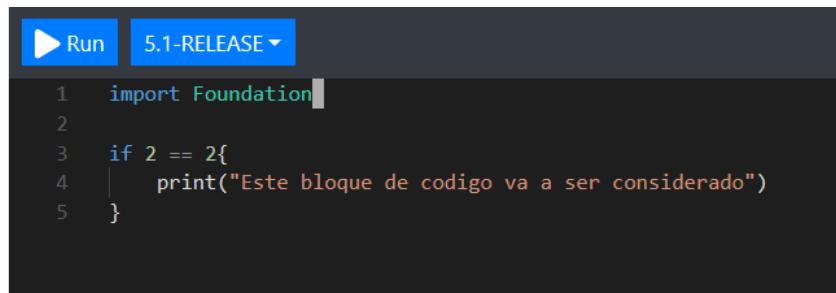


2.2.1. Condicionales

SENTENCIA IF

Una condición nos va a permitir ejecutar cierto bloque de código bajo uno o varios criterios a considerar.

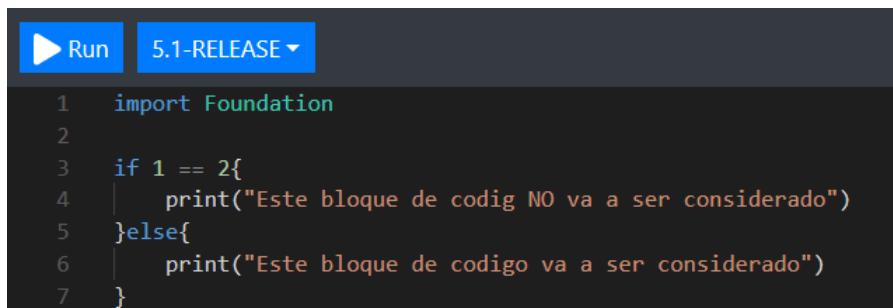
La sintaxis para realizar una condicional básica en Swift sería la siguiente:



A screenshot of the Xcode IDE showing a Swift script. The code is:

```
Run 5.1-RELEASE ▾
1 import Foundation
2
3 if 2 == 2{
4     print("Este bloque de codigo va a ser considerado")
5 }
```

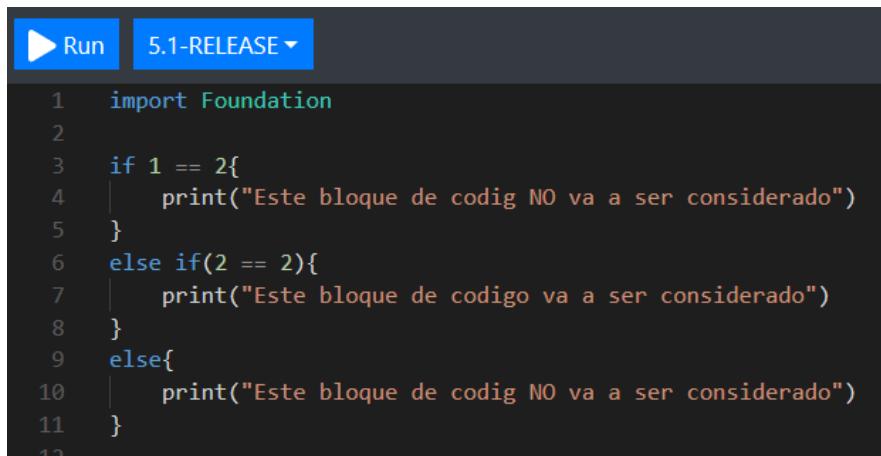
Para definir un bloque a ejecutar en caso de que la primera condicional no sea ejecutada, sería la siguiente:



A screenshot of the Xcode IDE showing a Swift script. The code is:

```
Run 5.1-RELEASE ▾
1 import Foundation
2
3 if 1 == 2{
4     print("Este bloque de codig NO va a ser considerado")
5 }else{
6     print("Este bloque de codigo va a ser considerado")
7 }
```

Para definir varios if anidados, sería por medio de la palabra “else if” tal que así:



A screenshot of the Xcode IDE showing a Swift script. The code is:

```
Run 5.1-RELEASE ▾
1 import Foundation
2
3 if 1 == 2{
4     print("Este bloque de codig NO va a ser considerado")
5 }
6 else if(2 == 2){
7     print("Este bloque de codigo va a ser considerado")
8 }
9 else{
10    print("Este bloque de codig NO va a ser considerado")
11 }
12 }
```

SENTENCIA SWITCH

Por medio de esta sentencia se evalúa una expresión para después comparar el resultado con un conjunto de casos.

La sintaxis para la definición de un switch básico sería la siguiente:



```
import Foundation

var edad:Int = 20

switch(edad){
    case 18:
        print("Tienes 18 años de edad")
    case 19:
        print("Tienes 19 años de edad")
    default:
        print("Tu edad no se encuentra en los casos")
}
```

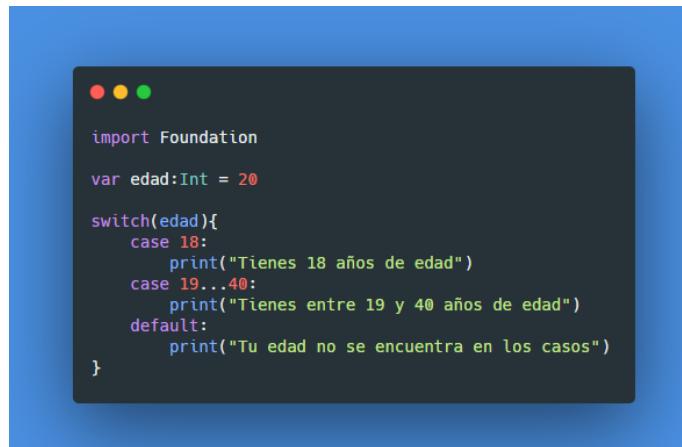
Dentro de esta estructura podemos apreciar la palabra `switch`, en la cual es donde vamos a colocar la expresión a evaluar dentro de los paréntesis, seguido de dos llaves para empezar a definir cada uno de los posibles casos, cada caso debe definirse por la palabra reservada “`case`” un espacio y el posible valor de la expresión capturada con anterior, posterior a ello dos puntos y el conjunto de instrucciones a considerar.

En caso de que un caso no requiera tener un código implementado simplemente se puede hacer uso de la palabra “`break`” para evitar un error de sintaxis.

Para definir un caso por defecto es necesario definirlo con la palabra reservada “`default`” seguido de dos puntos y el código a ejecutar.

Una parte interesante de este lenguaje de programación es que en los casos a definir se pueden hacer comparaciones por medio de rango, o varios valores para abarcar más posibilidades dentro de un mismo caso.

Ejemplo con rangos:



```
import Foundation

var edad:Int = 20

switch(edad){
    case 18:
        print("Tienes 18 años de edad")
    case 19...40:
        print("Tienes entre 19 y 40 años de edad")
    default:
        print("Tu edad no se encuentra en los casos")
}
```

Ejemplo con varios valores:



```
import Foundation

var edad:Int = 20

switch(edad){
    case 17, 18:
        print("Tienes 17 o 18 años de edad")
    case 20:
        print("Tienes 20 años de edad")
    default:
        print("Tu edad no se encuentra en los casos")
}
```

Cuando hacemos uso de la sentencia switch dentro de Swift es interesante saber que tenemos una forma de hacer comparaciones más complejas dentro de nuestros casos por medio de la palabra reservada “where”:



```
import Foundation

var edad:Int = 20

switch(edad){
    case let x where x >= 18:
        print("Eres mayor de edad")
    default:
        print("No eres mayor de edad")
}
```

Otra funcionalidad interesante de esta sentencia es la palabra reservada “fallthrough” la cual nos permite recorrer varios casos a la vez, de tal manera que si se cumple cierto caso y al llegar a la instrucción fallthrough, se ejecutara también el siguiente caso sin importar la evaluación de su valor respecto a la expresión dada en el switch.

```
Run 5.1-RELEASE ▾
1 import Foundation
2
3 var numero = 1
4
5 switch(numero){
6     case 1:
7         print("Se ejecuta el primer caso")
8         fallthrough
9     case 3...:
10        print("Se ejecuta el segundo caso")
11        fallthrough
12    default:
13        print("Se ejecuta el tercer caso (default)")
```

Se ejecuta el primer caso
Se ejecuta el segundo caso
Se ejecuta el tercer caso (default)

SENTENCIA IF-INLINE O CONDICIONAL TERNARIA

Esta es una manera de obtener un valor en particular dependiendo de la expresión lógica que se le defina, la sintaxis sería la siguiente:

```
Run 5.1-RELEASE ▾
1 import Foundation
2
3 var genero = "Mujer"
4
5 var esUnaMujer = genero == "Mujer"
6
7 var resultado = (esUnaMujer ? "Es una mujer" : "Es un hombre")
```

1 2 3

Siendo el punto número uno la expresión a evaluar, el punto dos el valor a retornar en caso de que la expresión sea verdadera y el punto número tres el valor a retornar en caso de que la expresión dada sea falsa.

2.2.2. Ciclos

CICLO FOR IN

Este ciclo es ideal para realizar iteraciones sobre una cantidad específica de elementos cuando el número de iteraciones es conocido y/o fácil de adivinar

La sintaxis es la siguiente:

```
for <constante> in <fuente de datos> {  
    // Código a ejecutar  
}
```

Este bucle hará las iteraciones de acuerdo al número de elementos que existan en la fuente de datos, mientras que en la constante será igual al valor correspondiente al número de iteración y su posición en la fuente de datos. Ejemplo:

```
let nombres = ["Pepe", "Pedro", "Paco", "Simon"]  
for nombre in nombres {  
    print("Hola, \(nombre)!")  
}
```

También podemos utilizar la palabra reservada **where**, la cual es una cláusula que verifica condiciones adicionales, ejemplo:

```
for numero in 1...10 where numero.isMultiple(of: 2) {  
    print("Número: \(numero)")  
}
```

CICLO WHILE

Este ejecuta el código encontrado entre sus llaves mientras que la condición no devuelva false. Este tipo de ciclos se utilizan cuando desconocemos el número de iteraciones y las ejecuciones dependen de una expresión lógica.

Existen dos tipos de ciclos while:

- While

Este evalúa la condición al comienzo de cada iteración

La sintaxis es la siguiente:

```
while <condicion> {  
    //Código a ejecutar  
}
```

Se repetirá siempre y cuando la condición sea true hasta que se convierta en false, ejemplo:

```
● ● ●

var nivelActual:Int = 0, nivelFinal:Int = 5
let juegoCompletado = true
while (nivelActual <= nivelFinal) {
    //jugar
    if juegoCompletado {
        print("Acabas de completar el nivel \(nivelActual)")
        nivelActual += 1
    }
}
```

- Repeat-while

Evalúa la condición al final de cada iteración

La sintaxis es la siguiente:

```
● ● ●

repeat {
    // Código a ejecutar
} while (<condicion>)
```

El contenido de este ciclo se ejecuta una vez (antes de comprobar la expresión de prueba). Solo entonces, se verifica la condición, ejemplo:

```
● ● ●

var nivelActual:Int = 0, nivelFinal:Int = 5
let juegoCompletado = true
repeat {
    //jugar
    if juegoCompletado {
        print("Acabas de completar el nivel \(nivelActual)")
        nivelActual += 1
    }
} while (nivelActual <= nivelFinal)
```

2.2.3. Arreglos

Un arreglo es un conjunto de datos de un mismo tipo ordenados de forma lineal, los componentes de un arreglo se deben indicar con el nombre del arreglo y un índice para referenciar al componente deseado.

Para crear un arreglo podemos hacerlo de diferentes maneras:

```
● ● ●

let nombreDelArreglo: Array = [<contenido>]

var nombreArreglo: Array<String>

var nombreDeArreglo: [String]
```

Dependiendo de nuestras necesidades podemos usar los arreglos a nuestro beneficio, ejemplo:



```
let unArreglo: Array = ["Paco", "Pedro", "Simon"]

let otroArreglo: [String] = ["Paco", "Pedro", "Simon"]

// Arreglo vacío
var unosInts = [Int]()
```

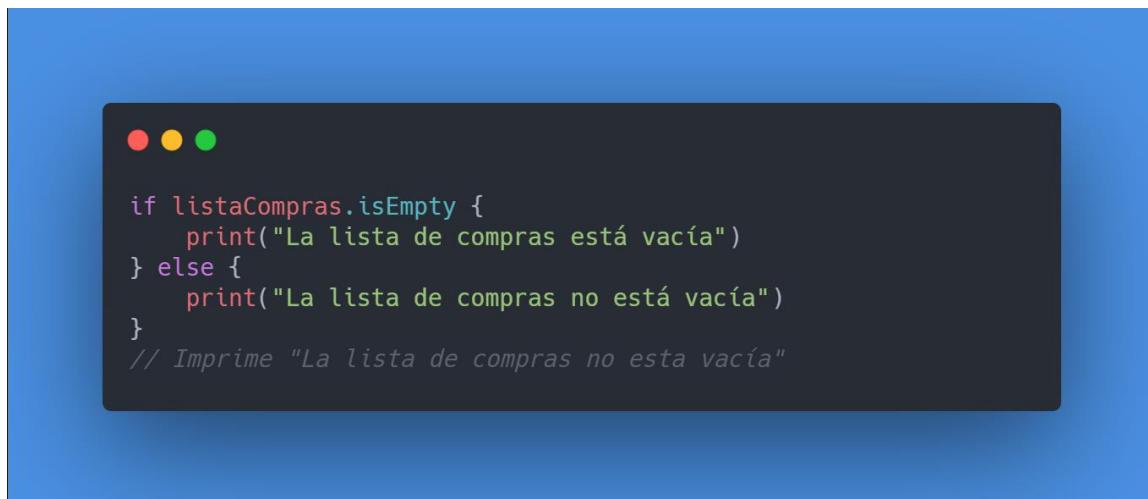
Así como crearlos podemos usarlos y modificar su contenido

Para ver cuantos elementos tiene un arreglo usamos .count, ejemplo:



```
print("La lista de compras contiene \(listaCompras.count) artículos.")
// Imprime "La lista de compras contiene 2 artículos."
```

Podemos utilizar la propiedad isEmpty para verificar si el arreglo esta vacío o no, ejemplo:



```
if listaCompras.isEmpty {
    print("La lista de compras está vacía")
} else {
    print("La lista de compras no está vacía")
}
// Imprime "La lista de compras no esta vacía"
```

Para agregar un nuevo elemento al final del arreglo utilizamos .append, ejemplo:

```
● ● ●  
listaCompras.append("Harina")  
// listaCompras ahora tiene 3 artículos
```

Para eliminar elementos usamos el método remove(at:). Este método elimina el elemento del índice especificado y devuelve el elemento eliminado, ejemplo:

```
● ● ●  
let harina = listaCompras.remove(at: 0)  
// lista de compras ahora tiene 2 elementos y ya no tiene  
// Harina  
// harina es una constante que tiene el string "Harina"
```

2.2.4. Funciones

Las funciones son fragmentos de código que realizan una tarea específica. Se le da a una función un nombre que identifica lo que hace, y este nombre se usa para "llamar" a la función para que realice la tarea definida cuando sea necesario.

La sintaxis es la siguiente:

```
● ● ●  
func nombreFuncion() {  
    // Código a ejecutar  
}
```

Podemos crear funciones utilizando parámetros, esta es la sintaxis:

```
func funcionParametro(size size:Int) {  
    // Código utilizando size, el cual es el parametro  
}
```

Ejemplo de una función utilizando parámetros

```
func imprimirMensaje(mensaje: String, ocasiones: Int) {  
    for i in 0..        println("\(i) \(mensaje)")  
    }  
}
```

2.2.5. Tuplas

Las tuplas agrupan varios valores en un solo valor compuesto. Los valores dentro de una tupla pueden ser de cualquier tipo y no es necesario que sean del mismo tipo entre sí. Las tuplas se crean agrupando cualquier cantidad de valores, la sintaxis sería la siguiente:

```
let tupla = ("uno", 2, "tres")
```

También se pueden nombrar valores individuales cuando se define la tupla:

```
let tuplaNombrada = (primero: 1, medio: "dos", ultimo: 3)  
  
// los valores se pueden leer con el nombre de la propiedad  
print(tuplaNombrada.primero) // 1  
print(tuplaNombrada.medio) // dos  
  
// y aún se puede utilizar el numero del indice  
print(tuplaNombrada.2) // 3
```

2.2.6. Enumeraciones

Son una manera de definir valores específicos para cubrir ciertas opciones en base a un grupo específico.

Las enumeraciones son usadas más que nada para definir posibilidades ante el manejo de un control en específico y constante.

La sintaxis sería la siguiente:



A screenshot of the Xcode code editor. It shows a dark-themed interface with three colored dots (red, yellow, green) at the top. The code area contains the following Swift code:

```
import Foundation

enum NivelDeComplejidad{
    case alto
    case intermedio
    case bajo
}
```

Definiendo el nombre del grupo por medio de la palabra reservada “enum” seguido del identificador que le queremos dar, posteriormente se abren llaves para indicar dentro las distintas opciones a considerar por medio de la palabra reservada “case” y su debido identificador.

Otra manera más sencilla, sería por medio del uso de las comas para definir cada uno de los posibles casos.



A screenshot of the Xcode code editor. It shows a dark-themed interface with three colored dots (red, yellow, green) at the top. The code area contains the following Swift code:

```
import Foundation

enum NivelDeComplejidad{
    case alto, intermedio, bajo
}
```

Otra característica a resaltar en los enumeradores es que se le pueden asociar valores, de tal manera que sabemos más de manera específica a que se refiere.



```
import Foundation

enum SistemaOperativo{
    case windows(String)
    case macOS(String)
    case linux(String)
}

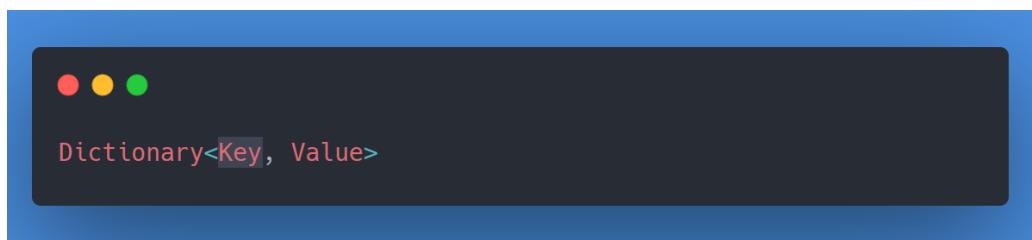
var miSistemaOperativo = SistemaOperativo.windows("10")

switch miSistemaOperativo {
    case .windows(let valor):
        print("Mi sistema operativo es Windows \(valor)")
    default:
        break
}
```

2.2.7. Diccionarios

Un diccionario almacena asociaciones entre claves del mismo tipo y valores del mismo tipo en una colección sin un orden definido. Cada valor está asociado con una clave única, que actúa como un identificador para ese valor dentro del diccionario. A diferencia de los elementos de un arreglo, los elementos de un diccionario no tienen un orden especificado. Se usa un diccionario cuando se necesita buscar valores basados en su identificador, de la misma manera que se usa un diccionario del mundo real para buscar la definición de una palabra en particular.

La sintaxis es la siguiente:



Podemos crear un diccionario vacío de la siguiente manera:

```
var nombresDeInts = [Int: String]()
```

También podemos hacer diccionarios con la combinación llave: valor, estos son diccionarios literales

```
var aeropuertos: [String: String] = ["YYZ": "Toronto Pearson",
"DUB": "Dublin"]
```

Podemos utilizar métodos similares a los que usamos con los arreglos, por ejemplo es posible usar `.isEmpty` para verificar si un diccionario está vacío o no

```
if aeropuertos.isEmpty {
    print("El diccionario de aeropuertos está vacío")
} else {
    print("El diccionario de aeropuertos no está vacío")
}
// Imprime "El diccionario de aeropuertos no está vacío"
```

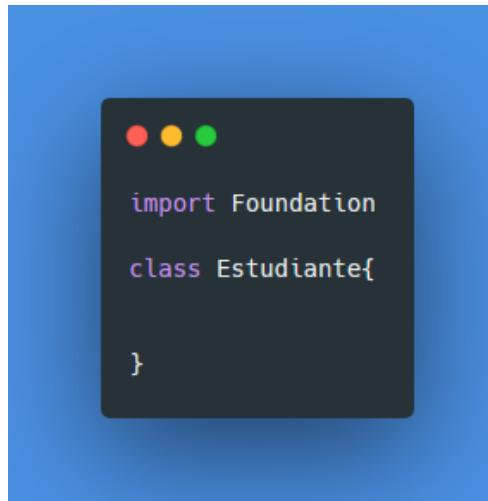
Podemos agregar elementos al diccionario de la siguiente manera:

```
aeropuertos["LHR"] = "London"
// el diccionario de aeropuertos ahora tiene 3 elementos
```

2.2.8. Clases

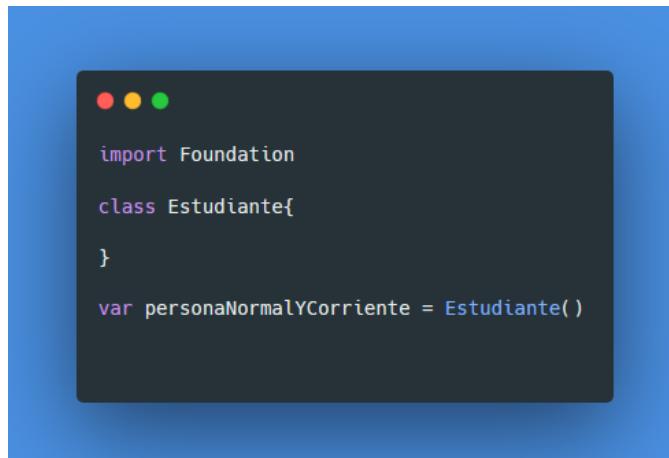
Swift al igual que muchos otros lenguajes de programación soporta el paradigma de programación orientada a objetos, dando como resultado un conjunto de palabras reservadas y normas generales para la implementación de este paradigma en este lenguaje de programación.

Para definir una clase en Swift es por medio de la palabra reservada “class” seguido del nombre de la clase que nosotros le queramos definir, por buena practica en este entorno de Swift, se recomienda utilizar la notación camello, quedando de la siguiente manera.



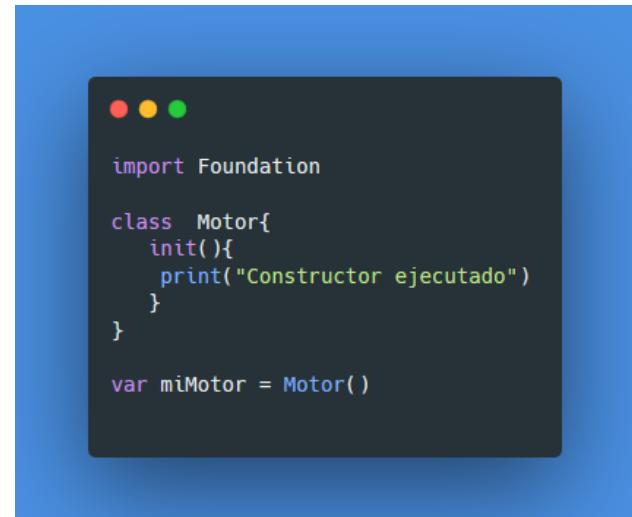
```
import Foundation
class Estudiante{}
```

Para definir una instancia de la clase, sería de la siguiente forma:



```
import Foundation
class Estudiante{}
var personaNormalYCorriente = Estudiante()
```

Para definir uno o varios constructores (según sea la necesidad) se hace uso de la palabra reservada “init” seguido de dos paréntesis y la apertura y cierre de llaves para definir las instrucciones a realizar durante la ejecución de este constructor.



```
import Foundation

class Motor{
    init(){
        print("Constructor ejecutado")
    }
}

var miMotor = Motor()
```

Para que una clase pueda heredar de otra, es necesario definirlo por medio del símbolo de dos puntos y seguido de la clase a la que queremos hacer referencia.



```
import Foundation

class Persona{
    var nombre : String { get{return nombre} set{nombre = newValue} }
    var apellido: String { get{return apellido} }
}

class Estudiante : Persona{
    var semestre: Int?
}
```

De esta forma la clase Estudiante tiene acceso tanto a las propiedades como a los métodos que la clase Persona tiene.

2.2.8.1. Propiedades

Para definir propiedades dentro de una clase es por medio de la palabra “var”, o bien por medio de la palabra “let” si nuestra propiedad nunca va a cambiar de valor.

Para definir la accesibilidad hacia nuestras propiedades es por medio de las palabras reservadas “public” para propiedades públicas y “private” para propiedades privadas.

Por defecto todas las propiedades que se definen dentro de la clase serán públicas, por lo que es indiferente si nosotros la dejamos como tal o le agregamos la palabra “public”.

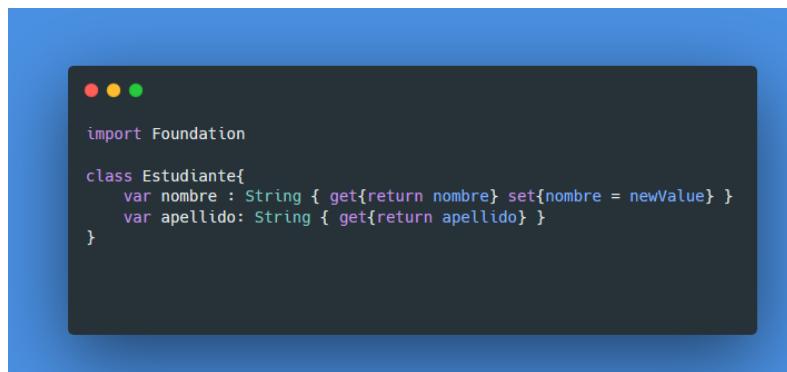


```
import Foundation

class Estudiante{
    //Esto
    var nombre : String = ""
    // Y esto, son prop publicas
    public var apellido : String = ""

}
```

Para definir si una propiedad solo será de lectura o escritura, se pueden definir por medio de las palabras reservadas “get” y “set”.



```
import Foundation

class Estudiante{
    var nombre : String { get{return nombre} set{nombre = newValue} }
    var apellido: String { get{return apellido} }
}
```

Para acceder a las propiedades de una instancia, seria por medio del nombre de la instancia de la clase que se definió, seguido de un punto y el nombre de la propiedad accesible, tal que así:



```
import Foundation

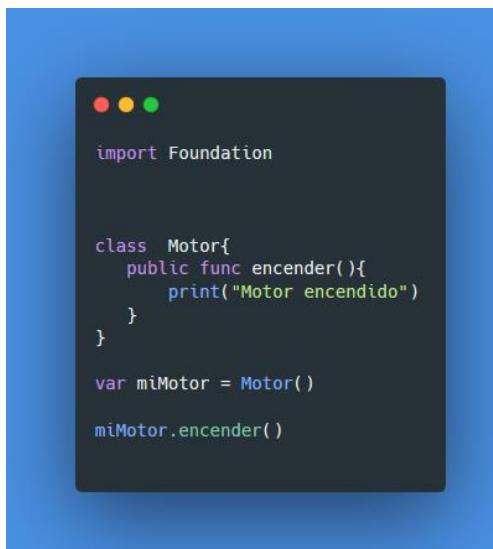
class Estudiante{
    var nombre : String?
    var apellido: String { get{return apellido} }
}

var personaNormalYCorriente = Estudiante()
personaNormalYCorriente.nombre = "Felipe"
print(personaNormalYCorriente.nombre!)
```

2.2.8.2. Métodos

Para definir los métodos de una clase basta con usar la manera tradicional que utilizamos para crear funciones fuera de una clase, lo único que vendría a cambiar es la manera en que hacemos referencia a él y la manera en que nosotros podemos definir la accesibilidad del mismo. Por default, al igual que las propiedades son públicas, por lo tanto, da igual si dejamos como tal la palabra reservada “public” así como si no la ponemos, sin embargo, si queremos definir que un método no va a ser accesible fuera de la clase, sí hay que definirlo por medio de la palabra reservada “private”.

La sintaxis para definir y hacer uso del método de la clase sería la siguiente:



```
import Foundation

class Motor{
    public func encender(){
        print("Motor encendido")
    }
}

var miMotor = Motor()

miMotor.encender()
```

Si nosotros queremos hacer referencia a una propiedad o alguna otra función referente al contexto de la clase se hace por medio de la palabra reservada “self” seguida del recurso al cual queremos hacer referencia, la cual quedaría de la siguiente manera:



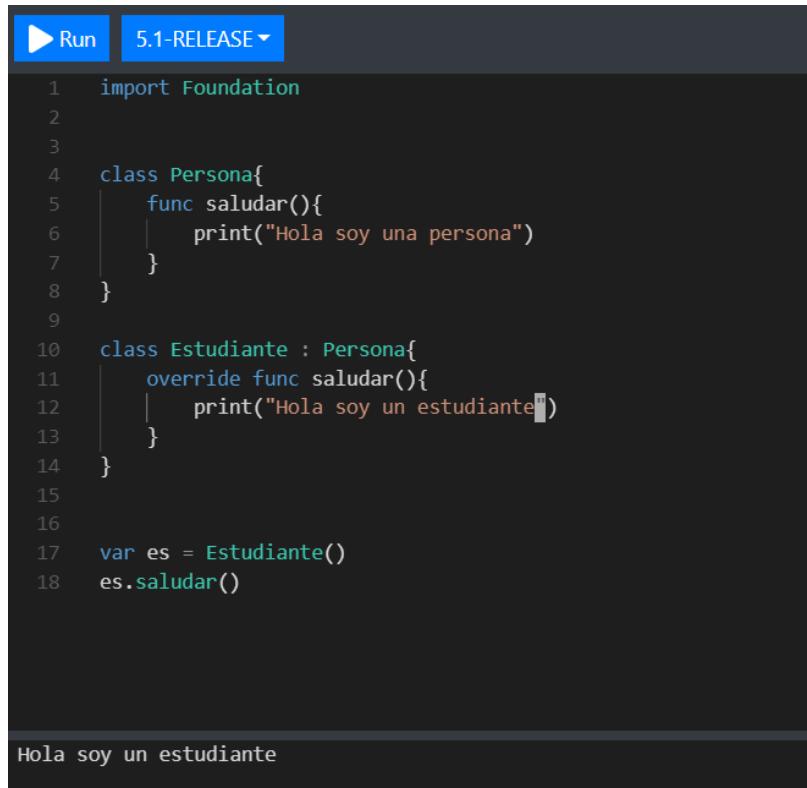
```
import Foundation

var modelo: String? = "002332"

class Motor{
    public func encender(){
        print("Motor \(self.modelo) encendido")
    }
}

var miMotor = Motor()
miMotor.encender()
```

Para sobrescribir un método heredado, tendría que definirse por medio de la palabra reservada “override” y el nombre del método a sobrescribir (siempre y cuando el método no se encuentre sellado).



```
Run 5.1-RELEASE ▾

1 import Foundation
2
3
4 class Persona{
5     func saludar(){
6         print("Hola soy una persona")
7     }
8 }
9
10 class Estudiante : Persona{
11     override func saludar(){
12         print("Hola soy un estudiante")
13     }
14 }
15
16
17 var es = Estudiante()
18 es.saludar()
```

Hola soy un estudiante

2.3 PRACTICAS DE PROGRAMACIÓN

Juan Felipe Garza Sánchez



2.3.1. Se tiene la edad de Loky (en años perro) en una variable. Determine la edad de Loky en años humanos, considerando que 1 año humano equivale a 7 años perro.

Pruebas

edadLokyPerro	edadLokyHumano
50	7
14	2
32	4

```
import Foundation

var edadLoky = 32
var edadLokyHumano = edadLoky / 7
print(edadLokyHumano)
```

Execute | Share | main.swift | STDIN | Result

```
1 import Foundation
2
3 var edadLoky = 50
4 var edadLokyHumano = edadLoky / 7
5 print(edadLokyHumano)
```

\$swift main.swift
7

Execute | Share | main.swift | STDIN | Result

```
1 import Foundation
2
3 var edadLoky = 14
4 var edadLokyHumano = edadLoky / 7
5 print(edadLokyHumano)
```

\$swift main.swift
2

Execute | Share | main.swift | STDIN | Result

```
1 import Foundation
2
3 var edadLoky = 32
4 var edadLokyHumano = edadLoky / 7
5 print(edadLokyHumano)
```

\$swift main.swift
4

Se logró realizar el ejercicio por medio del operador de división y la inferencia de datos que se maneja en Swift para la asignación de los valores según la razón de conversión que se dio en el ejercicio.

2.3.2. Se tiene el tiempo de un recorrido, almacenado en tres variables: una para la hora, otra para los minutos y otra para los segundos; así también, se tiene la distancia (en metros) en otra variable. Despliegue la velocidad en metros/segundo, kilómetros/hora y millas/hora.

Pruebas							
distancia	horas	minutos	segundos	Mts/seg	Km/h	Millas/h	
2500	5	56	23	0.116915	0.420895	0.261588	
50000	1	35	56	8.686588	31.27172	19.4355	

```
import Foundation
var hora: Float = 1
var minutos: Float = 35
var segundos: Float = 56
var distanciaMetros: Float = 50000
var distanciaKM: Float = distanciaMetros / 1000
var distanciaMi: Float = distanciaKM / 1.60934
var segundosTotales: Float = segundos + (minutos * 60) + (hora * 60 * 60)
var horasTotales: Float = (segundosTotales / 60) / 60
var ms: Float = distanciaMetros / segundosTotales
var kmh: Float = distanciaKM / horasTotales
var mih : Float = distanciaMi / horasTotales
print( ms )
print( kmh )
print( mih )
```

Execute | Share main.swift STDIN Result

```
1 import Foundation
2 var hora: Float = 1
3 var minutos: Float = 35
4 var segundos: Float = 56
5 var distanciaMetros: Float = 50000
6 var distanciaKM: Float =
    distanciaMetros / 1000
```

\$swift main.swift
8.68659
31.2717
19.4314

Execute | Share main.swift STDIN Result

```
1 import Foundation
2 var hora: Float = 5
3 var minutos: Float = 56
4 var segundos: Float = 23
5 var distanciaMetros: Float = 2500
6 var distanciaKM: Float =
    distanciaMetros / 1000
7 var distanciaMi: Float = distanciaKM
```

\$swift main.swift
0.116915
0.420895
0.261533

El ejercicio se logró realizar por operaciones básicas de multiplicación, división y suma, siguiendo siempre y cuando las reglas para las conversiones necesarias.

2.3.3. Dadas la suma y la diferencia entre dos números, encuentre los valores de dichos números y almacénelos en variables llamadas *a* y *b*. Imprima los resultados.

Pruebas

sum	dif	a	b
16	4	10	6
11	3	7	4
4	2	3	1

```
import Foundation
var sum = 16
var dif = 4
var a = (sum + dif) / 2
var b = (sum - dif) / 2
print(a)
print(b)
```

```
Execute | Share main.swift STDIN Result
1 import Foundation
2 var sum = 16
3 var dif = 4
4 var a = (sum + dif) / 2
5 var b = (sum - dif) / 2
6 print(a)
7 print(b)

$swift main.swift
10
6
```

```
Execute | Share main.swift STDIN Result
1 import Foundation
2 var sum = 11
3 var dif = 3
4 var a = (sum + dif) / 2
5 var b = (sum - dif) / 2
6 print(a)
7 print(b)

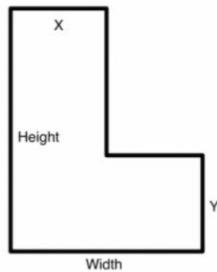
$swift main.swift
7
4
```

```
Execute | Share main.swift STDIN Result
1 import Foundation
2 var sum = 4
3 var dif = 2
4 var a = (sum + dif) / 2
5 var b = (sum - dif) / 2
6 print(a)
7 print(b)

$swift main.swift
3
1
```

Simplemente se hacen uso de los operadores de suma y resta para asignar los resultados en dos variables.

2.3.4. Se tienen cuatro variables (*ancho*, *alto*, *x*, *y*), que describen las dimensiones de una figura en forma de **L**. Calcule, almacene e imprima el perímetro y el área de dicha figura.



Pruebas					
ancho	alto	x	y	perímetro	área
8	12	4	3	40	60
8	4	2	2	24	20

```
import Foundation
var ancho = 8
var alto = 4
var x = 2
var y = 2
var perimetro = x + alto + ancho + y + (ancho - x) + (alto - y)
print(perimetro)
var area = ((ancho * y) + (x * alto)) - (x * y)
print(area)
```

Screenshot of a Swift code editor showing two runs of the program:

Run 1 (Inputs: ancho=8, alto=4, x=2, y=2):

```
Execute | Share | main.swift | STDIN | Result
$swift main.swift
24
20
```

Run 2 (Inputs: ancho=8, alto=12, x=4, y=3):

```
Execute | Share | main.swift | STDIN | Result
$swift main.swift
40
60
```

Por medio de la suma de las áreas que se conocen como lo es los dos rectángulos que se logran juntar en una misma figura, menos la intersección que se encuentran entre ambos, para así no tener esa intersección de más.

2.3.5. Dado un número de tres dígitos almacenado en *a*, calcule aritméticamente e imprima el último dígito.

Pruebas

a	ultimo digito
123	3
337	7
100	0

```
import Foundation
var a = 337
var centenas = (a / 100) * 100
var decimas = ((a - centenas) / 10) * 10
var unidades = a - centenas - decimas
print(unidades)
```

Execute | Share main.swift STDIN Result

```
1 import Foundation
2 var a = 123
3 var centenas = (a / 100) * 100
4 var decimas = ((a - centenas) / 10) *
    10
5 var unidades = a - centenas - decimas
6 print(unidades)
```

\$swift main.swift
3

Execute | Share main.swift STDIN Result

```
1 import Foundation
2 var a = 337
3 var centenas = (a / 100) * 100
4 var decimas = ((a - centenas) / 10) *
    10
5 var unidades = a - centenas - decimas
6 print(unidades)
```

\$swift main.swift
7

Execute | Share main.swift STDIN Result

```
1 import Foundation
2 var a = 100
3 var centenas = (a / 100) * 100
4 var decimas = ((a - centenas) / 10) *
    10
```

\$swift main.swift
0

Sabiendo que son 3 dígitos, simplemente se procedió a sacar las veces que cabían en razón de las centenas, decenas y unidades del valor dado.

2.3.6. En x años a partir de ahora, Anahí será y veces mayor en edad que su hermana Abril. Conociendo la edad actual de Abril, calcule la edad de Anahí.

Pruebas

x	y	abril	anahi
3	2	12	27
1	3	12	38

```
import Foundation
var x = 3
var y = 2
var abril = 12
var anahi = ((abril + x) * y) - x
print(anahi)
```

Execute | Share main.swift | STDIN | Result

```
1 import Foundation
2 var x = 3
3 var y = 2
4 var abril = 12
5 var anahi = ((abril + x) * y) - x
6 print(anahi)
```

\$swift main.swift
27

Execute | Share main.swift | STDIN | Result

```
1 import Foundation
2 var x = 1
3 var y = 3
4 var abril = 12
5 var anahi = ((abril + x) * y) - x
6 print(anahi)
```

\$swift main.swift
38

Sabiendo la razón de edades entre Abril y Anahí, se puede intuir la formula para sacar los resultados esperados. Dicha razón es los años más la edad de Abril por la veces que será mayor menos los años.

2.3.7. Suponga que usted tiene x manzanas. Lizbeth intercambia 3 naranjas por 5 manzanas. ¿Cuántas naranjas puede obtener de Lizbeth y cuantas manzanas le quedarían después del intercambio? Imprima los resultados.

Pruebas

x	manzanas	naranjas
17	2	9
25	0	15
4	4	0

```
import Foundation
var x = 4
var naranjas = (x / 5) * 3
var manzanasRestantes = x - (naranjas / 3 * 5)
print(naranjas)
print(manzanasRestantes)
```

The screenshot shows three separate code execution environments, each with an "Execute" button, a "Share" button, and tabs for "main.swift" and "STDIN". The first environment has x=4, resulting in 9 naranjas and 2 manzanasRestantes. The second environment has x=25, resulting in 15 naranjas and 0 manzanasRestantes. The third environment has x=17, resulting in 2 naranjas and 9 manzanasRestantes. The "Result" pane shows the command \$swift main.swift followed by the output values.

x	naranjas	manzanasRestantes
4	9	2
25	15	0
17	2	9

Sabiendo la razón de manzanas a naranjas se puede intuir operaciones básicas para el cálculo de las naranjas resultantes.

2.3.8. Dados dos números en a y b , determine e imprima el mayor de ellos o indique si son iguales.

Pruebas

a	b	mayor
11	22	22
23	12	23
8	8	Iguales

```
import Foundation  
var a = 8  
var b = 8  
if a > b{  
    print(a)  
}  
else if b > a{  
    print(b)  
}  
else{  
    print("iguales")  
}
```

The screenshot shows a terminal interface with three separate code executions and their corresponding results:

- Execution 1:** The code compares $a = 11$ and $b = 22$. The result is 22 .
- Execution 2:** The code compares $a = 23$ and $b = 12$. The result is 23 .
- Execution 3:** The code compares $a = 8$ and $b = 8$. The result is $iguales$.

2.3.9. Imprima si el valor en *x* es par o impar.

Pruebas

x	resultado
31	Impar
38	Par

```
import Foundation
var x = 38
if x % 2 == 0 {
    print("Par")
}else{
    print("Impar")
}
```

Execute | > Share [main.swift] [STDIN] | Result

```
1 import Foundation
2 var x = 31
3 if x % 2 == 0 {
4     print("Par")
```

\$swift main.swift
Impar

Execute | > Share [main.swift] [STDIN] | Result

```
1 import Foundation
2 var x = 38
3 if x % 2 == 0 {
4     print("Par")
```

\$swift main.swift
Par

2.3.10. Tiene dos números en a y b . Indique si a es divisible entre b .

Pruebas		
a	b	resultado
22	11	Divisible
12	5	No divisible
18	9	Divisible

```
import Foundation
var a = 18
var b = 9
if a % b == 0 {
    print("Divisible")
}else{
    print("No divisible")
}
```

Execute | > Share main.swift | STDIN | Result

```
1 import Foundation
2 var a = 22
3 var b = 11
4 if a % b == 0 {
```

\$swift main.swift
Divisible

Execute | > Share main.swift | STDIN | Result

```
1 import Foundation
2 var a = 12
3 var b = 5
4 if a % b == 0 {
```

\$swift main.swift
No divisible

Execute | > Share main.swift | STDIN | Result

```
1 import Foundation
2 var a = 18
3 var b = 9
4 if a % b == 0 {
```

\$swift main.swift
Divisible

2.3.11. Se tienen tres variables (a, b, c). Imprima la leyenda “Al menos dos variables son iguales” o bien la leyenda “Todas las variables son diferentes”, según sea el caso.

Pruebas

a	b	c	resultado
1	2	3	Todas las variables son diferentes
1	2	1	Al menos dos variables son iguales
8	8	8	Al menos dos variables son iguales

```
import Foundation
var a = 1
var b = 2
var c = 1
if a == b || b == c || a == c
{
    print("Al menos dos variables son iguales")
}
else
{
    print("Todas las variables son diferentes")
}
```

Execute Share main.swift STDIN	Result
1 import Foundation 2 var a = 1 3 var b = 2 4 var c = 3 5 if a == b b == c a == c	\$swift main.swift Todas las variables son diferentes
Execute Share main.swift STDIN	Result
1 import Foundation 2 var a = 1 3 var b = 2 4 var c = 1 5 if a == b b == c a == c 6 { 7 print("Al menos dos variables	\$swift main.swift Al menos dos variables son iguales
Execute Share main.swift STDIN	Result
1 import Foundation 2 var a = 8 3 var b = 8 4 var c = 8	\$swift main.swift Al menos dos variables son iguales

2.3.12. Antes de preparar el desayuno, debemos saber si aún se pueden cocinar los ingredientes. Considere que los blanquillos se echan a perder después de tres semanas (21 días) y el tocino después de una semana (7 días). Dados los días de los blanquillos y el tocino en su refrigerador, determine si ambos se pueden cocinar, o si hay alguno que deba desecharse.

Pruebas

blanquillos	tocino	resultado
2	3	Ambos se pueden cocinar
20	9	Debe desechar el tocino
23	11	Debe desechar ambos
25	6	Debe desechar los blanquillos

```
import Foundation
var blanquillos = 25
var tocino = 6

if blanquillos > 21 && tocino > 7 {
    print("Debe desechar ambos")
}else if blanquillos > 21{
    print("Debe desechar los blanquillos")
}else if tocino > 7
{
    print("Debe desechar el tocino")
}else{
    print("Ambos se pueden cocinar")
}
```

Execute | Share main.swift STDIN Result

```
1 import Foundation
2 var blanquillos = 2
3 var tocino = 3
```

\$swift main.swift
Ambos se pueden cocinar

Execute | Share main.swift STDIN Result

```
1 import Foundation
2 var blanquillos = 20
3 var tocino = 9
```

\$swift main.swift
Debe desechar el tocino

Execute | Share main.swift STDIN Result

```
1 import Foundation
2 var blanquillos = 23
3 var tocino = 11
```

\$swift main.swift
Debe desechar ambos

Execute | Share main.swift STDIN

```
1 import Foundation
2 var blanquillos = 25
3 var tocino = 6
4
5 if blanquillos > 21 && tocino > 7 {
```

Result

```
$swift main.swift
Debe desechar los blanquillos
```

2.3.13. Dado un año, imprima si es o no es bisiesto. Las reglas (para este ejercicio) para determinar un año bisiesto, son las siguientes:

- Divisible entre 4
- Cada inicio de siglo, si el año es divisible entre 400

Pruebas

año	resultado
2000	Es bisiesto
1985	No es bisiesto
1900	No es bisiesto
2016	Es bisiesto

```
import Foundation
var año = 2016

if(año % 4 == 0) && ((año % 100 != 0) || (año % 400 == 0)){
    print("Es bisiesto")
}else{
    print("No es bisiesto")
}
```

Execute | Share main.swift STDIN

```
1 import Foundation
2 var año = 2000
3
4 if(año % 4 == 0) && ((año % 100 != 0)
```

Result

```
$swift main.swift
Es bisiesto
```

Execute | Share main.swift STDIN

```
1 import Foundation
2 var año = 1985
3
4 if(año % 4 == 0) && ((año % 100 != 0)
```

Result

```
$swift main.swift
No es bisiesto
```

Execute | Share main.swift STDIN

```
1 import Foundation
2 var año = 1900
3
4 if(año % 4 == 0) && ((año % 100 != 0)
```

Result

```
$swift main.swift
No es bisiesto
```

Execute | Share main.swift STDIN

```
1 import Foundation
2 var año = 2016
3
4 if(año % 4 == 0) && ((año % 100 != 0)
```

Result

```
$swift main.swift
Es bisiesto
```

2.3.14. La función `Int.random(in: mínimo...máximo)` entrega un valor aleatorio, comprendido entre mínimo y máximo. Utilice dicha función para simular que lanza un volado con una moneda y determine si cayó águila (par) o sello (impar).

```
import Foundation
var moneda = Int.random(in: 1...2)

if (moneda % 2) == 0 {
    print("Aguila")
}
else{
    print("Sello")
}
```

The image displays two side-by-side screenshots of a Swift code editor interface. Both screenshots show the same code for flipping a coin using the `Int.random` function. The code is as follows:

```
import Foundation
var moneda = Int.random(in: 1...2)

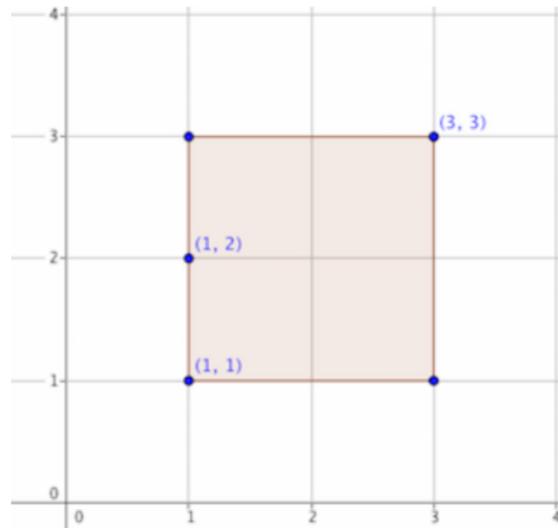
if (moneda % 2) == 0 {
    print("Aguila")
}
else{
    print("Sello")
}
```

In the top bar of both screenshots, there is a "Run" button and a dropdown menu set to "5.1-RELEASE".

The left screenshot shows the output "Sello" at the bottom of the editor window.

The right screenshot shows the output "Aguila" at the bottom of the editor window.

2.3.15. Determine si un punto o coordenada (x, y) se encuentra dentro de un rectángulo definido por las coordenadas de la esquina inferior izquierda (x_1, y_1) y la esquina superior derecha (x_2, y_2) .



```
import Foundation
var x = 0
var y = 4
var x1 = 1
var y1 = 0
var x2 = 4
var y2 = 3
if (x >= x1 && x <= x2) && (y >= y1 && y <= y2)
{
    print("Dentro")
} else {
    print("Fuera")
}
```

Execute | Share | main.swift | STDIN | Result

```
1 import Foundation
2 var x = 0
3 var y = 4
4 var x1 = 1
5 var y1 = 0
6 var x2 = 4
7 var y2 = 3
8 if (x >= x1 && x <= x2) && (y >=
    y1 && y <= y2 )
9 {
10     print("Dentro")
11 } else{
12     print("Fuera")
13 }
```

\$swift main.swift
Fuera

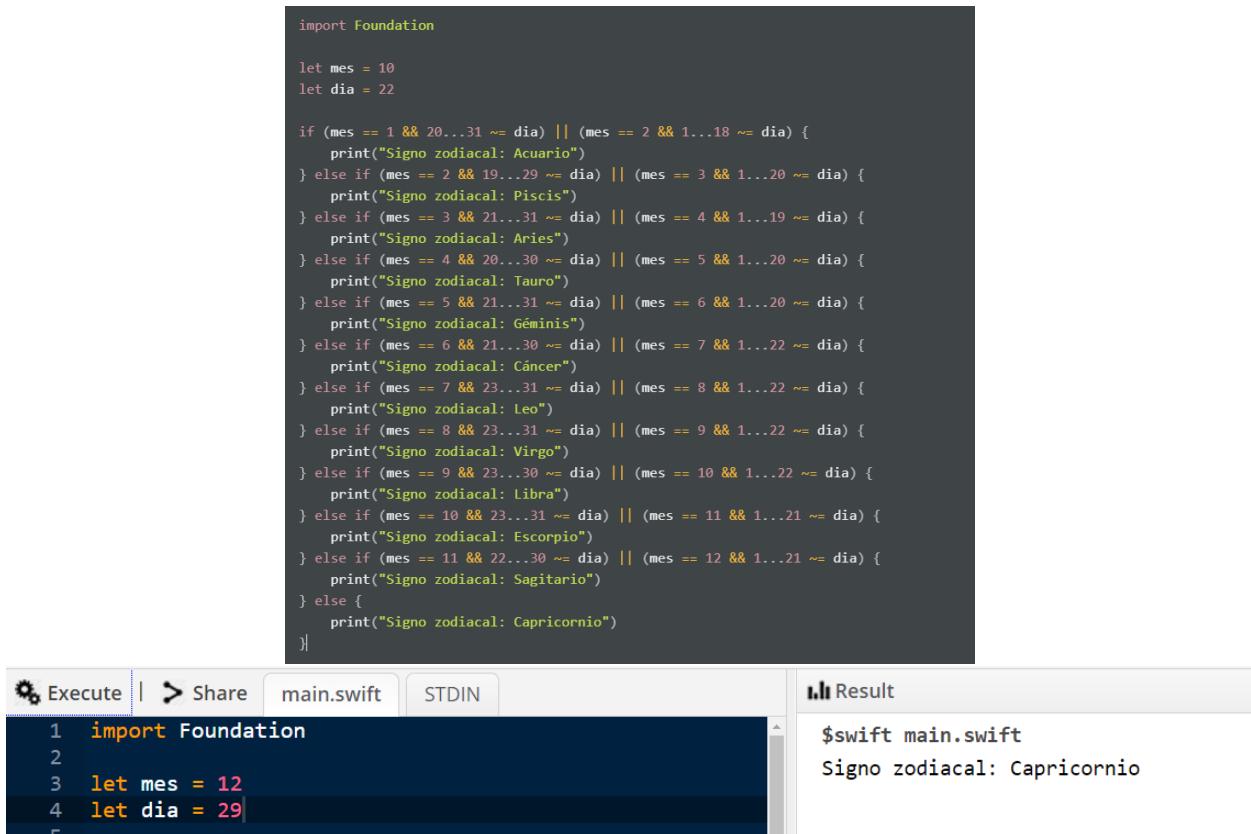
2.3.16. Dado un carácter, determine si es vocal, consonante, dígito, operador aritmético u otro.

```
import Foundation
var carácter: Character = "/"
let vocales: Array<Character> = ["a","e","i","o","u"]
var code = carácter.asciiValue!
switch(code){
    case 65...90, 97...122:
        if vocales.contains(Character(String(carácter).lowercased())){
            print("Es vocal")
        }
        else{
            print("Es constante")
        }
    case 48...57:
        print("Es un dígito")
    case 37,42,43,45,47:
        print("Operador aritmético")
    default:
        print("Otro")
}
```

```
import Foundation
var carácter: Character = "b"
let vocales: Array<Character> = ["a","e","i","o","u"]
var code = carácter.asciiValue!
switch(code){
    case 65...90, 97...122:
        if vocales.contains(Character(String(carácter).lowercased())){
            print("Es vocal")
        }
        else{
            print("Es constante")
        }
    case 48...57:
        print("Es un dígito")
    case 37,42,43,45,47:
        print("Operador aritmético")
    default:
        print("Otro")
```

Operador aritmético Es constante

2.3.17. En base al mes y día de nacimiento, determine el signo zodiacal de una persona.



```
import Foundation

let mes = 10
let dia = 22

if (mes == 1 && 20...31 ~= dia) || (mes == 2 && 1...18 ~= dia) {
    print("Signo zodiacal: Acuario")
} else if (mes == 2 && 19...29 ~= dia) || (mes == 3 && 1...20 ~= dia) {
    print("Signo zodiacal: Piscis")
} else if (mes == 3 && 21...31 ~= dia) || (mes == 4 && 1...19 ~= dia) {
    print("Signo zodiacal: Aries")
} else if (mes == 4 && 20...30 ~= dia) || (mes == 5 && 1...20 ~= dia) {
    print("Signo zodiacal: Tauro")
} else if (mes == 5 && 21...31 ~= dia) || (mes == 6 && 1...20 ~= dia) {
    print("Signo zodiacal: Géminis")
} else if (mes == 6 && 21...30 ~= dia) || (mes == 7 && 1...22 ~= dia) {
    print("Signo zodiacal: Cáncer")
} else if (mes == 7 && 23...31 ~= dia) || (mes == 8 && 1...22 ~= dia) {
    print("Signo zodiacal: Leo")
} else if (mes == 8 && 23...31 ~= dia) || (mes == 9 && 1...22 ~= dia) {
    print("Signo zodiacal: Virgo")
} else if (mes == 9 && 23...30 ~= dia) || (mes == 10 && 1...22 ~= dia) {
    print("Signo zodiacal: Libra")
} else if (mes == 10 && 23...31 ~= dia) || (mes == 11 && 1...21 ~= dia) {
    print("Signo zodiacal: Escorpio")
} else if (mes == 11 && 22...30 ~= dia) || (mes == 12 && 1...21 ~= dia) {
    print("Signo zodiacal: Sagitario")
} else {
    print("Signo zodiacal: Capricornio")
}
```

The screenshot shows a Swift code editor interface. The top part displays the source code for `main.swift`. The bottom part shows the execution results in the "Result" tab, where the command `$swift main.swift` is run and the output is "Signo zodiacal: Capricornio". The code itself defines variables `mes` and `dia`, and then uses a series of nested `if` statements to determine the zodiac sign based on the month and day. The logic covers all months from January to December, with specific ranges for each month's days and additional logic for the first few days of the next month.

2.3.18. Un número cuadrado perfecto es el producto de multiplicar un entero por sí mismo. Imprima los N cuadrados perfectos.

Pruebas

N	salida
2	1 4
5	1 4 9 16 25

```
import Foundation
var x = 5
for y in 1...x{
    print(y*y)
}
```

Execute | Share main.swift STDIN Result

```
1 import Foundation
2 var x = 2
3 for y in 1...x{
4     print(y*y)
5 }
```

```
$swift main.swift
1
4
```

Execute | Share main.swift STDIN Result

```
1 import Foundation
2 var x = 5
3 for y in 1...x{
4     print(y*y)
5 }
```

```
$swift main.swift
1
4
9
16
25
```

2.3.19. Imprima la serie de números de 1 a N alternando su orden, es decir, el primer número siempre es el 1, seguido de N , luego 2, luego $N-1$ y así sucesivamente. La serie debe imprimirse en un solo renglón, separando los números con espacios.

Pruebas

N	salida
4	1 4 2 3
9	1 9 2 8 3 7 4 6 5

```
import Foundation
var n = 9
var impar = 0;
var resultado = ""
for x in 1...n{
    if( x % 2 == 0){
        resultado = resultado + String( n - (impar-1) ) + " "
    }else{
        impar+=1
        resultado = resultado + String(impar) + " "
    }
}
print(resultado)
```

The screenshot shows two separate runs of the Swift code in a terminal-like interface. Each run consists of a code editor window and a results window.

Run 1 (Top):

- Code Editor:

```
1 import Foundation
2 var n = 4
3 var impar = 0;
4 var resultado = ""
5 for x in 1...n{
```
- Results Window:

Execute | Share main.swift STDIN Result

```
$swift main.swift
1 4 2 3
```

Run 2 (Bottom):

- Code Editor:

```
1 import Foundation
2 var n = 9
3 var impar = 0;
4 var resultado = ""
5 for x in 1...n{
```
- Results Window:

Execute | Share main.swift STDIN Result

```
$swift main.swift
1 9 2 8 3 7 4 6 5
```

2.3.20. Dado N , dibuje una pirámide de asteriscos. La pirámide debe tener N líneas. En la i -ésima línea debe haber $N-i$ espacios, seguido de i^2-1 asteriscos.

Pruebas

N	salida
1	*
2	* ***
3	* *** *****
4	* *** ***** ******

```
import Foundation

var n = 5
var cantidadDeAsteriscos = -1
for linea in 1...n {
    var cadena = ""
    cantidadDeAsteriscos += 2
    var espacios: Int = (n) - linea
    for _ in 1...cantidadDeAsteriscos {
        cadena += "*"
    }
    for _ in 0...espacios {
        cadena = " " + cadena
    }
    print(cadena)
}
```

Execute | Share main.swift STDIN Result

```
1 import Foundation
2
3 var n = 1
4 var cantidadDeAsteriscos = -1
```

\$swift main.swift
*

Execute | Share main.swift STDIN

```
1 import Foundation
2
3 var n = 2
4 var cantidadDeAsteriscos = -1
5 for linea in 1...n {
```

Result

```
$swift main.swift
*
***
```

Execute | Share main.swift STDIN

```
1 import Foundation
2
3 var n = 3
4 var cantidadDeAsteriscos = -1
5 for linea in 1...n {
```

Result

```
$swift main.swift
*
***
```

Execute | Share main.swift STDIN

```
1 import Foundation
2
3 var n = 4
4 var cantidadDeAsteriscos = -1
5 for linea in 1...n {
6     var cadena = ""
7     cantidadDeAsteriscos += 2
```

Result

```
$swift main.swift
*
***
```

2.3.21. Imprima los primeros N números de la serie de *Fibonacci*. Los primeros dos números de la serie siempre son 1, el resto son la suma de los dos anteriores. Imprima la serie en un solo renglón con los valores separados por coma.

Pruebas

N	salida
3	1, 1, 2
6	1, 1, 2, 3, 5, 8
10	1, 1, 2, 3, 5, 8, 13, 21, 34, 55

```
import Foundation

var n = 10
var a = 0
var b = 1
var c = 0
for x in 1..
```

The screenshot shows a code editor interface with the following components:

- Execute:** A button with a gear icon.
- Share:** A button with a share icon.
- File Name:** main.swift
- STDIN:** An input field.
- Result:** A terminal window showing the output of the executed code.

The code in the editor is:

```
1 import Foundation
2
3 var n = 3
4 var a = 0
5 var b = 1
6 var c = 0
```

The terminal output is:

```
$swift main.swift
1
1
2
```

Execute | Share main.swift STDIN

```
1 import Foundation
2
3 var n = 6
4 var a = 0
5 var b = 1
6 var c = 0
7 for x in 1..

Result



```
$swift main.swift
1
1
2
3
5
8
```


```

Execute | Share main.swift STDIN

```
1 import Foundation
2
3 var n = 10
4 var a = 0
5 var b = 1
6 var c = 0
7 for x in 1..

Result



```
$swift main.swift
1
1
2
3
5
8
13
21
34
55
```


```

2.3.22. Dado un número en x , determine si es un número primo. Los números primos solo pueden dividirse entre 1 y entre sí mismos (división exacta).

Pruebas

x	resultado
2	Es primo
3	Es primo
15	No es primo
1	No es primo

```
import Foundation
var x = 15
var primo = x != 1
if(primo){
    for y in 1...x{
        if(y == 1 || y == x){
        }else{
            if(x % y == 0){
                primo = false
                break
            }
        }
    }
}
if(primo){
    print("es primo")
}else{
    print("no es primo")
}
```

The screenshot shows a code editor interface with a dark theme. At the top, there are tabs for "Execute", "Share", "main.swift", "STDIN", and "Result". The "Execute" tab is currently selected. Below the tabs, the code for "main.swift" is displayed:

```
1 import Foundation
2 var x = 2
3 var primo = x != 1
```

To the right of the code, the "Result" tab shows the terminal output:

```
$swift main.swift
es primo
```

Execute | > Share main.swift STDIN

```
1
2 import Foundation
3 var x = 3
4 var primo = x != 1
```

Result

```
$swift main.swift
es primo
```

Execute | > Share main.swift STDIN

```
1
2 import Foundation
3 var x = 15
4 var primo = x != 1
5 if(primo){
```

Result

```
$swift main.swift
no es primo
```

Execute | > Share main.swift STDIN

```
1
2 import Foundation
3 var x = 1
4 var primo = x != 1
```

Result

```
$swift main.swift
no es primo
```

2.3.23. Convierta un número de decimal a binario con el método de escalera.

Pruebas

decimal	resultado
78	1001110
92	1011100
2015	11111011111

```
import Foundation

var decimal: Int = 2015
var actual: Int = decimal
var resultado = ""
while( actual >= 1){

    if(actual % 2 == 0){
        resultado = "0" + resultado
    }else{
        resultado = "1" + resultado
    }
    actual = actual / 2
}
print(resultado)
```

Execute | Share main.swift STDIN

```
1 import Foundation
2
3 var decimal: Int = 78
```

Result

```
$swift main.swift
1001110
```

Execute | Share main.swift STDIN

```
1 import Foundation
2
3 var decimal: Int = 92
4 var actual: Int = decimal
```

Result

```
$swift main.swift
1011100
```

Execute | Share main.swift STDIN

```
1 import Foundation
2
3 var decimal: Int = 2015
4 var actual: Int = decimal
```

Result

```
$swift main.swift
11111011111
```

2.3.24. Imprima el elemento mayor de un arreglo de N números.

Pruebas

Arreglo	mayor
[1, 2, 3, 10, 100]	100
[10, 12, 33, 11, 1, 8]	33

```
import Foundation

var arreglo: [Int] = [1, 2 ,3 ,10, 100]
arreglo.sort()
arreglo.reverse()
print(arreglo[0])
```

Execute | Share | main.swift | STDIN | Result

```
1 import Foundation
2
3 var arreglo: [Int] = [1, 2 ,3 ,10,
100]
4 arreglo.sort()
5 arreglo.reverse()
6 print(arreglo[0])|
```

\$swift main.swift
100

Execute | Share | main.swift | STDIN | Result

```
1 import Foundation
2
3 var arreglo: [Int] = [10, 12 ,33 ,11,
1, 8]
```

\$swift main.swift
33

2.3.25. Imprima los elementos de un vector en orden inverso al que se encuentran almacenados.

Pruebas

vector	salida
[1, 2, 3, 10, 100]	100, 10, 3, 2, 1
[10, 12, 33, 11, 1, 8]	8, 1, 11, 33, 12, 10

```
import Foundation

var arreglo: [Int] = [1, 2 ,3 ,10, 100]
arreglo.reverse()
print(arreglo)
```

The screenshot shows two separate executions of a Swift script named `main.swift`. Each execution consists of three parts: the code input, the execution status, and the output result.

Execution 1:

- Code Input:

```
1 import Foundation
2
3 var arreglo: [Int] = [1, 2 ,3 ,10,
100]
```
- Execution Status: "Execute" button is highlighted.
- Output Result:

```
$swift main.swift
[100, 10, 3, 2, 1]
```

Execution 2:

- Code Input:

```
1 import Foundation
2
3 var arreglo: [Int] = [10, 12, 33,11,1
,8]
```
- Execution Status: "Execute" button is highlighted.
- Output Result:

```
$swift main.swift
[8, 1, 11, 33, 12, 10]
```

2.3.26. Invierta los elementos de un arreglo sin crear otro.

Pruebas

arreglo	salida
[1, 2, 3, 10, 100]	[100, 10, 3, 2, 1]
[10, 12, 33, 11, 1, 8]	[8, 1, 11, 33, 12, 10]

```
import Foundation

var arreglo: [Int] = [1, 2 ,3 ,10, 100]
arreglo.reverse()
print(arreglo)
```

Execute | > Share main.swift STDIN | Result

```
1 import Foundation
2
3 var arreglo: [Int] = [1, 2 ,3 ,10,
100]
```

\$swift main.swift
[100, 10, 3, 2, 1]

Execute | > Share main.swift STDIN | Result

```
1 import Foundation
2
3 var arreglo: [Int] = [10, 12 ,33, 11,
1, 8]
```

\$swift main.swift
[8, 1, 11, 33, 12, 10]

2.3.27. Dados los vectores A y B , imprima todos los elementos de B que se encuentren en A . Si no existen elementos en común, no debe imprimir nada.

Pruebas

A	B	salida
[1, 2, 3, 10, 100]	[1, 2, 3, 4, 5, 6]	1 2 3
[1, 2, 3, 10, 100]	[5, 2, 3, 10, 13]	2 3 10
[1, 2, 3, 10, 100]	[5, 6]	

```
import Foundation
var arregloA = [1,2,3,10,100]
var arregloB = [1,2,3,4,5,6]
for x in arregloA {
    if(arregloB.contains(x)){
        print(x)
    }
}
```

Execute | Share | main.swift | STDIN | Result

```
1
2 import Foundation
3 var arregloA = [1,2,3,10,100]
4 var arregloB = [1,2,3,4,5,6]
5 for x in arregloA {
```

\$swift main.swift

1
2
3

Execute | Share | main.swift | STDIN | Result

```
1
2 import Foundation
3 var arregloA = [1,2,3,10,100]
4 var arregloB = [5,2,3,10,13]
5 for x in arregloA {
```

\$swift main.swift

2
3
10

2.3.28. Extraiga cada dígito, de izquierda a derecha, de un número dado en n y almacénelo en un vector. Resuélvalo aritméticamente, es decir, sin convertir n a cadena de caracteres.

Pruebas

n	vector
12345	[1, 2, 3, 4, 5]
20143831	[2, 0, 1, 4, 3, 8, 3, 1]

```
import Foundation
var n = 20143831;
var vector: [Int] = Array()
var maximoDivisible = 1;
while((n / maximoDivisible) > 0){
    maximoDivisible*=10
}
maximoDivisible/=10
while(n > 0){
    vector.append((n / maximoDivisible))
    n-=(n / maximoDivisible)*maximoDivisible
    maximoDivisible/=10
}
print(vector)
```

Execute | Share main.swift STDIN Result

```
1 import Foundation
2 var n = 20143831;
3 var vector: [Int] = Array()
```

```
$swift main.swift
[2, 0, 1, 4, 3, 8, 3, 1]
```

Execute | Share main.swift STDIN Result

```
1 import Foundation
2 var n = 12345;
3 var vector: [Int] = Array()
4 var maximoDivisible = 1;
5 while((n / maximoDivisible) > 0){
```

```
$swift main.swift
[1, 2, 3, 4, 5]
```

2.3.29. Dadas las matrices

$$A = \begin{pmatrix} 2 & 0 & 1 \\ 3 & 0 & 0 \\ 5 & 1 & 1 \end{pmatrix}; \quad B = \begin{pmatrix} 1 & 0 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 0 \end{pmatrix}$$

Calcular:

a) $A + B$

$$A + B = \begin{pmatrix} 2 & 0 & 1 \\ 3 & 0 & 0 \\ 5 & 1 & 1 \end{pmatrix} + \begin{pmatrix} 1 & 0 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 2+1 & 0+0 & 1+1 \\ 3+1 & 0+2 & 0+1 \\ 5+1 & 1+1 & 1+0 \end{pmatrix} = \begin{pmatrix} 3 & 0 & 2 \\ 4 & 2 & 1 \\ 6 & 2 & 1 \end{pmatrix}$$

```
import Foundation
var arregloA = [[2,0,1], [3,0,0], [5,1,1]]
var arregloB = [[1,0,1], [1,2,1], [1,1,0]]
var arregloResultante: [[Int]] = Array()
for x in 0...(arregloA.count-1) {
    var arregloTemporal: [Int] = Array()
    for y in 0...(arregloA[x].count-1){
        arregloTemporal.append( arregloA[x][y] + arregloB[x][y])
    }
    arregloResultante.append(arregloTemporal)
}
print(arregloResultante)
```

The screenshot shows the Xcode interface with two main panes: 'Execute' and 'Result'. In the 'Execute' pane, the Swift code for matrix addition is pasted. In the 'Result' pane, the command '\$swift main.swift' is run, followed by the output: '[[3, 0, 2], [4, 2, 1], [6, 2, 1]]', which matches the calculated result from the problem statement.

```
1 import Foundation
2 var arregloA = [[2,0,1], [3,0,0], [5
,1,1]]
3 var arregloB = [[1,0,1], [1,2,1], [1
,1,0]]
4 var arregloResultante: [[Int]] =
```

\$swift main.swift
[[3, 0, 2], [4, 2, 1], [6, 2, 1]]

b) $A - B$

$$A - B = \begin{pmatrix} 2 & 0 & 1 \\ 3 & 0 & 0 \\ 5 & 1 & 1 \end{pmatrix} - \begin{pmatrix} 1 & 0 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 2-1 & 0-0 & 1-1 \\ 3-1 & 0-2 & 0-1 \\ 5-1 & 1-1 & 1-0 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 2 & -2 & -1 \\ 4 & 0 & 1 \end{pmatrix}$$

```
import Foundation

var arregloA = [[2,0,1], [3,0,0], [5,1,1]]
var arregloB = [[1,0,1], [1,2,1], [1,1,0]]

var arregloResultante: [[Int]] = Array()

for x in 0...(arregloA.count-1) {
    var arregloTemporal: [Int] = Array()
    for y in 0...(arregloA[x].count-1){
        arregloTemporal.append( arregloA[x][y] - arregloB[x][y])
    }
    arregloResultante.append(arregloTemporal)
}

print(arregloResultante)
```

The screenshot shows a terminal window with the following content:

```
Execute | Share main.swift STDIN
1 import Foundation
2
3
4 var arregloA = [[2,0,1], [3,0,0], [5
,1,1]]
5 var arregloB = [[1,0,1], [1,2,1], [1
,1,0]]
$swift main.swift
[[1, 0, 0], [2, -2, -1], [4, 0, 1]]
```

c) A x B

$$A \cdot B = \begin{pmatrix} 2 & 0 & 1 \\ 3 & 0 & 0 \\ 5 & 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 2 \cdot 1 + 0 \cdot 1 + 1 \cdot 1 & 2 \cdot 0 + 0 \cdot 2 + 1 \cdot 1 & 2 \cdot 1 + 0 \cdot 1 + 1 \cdot 0 \\ 3 \cdot 1 + 0 \cdot 1 + 0 \cdot 1 & 3 \cdot 0 + 0 \cdot 2 + 0 \cdot 1 & 3 \cdot 1 + 0 \cdot 1 + 0 \cdot 0 \\ 5 \cdot 1 + 1 \cdot 1 + 1 \cdot 1 & 5 \cdot 0 + 1 \cdot 2 + 1 \cdot 1 & 5 \cdot 1 + 1 \cdot 1 + 1 \cdot 0 \end{pmatrix} = \begin{pmatrix} 3 & 1 & 2 \\ 3 & 0 & 3 \\ 7 & 3 & 6 \end{pmatrix}$$

```
import Foundation

var arregloA = [[2,0,1], [3,0,0], [5,1,1]]
var arregloB = [[1,0,1], [1,2,1], [1,1,0]]
var arregloC:[[Int]] = Array()

for y in 0...arregloA.count-1{
    //print("y: " + String(y))
    var arregloAuxiliar:[Int] = Array()
    for x in 0...arregloA.count-1{
        var resultado = 0;
        for z in 0...arregloA.count-1{
            //print(String(arregloA[y][z]))
            // print(String(arregloB[z][x]))

            resultado += (arregloA[y][z] * arregloB[z][x])
        }
        //    print("--")
        arregloAuxiliar.append(resultado)

    }
    arregloC.append(arregloAuxiliar)
}

print(arregloC)
```

```
Execute | Share | main.swift | STDIN | Result
$swift main.swift
[[3, 1, 2], [3, 0, 3], [7, 3, 6]]
```

```
2
3 var arregloA = [[2,0,1], [3,0,0],
4     [5,1,1]]
5 var arregloB = [[1,0,1], [1,2,1],
6     [1,1,0]]
7 var arregloC:[[Int]] = Array()
```

2.3.30. Dada una matriz cuadrada A , almacene los elementos de la diagonal principal y los de la diagonal inversa, en vectores llamados DP y DI respectivamente.

Pruebas

A	DP	DI
3 5 8 2	3	2
2 7 9 5	7	9
2 8 9 2	9	8
4 6 7 1	1	4

```
import Foundation
var A = [[3,5,8,2], [2,7,9,5], [2,8,9,2], [4,6,7,1]]
var DP: [Int] = Array()
var DI: [Int] = Array()
for x in 1...A.count {
    DP.append(A[x-1][x-1])
    DI.append(A[x-1][A.count-x])
}
print(DP)
print(DI)
```

Execute | Share | main.swift | STDIN | Result

```
1 import Foundation
2 var A = [[3,5,8,2], [2,7,9,5], [2,8,9,2], [4,6,7,1]]
3 var DP: [Int] = Array()
4 var DI: [Int] = Array()
```

\$swift main.swift
[3, 7, 9, 1]
[2, 9, 8, 4]

- 2.3.31.** Dada una matriz cuadrada A , imprima el resultado de sumar los elementos que no corresponden a la periferia de la matriz.

Pruebas

A

3	5	8	9	2
1	4	2	1	0
4	5	4	8	1
9	8	1	0	3
7	2	1	1	3

Suma=33

```
import Foundation

var A = [[3,5,8,9, 2], [1,4,2,1,0] , [4,5,4,8,1], [9,8,1,0,3], [7,2,1,1,3]]
var suma = 0

for y in 0...A.count-1 {
    for x in 0...A.count-1 {
        if(y != 0 && x != 0 && x != A.count-1 && y != A.count-1){
            suma+=A[y][x]
        }
    }
}
print(suma)
```

Execute | Share | main.swift | STDIN | Result

```
1 import Foundation
2
3 var A = [[3,5,8,9, 2], [1,4,2,1,0]
, [4,5,4,8,1], [9,8,1,0,3], [7,2
,1,1,3]]
4 var suma = 0
5
$swift main.swift
33
```

2.3.32. Escriba un método que reciba dos valores enteros y devuelva el mayor de ellos. Utilice el `_` para ignorar el nombre externo de cada parámetro.

Pruebas

a	b	mayor
2	5	5
8	3	8

```
import Foundation
func obtenerMayor(_ a : Int , _ b :Int) -> Int {
    return (a > b ? a : b)
}
print(obtenerMayor(2,5))
```

The screenshot shows the Xcode interface with the code for the first test case. The code defines a function `obtenerMayor` that takes two integers and returns the greater one. It then prints the result of calling this function with arguments 2 and 5. The "Execute" tab is selected in the top bar.

```
1 import Foundation
2 func obtenerMayor(_ a : Int , _ b :Int) -> Int {
3     return (a > b ? a : b)
4 }
5 print(obtenerMayor(2,5))
```

Result:

```
$swift main.swift
5
```

The screenshot shows the Xcode interface with the code for the second test case. The code is identical to the first one, but the arguments passed to the `print` statement are 8 and 3. The "Execute" tab is selected in the top bar.

```
1 import Foundation
2 func obtenerMayor(_ a : Int , _ b :Int) -> Int {
3     return (a > b ? a : b)
4 }
5 print(obtenerMayor(8,3))
```

Result:

```
$swift main.swift
8
```

2.3.33. Escriba un método que reciba un número entero y devuelva un vector con los valores de 1 a N .

Pruebas

N	vector
8	[1, 2, 3, 4, 5, 6, 7, 8]
3	[1, 2, 3]

```
import Foundation
func crearVector(_ n : Int) -> [Int] {
    var vector: [Int] =  Array()
    for x in 1...n{
        vector.append(x)
    }
    return vector
}
print(crearVector(8))
```

Execute | Share main.swift STDIN

```
1 import Foundation
2 func crearVector(_ n : Int) -> [Int]
3 {
4     var vector: [Int] =  Array()
5     for x in 1...n{
6         vector.append(x)
7     }
8     return vector
9 }
print(crearVector(3))
```

Result

```
$swift main.swift
[1, 2, 3]
```

Execute | Share main.swift STDIN

```
1 import Foundation
2 func crearVector(_ n : Int) -> [Int]
3 {
4     var vector: [Int] =  Array()
5     for x in 1...n{
6         vector.append(x)
7     }
8     return vector
9 }
print(crearVector(8))
```

Result

```
$swift main.swift
[1, 2, 3, 4, 5, 6, 7, 8]
```

2.3.34. Programe un método que reciba cuatro parámetros: hora inicial, minutos iniciales, hora final y minutos finales. La función debe calcular los minutos transcurridos entre los dos horarios compuestos de horas y minutos. Si los minutos se omiten en la invocación, se considerará que el valor es cero.

Pruebas

horaInicial	minutoInicial	horaFinal	minutoFinal	diferencia
12	3	13	10	67
8	10	17	30	560

```
import Foundation
func obtenerDiferenciaEnMinutos(horaInicial : Int, minutoInicial: Int, horaFinal: Int, minutoFinal: Int) -> Int {
    var minutosIniciales = (horaInicial * 60) + minutoInicial
    var minutosFinales = (horaFinal * 60) + minutoFinal
    return (minutosFinales - minutosIniciales)
}
print(obtenerDiferenciaEnMinutos(horaInicial: 12, minutoInicial: 3, horaFinal: 13, minutoFinal: 10))
```

Run 5.1-RELEASE ▾

Download ▾

```
1 import Foundation
2 func obtenerDiferenciaEnMinutos(horaInicial : Int, minutoInicial: Int, horaFinal: Int, minutoFinal: Int)
3     var minutosIniciales = (horaInicial * 60) + minutoInicial
4     var minutosFinales = (horaFinal * 60) + minutoFinal
5     return (minutosFinales - minutosIniciales)
6 }
7 print(obtenerDiferenciaEnMinutos(horaInicial: 8, minutoInicial: 10, horaFinal: 17, minutoFinal: 30))
```

560

Run 5.1-RELEASE ▾

Download ▾

```
1 import Foundation
2 func obtenerDiferenciaEnMinutos(horaInicial : Int, minutoInicial: Int, horaFinal: Int, minutoFinal: Int)
3     var minutosIniciales = (horaInicial * 60) + minutoInicial
4     var minutosFinales = (horaFinal * 60) + minutoFinal
5     return (minutosFinales - minutosIniciales)
6 }
7 print(obtenerDiferenciaEnMinutos(horaInicial: 12, minutoInicial: 3, horaFinal: 13, minutoFinal: 10))
```

67

2.3.35. Una pila es una estructura de datos (arreglo unidimensional), que puede realizar lo siguiente:

- Agregar un elemento al principio del arreglo
- Obtener el valor del primer elemento del arreglo
- Entregar el primer elemento del arreglo y removerlo (sacarlo de la pila)

Debe crear un método para cada operación descrita anteriormente, mismas que deberán trabajar sobre un arreglo pasado como parámetro por referencia.

```
import Foundation

var pila : [Int] = Array()
func agregarElemento( pila: inout [Int], item : Int){
    pila.insert(item, at:0)
}
func obtenerElemento( pila: inout [Int]) -> Int{
    return pila[0]
}
func obtenerYRemoverElemento( pila: inout [Int]) -> Int{
    var v = pila[0]
    pila.remove(at: 0)
    return v
}
agregarElemento( pila:&pila, item: 2)
agregarElemento( pila:&pila, item: 3)
agregarElemento( pila:&pila, item: 4)
print(pila)
print(obtenerElemento( pila:&pila))
print(obtenerYRemoverElemento( pila:&pila))
print(pila)
```

Execute | Share | main.swift | STDIN

```
1 import Foundation
2
3
4 var pila : [Int] = Array()
5 func agregarElemento( pila: inout
6     [Int], item : Int){
7     pila.insert(item, at:0)
8 }
9
10 func obtenerElemento( pila: inout
11     [Int]) -> Int{
12     return pila[0]
13 }
14
15 func obtenerYRemoverElemento( pila:
16     inout [Int]) -> Int{
17     let v = pila[0]
18     pila.remove(at: 0)
19     return v
20 }
21
22 agregarElemento( pila:&pila, item: 2
23 )
24 agregarElemento( pila:&pila, item: 3
25 )
26 agregarElemento( pila:&pila, item: 4
27 )
28 print(pila)
29 print(obtenerElemento( pila:&pila))
30 print(obtenerYRemoverElemento( pila
31     :&pila))
32 print(pila)
```

Result

```
$swift main.swift
[4, 3, 2]
4
4
[3, 2]
```

2.3.36. Escriba los métodos para agregar y entregar un elemento bajo el concepto de la estructura de dato llamada cola. Utilice el arreglo como parámetro por referencia.

```
import Foundation
var fila : [Int] = Array()
func agregarElemento( fila: inout [Int], item : Int){
    fila.append(item)
}
func obtenerElemento( fila: inout [Int]) -> Int{
    return fila[0]
}
func obtenerYRemoverElemento( fila: inout [Int]) -> Int{
    var v = fila[0]
    fila.remove(at: 0)
    return v
}
agregarElemento( fila:&fila, item: 2)
agregarElemento( fila:&fila, item: 3)
agregarElemento( fila:&fila, item: 4)
print(fila)
print(obtenerElemento( fila:&fila))
print(obtenerYRemoverElemento( fila:&fila))
print(fila)
```

```
Execute | Share | main.swift | STDIN | Result
1 import Foundation
2 var fila : [Int] = Array()
3 func agregarElemento( fila: inout
4     [Int], item : Int){
5     fila.append(item)
6 }
7 func obtenerElemento( fila: inout
8     [Int]) -> Int{
9     return fila[0]
10 }
11 func obtenerYRemoverElemento( fila:
12     inout [Int]) -> Int{
13     let v = fila[0]
14     fila.remove(at: 0)
15     return v
16 }
17 agregarElemento( fila:&fila, item: 2
18 )
19 agregarElemento( fila:&fila, item: 3
20 )
21 agregarElemento( fila:&fila, item: 4
22 )
23 print(fila)
24 print(obtenerElemento( fila:&fila))
25 print(obtenerYRemoverElemento( fila
26 :&fila))
27 print(fila)
```

```
$swift main.swift
[2, 3, 4]
2
2
[3, 4]
```

2.3.37. Escriba un método que calcule la factorial de un número dado como parámetro. Utilice recursividad.

Pruebas

n	factorial
3	6
5	120
10	3628800

```
import Foundation
func calcularFactorial(n : Int) -> Int{
    if(n == 1){
        return 1
    }
    else{
        var resultado = n * calcularFactorial(n: n-1)
        return resultado
    }
}
print(calcularFactorial(n: 10))
```

The screenshot shows a code editor interface with two panes. The left pane contains the Swift code for calculating a factorial. The right pane shows the execution results.

Left Pane (Code):

```
1 import Foundation
2 func calcularFactorial(n : Int) ->
3     Int{
4     if(n == 1){
5         return 1
6     }
7     let resultado = n *
8         calcularFactorial(n: n-1
9     )
10    return resultado
11 }
12 print(calcularFactorial(n: 10))
```

Right Pane (Result):

```
$swift main.swift
3628800
```


2.3.38. Programe un método recursivo que calcule la multiplicación de dos números dados como parámetros por referencia (A y B). Deberá utilizar el método ruso para multiplicar, el cual (en papel) consiste en escribir una serie de pares números bajo dos columnas. La primera columna la encabeza el valor A y la segunda columna la encabeza el valor B. El valor A se divide entre dos y debajo se coloca el cociente entero de la división (ignorando el residuo), repetir sucesivamente hasta llegar al resultado 1. Por cada vez que se divide la columna A, deberá multiplicar el valor de la columna B por dos. Al final se suman todos los valores bajo la columna B, que se encuentren al nivel de un numero impar de la columna A. Cree e invoque una función booleana de apoyo, que determine si un numero es par. Caso de ejemplo: $27 \times 82 = 2214$

A	B	sumandos
27	82	82
13	164	164
6	328	
3	656	656
1	1312	1312

$82 + 164 + 656 + 1312 = 2214$

```
import Foundation

var A = 27
var B = 82

func esPar(numero: Int) -> Bool{
    return (numero % 2 == 0)
}

func multiplicarPorMetodoRuso(multiplicador: inout Int, multiplicando: inout Int) -> Int {
    if(multiplicador == 0){
        return 0;
    }else{
        var resultado = 0;
        if !esPar(numero: multiplicador) {
            resultado+=multiplicando
        }
        multiplicador/=2
        multiplicando*=2
        return multiplicarPorMetodoRuso(multiplicador: &multiplicador, multiplicando: &multiplicando) + resultado
    }
}
print(multiplicarPorMetodoRuso(multiplicador: &A , multiplicando: &B))

print(multiplicarPorMetodoRuso(multiplicador:&A, multiplicando:&B))
```

```
Execute | Share main.swift STDIN Result
$swift main.swift
2214
```

2.3.39. Escriba una función que reciba tres valores enteros y devuelva al mismo tiempo el mayor y el menor de ellos. Utilice el `_` para ignorar el nombre externo de cada parámetro.

Pruebas

a	b	c	mayor	menor
2	5	8	8	2
6	3	6	6	3

```
import Foundation

func obtenerMenorYMayor(_ a : Int, _ b : Int, _ c : Int) -> (Int, Int){
    var listado: [Int] = Array()
    listado.append(a)
    listado.append(b)
    listado.append(c)
    listado.sort()
    return (listado[0], listado[2])
}

var resultado = obtenerMenorYMayor(2,5,8)
print("Valor menor: \(String(resultado.0)), valor mayor : \(String(resultado.1))")
```

The screenshot shows a code editor interface with two panes. The left pane is titled "Execute" and contains the Swift code provided above. The right pane is titled "Result" and shows the output of running the code in the terminal: \$swift main.swift followed by the printed output "Valor menor: 2, valor mayor : 8".

```
1 import Foundation
2
3 func obtenerMenorYMayor(_ a : Int, _ b : Int, _ c : Int) -> (Int, Int){
4     var listado: [Int] = Array()
5     listado.append(a)
6     listado.append(b)
7     listado.append(c)
8     listado.sort()
9     return (listado[0], listado[2])
10}
11
12 var resultado = obtenerMenorYMayor(2,5,8)
13 print("Valor menor: \(String(resultado.0)), valor mayor : \(String(resultado.1))")
```

\$swift main.swift
Valor menor: 2, valor mayor : 8

2.3.40. Suponga que se encuentra programando un juego en el cual el personaje camina en una cuadrícula, es decir, puede desplazarse un paso a la vez, ya sea hacia la derecha, izquierda, arriba o abajo. La posición inicial siempre es $(0, 0)$ y mediante un arreglo que contenga los pasos, debe calcular la posición final del personaje después de dar dichos pasos. Utilice una enumeración para definir cada una de las cuatro direcciones.

Pruebas

pasos	posición
arriba, arriba, izquierda, abajo, izquierda	(-2, 1)
arriba, arriba, izquierda, abajo, izquierda, abajo, abajo, derecha, derecha, abajo, derecha	(1, -2)
<i>* Comience en (5, 2) arriba, derecha, arriba, derecha, arriba, derecha, abajo, derecha</i>	(9, 4)

```

import Foundation
enum Paso {
    case arriba
    case abajo
    case izquierda
    case derecha
}
var pasos: [Paso] = Array()
func calcularPosicion(_ pasos: [Paso]) -> (Int, Int){
    var posicionActualX = 0;
    var posicionActualY = 0;
    for pasoActual in pasos{
        switch pasoActual {
            case .arriba:
                posicionActualY+=1
            case .abajo:
                posicionActualY-=1
            case .derecha:
                posicionActualX+=1
            case .izquierda:
                posicionActualX-=1
        }
    }

    return (x: posicionActualX, y: posicionActualY)
}
pasos.append(contentsOf: [Paso.arriba, Paso.arriba, Paso.izquierda, Paso.abajo, Paso.izquier
print(calcularPosicion(pasos))
pasos.removeAll()
pasos.append(contentsOf: [Paso.arriba, Paso.arriba, Paso.izquierda, Paso.abajo, Paso.izquier
print(calcularPosicion(pasos))

```

```
Execute | Share | main.swift | STDIN | Result  
2 enum Paso {  
3     case arriba  
4     case abajo  
5     case izquierda  
6     case derecha  
7 }  
8 var pasos: [Paso] = Array()  
9 func calcularPosicion(_ pasos: [Paso]) -> (Int, Int){  
10    var posicionActualX = 0;  
11    var posicionActualY = 0;  
12    for pasoActual in pasos{  
13        switch pasoActual {  
14            case .arriba:  
15                posicionActualY+=1  
16            case .abajo:  
17                posicionActualY-=1  
18            case .derecha:  
19                posicionActualX+=1  
20            case .izquierda:  
21                posicionActualX-=1  
22        }  
23    }  
24  
25    return (x: posicionActualX, y: posicionActualY)  
26 }  
27 pasos.append(contentsOf: [Paso.arriba, Paso.arriba,  
28                     Paso.izquierda, Paso.abajo, Paso.izquierda ])  
29 print(calcularPosicion(pasos))  
30 pasos.removeAll()  
31 pasos.append(contentsOf: [Paso.arriba, Paso.arriba,  
28                     Paso.izquierda, Paso.abajo, Paso.izquierda, Paso  
29                     .abajo, Paso.abajo, Paso.derecha, Paso.derecha,  
30                     Paso.abajo, Paso.derecha ])  
31 print(calcularPosicion(pasos))
```

\$swift main.swift
(-2, 1)
(1, -2)

- 2.3.41.** 1) Defina una enumeración con tres miembros: *piedra*, *papel*, *tijera*.
 2) Defina otra enumeración con tres miembros: *gana*, *pierde*, *empata*.
 3) Escriba una función llamada *juego* que reciba lo que “elige” cada uno de los jugadores para jugar y que devuelva la respuesta correspondiente (de la segunda enumeración), respecto al primer jugador. Notas: Siempre juegan dos jugadores. No utilice valoraciones numéricas.

Pruebas

juego	resultado
piedra, tijera	gana
piedra, papel	pierde
tijera, tijera	empata

```
import Foundation

enum Opcion {
    case piedra
    case papel
    case tijera
}
enum Resultado {
    case gana
    case pierde
    case empata
}
func jugar(_ opcion1: Opcion, _ opcion2 : Opcion) -> Resultado{
    if(opcion1 == opcion2){
        return Resultado.empata
    }
    switch opcion1{
        case .piedra:
            if(opcion2 == Opcion.papel){
                return Resultado.pierde
            }
        case .papel:
            if(opcion2 == Opcion.tijera){
                return Resultado.pierde
            }
        case .tijera:
            if(opcion2 == Opcion.piedra){
                return Resultado.pierde
            }
    }
    return Resultado.gana
}
print(jugar(Opcion.piedra, Opcion.tijera))
print(jugar(Opcion.piedra, Opcion.papel))
print(jugar(Opcion.tijera, Opcion.tijera))
```

The screenshot shows a code editor interface with two panes. The left pane is titled "Execute" and contains the Swift code for a game logic program. The right pane is titled "Result" and shows the output of running the code.

```
2
3 enum Opcion {
4     case piedra
5     case papel
6     case tijera
7 }
8 enum Resultado {
9     case gana
10    case pierde
11    case empata
12 }
13 func jugar(_ opcion1: Opcion, _ opcion2 : Opcion) ->
14     Resultado{
15     if(opcion1 == opcion2){
16         return Resultado.empata
17     }
18     switch opcion1{
19     case .piedra:
20         if(opcion2 == Opcion.papel){
21             return Resultado.pierde
22         }
23     case .papel:
24         if(opcion2 == Opcion.tijera){
25             return Resultado.pierde
26         }
27     case .tijera:
28         if(opcion2 == Opcion.piedra){
29             return Resultado.pierde
30         }
31     return Resultado.gana
32 }
33 print(jugar(Opcion.piedra, Opcion.tijera))
34 print(jugar(Opcion.piedra, Opcion.papel))
35 print(jugar(Opcion.tijera, Opcion.tijera))
```

\$swift main.swift
gana
pierde
empata

2.3.42. Tiene un arreglo de diccionarios. Cada diccionario tiene un par de valores, uno con el apellido y otro con el nombre de una persona. Debe entregar un arreglo que contenga cada nombre completo.

Pruebas

diccionarios	resultado
[{"apellido": "Perea", "nombre": "Alicia"}, {"apellido": "Flores", "nombre": "Norma"}, {"apellido": "Noriega", "nombre": "Roberto"}]	["Alicia Perea", "Norma Flores", "Roberto Noriega"]

```
import Foundation

var arregloDeDiccionarios : [Dictionary<String, String>] = [ ["apellido" : "Perea", "nombre": "Alicia"],
[ "apellido" : "Flores", "nombre": "Norma"], [ "apellido" : "Noriega", "nombre": "Roberto" ] ]
var nombresCompletos : [String] = Array()

for arregloActual in arregloDeDiccionarios{
    nombresCompletos.append("\(arregloActual["nombre"]!) \( arregloActual["apellido"]! )")
}

print(nombresCompletos)
```

The screenshot shows a code editor interface with two panes. The left pane is titled 'Execute' and contains the provided Swift code. The right pane is titled 'Result' and shows the output of running the code in the terminal: \$swift main.swift followed by the array of names: ["Alicia Perea", "Norma Flores", "Roberto Noriega"].

```
Execute | Share | main.swift | STDIN
1 import Foundation
2
3 var arregloDeDiccionarios : [Dictionary<String, String>] = [ ["apellido" : "Perea", "nombre": "Alicia"],
4 [ "apellido" : "Flores", "nombre": "Norma"], [ "apellido" : "Noriega", "nombre": "Roberto" ] ]
5 var nombresCompletos : [String] = Array()
6
7
8 for arregloActual in arregloDeDiccionarios{
9     nombresCompletos.append("\(\
10        arregloActual["nombre"]!) \(\
11        arregloActual["apellido"]! )")
12 }
13 print(nombresCompletos)

Result
$swift main.swift
["Alicia Perea", "Norma Flores", "Roberto Noriega"]
```

2.3.43. Se tienen dos tuplas, cada una representa el numerador y el denominador de una fracción (en ese orden dentro de la tupla). Escriba una función que reciba ambas tuplas y que devuelva otra con la fracción resultante. Ejemplo: f1 = (34, 3); f2 = (11, 2); suma = (101, 6). Simplifique el resultado (cuando aplique)

$$\frac{a \cancel{\times} c}{b \cancel{\times} d} = \frac{(a \times d) + (b \times c)}{b \times d}$$

Pruebas

fracción 1	fracción 2	suma
5 / 8	17 / 9	181 / 72
1 / 5	1 / 6	11 / 30
3 / 12	3 / 6	3 / 4

```
import Foundation

var fraccionA: (Int, Int) = (numerador: 5, denominador: 8)
var fraccionB: (Int, Int) = (numerador: 17, denominador: 9)

func sumarFracciones (_ fraccionA : (numerador: Int, denominador: Int), _ fraccionB : (numerador: Int, denominador: Int)) -> (Int, Int){
    var numerador = (fraccionA.numerador * fraccionB.denominador) + (fraccionA.denominador * fraccionB.numerador)
    var denominador = (fraccionA.denominador * fraccionB.denominador)
    return(numerador , denominador)
}

var resultado = sumarFracciones(fraccionA, fraccionB)
func simplificar(_ n: Int, _ d: Int) -> (Int, Int) {
    var divisor = 2, numerador = n, denominador = d
    while((divisor <= numerador) && (divisor <= denominador)) {
        while((numerador % divisor == 0) && (denominador % divisor == 0)) {
            numerador = numerador / divisor
            denominador = denominador / divisor
        }
        divisor += 1
    }
    return (numerador, denominador)
}
print("\(simplificar(resultado.0, resultado.1).0) / \(simplificar(resultado.0, resultado.1).1) ")
```

Execute | Share main.swift STDIN Result

```
1 import Foundation
2
3 var fraccionA: (Int, Int) = (numerador: 5,
4     denominador: 8)
5 var fraccionB: (Int, Int) = (numerador: 17,
6     denominador: 9)
```

\$swift main.swift
181 / 72

Execute | Share main.swift STDIN Result

```
1 import Foundation
2
3 var fraccionA: (Int, Int) = (numerador: 1,
4     denominador: 5)
5 var fraccionB: (Int, Int) = (numerador: 1,
6     denominador: 6)
```

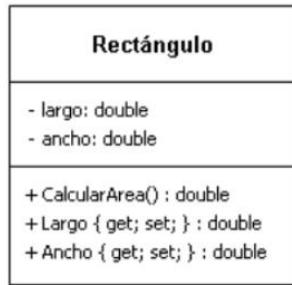
\$swift main.swift
11 / 30

Execute | Share main.swift STDIN Result

```
2
3 var fraccionA: (Int, Int) = (numerador: 3,
4     denominador: 12)
5 var fraccionB: (Int, Int) = (numerador: 3,
6     denominador: 6)
```

\$swift main.swift
3 / 4

2.3.44. Hay una pared rectangular que tiene una ventana rectangular; se requiere un programa que dadas las dimensiones (largo y ancho) de la ventana y de la pared, imprima los minutos necesarios para pintar la pared, sabiendo que se puede pintar 1 m² en 10 minutos. Utilice la siguiente clase:



Pruebas

pared		ventana		resultado
largo	ancho	largo	ancho	
10	5	2	1	480
3	3	1	1	80

```
import Foundation

class Rectangulo{

    private var largo: Double = 0
    private var ancho: Double = 0

    public var Largo: Double {get{return largo} set{largo = newValue}}
    public var Ancho: Double {get{return ancho} set{ancho = newValue}}

    init(){}
    func CalcularArea()-> Double{
        return largo * ancho
    }
}

var pared = Rectangulo()
pared.Largo = 10
pared.Ancho = 5

var ventana = Rectangulo()
ventana.Largo = 2
ventana.Ancho = 1

var m2Totales = pared.CalcularArea() - ventana.CalcularArea()

print(m2Totales * 10)
```

```
19 var pared = Rectangulo()  
20 pared.Largo = 10  
pared.Ancho = 5  
22  
23 var ventana = Rectangulo()  
ventana.Largo = 2  
ventana.Ancho = 1  
26  
27 var m2Totales = pared.CalcularArea() - ventana.CalcularArea()  
28  
29  
30 print(m2Totales * 10)  
31  
32
```

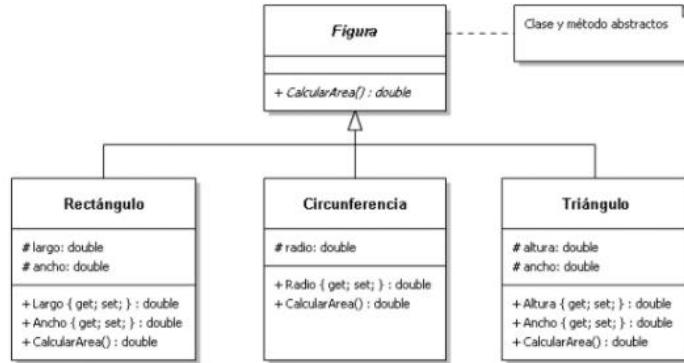
480.0

```
19 var pared = Rectangulo()  
pared.Largo = 3  
pared.Ancho = 3  
22  
23 var ventana = Rectangulo()  
ventana.Largo = 1  
ventana.Ancho = 1  
26  
27 var m2Totales = pared.CalcularArea() - ventana.CalcularArea()  
28  
29  
30 print(m2Totales * 10)  
31  
32
```

80.0

Se crea la clase rectángulo, para posteriormente hacer las instancias de pared y ventana, seteando así también sus respectivos valores de largo y ancho, para posteriormente usar el método calcular área y sacar la diferencia entre estos dos, una vez obtenida la diferencia, se procede a imprimir la diferencia por diez para que así tengamos el resultado esperado de tiempo en minutos.

2.3.45. Diseñe un programa que calcule el área de las figuras geométricas señaladas en el diagrama UML que se muestra a continuación. Utilice polimorfismo para la reutilización de métodos abstractos.



Pruebas

rectángulo			circunferencia		triángulo		
largo	ancho	área	radio	área	altura	ancho	área
6	8	48	14	615.75	4	3	6
15	4	60	8	201.06	12	10	60

```

import Foundation

protocol Figura{
    func CalcularArea()-> Double
}

class Rectangulo: Figura{
    private var largo: Double = 0
    private var ancho: Double = 0
    public var Largo: Double {get{return largo} set{largo = newValue}}
    public var Ancho: Double {get{return ancho} set{ancho = newValue}}
    init(){}
    func CalcularArea()-> Double{
        return largo * ancho
    }
}
class Circunferencia: Figura{
    private var radio: Double = 0
    public var Radio: Double {get{return radio} set{radio = newValue}}
    init(){}
    func CalcularArea()-> Double{
        return (radio * radio) * 3.1416
    }
}

class Triangulo: Figura{
    private var altura: Double = 0
    private var ancho: Double = 0
    public var Altura: Double {get{return altura} set{altura = newValue}}
    public var Ancho: Double {get{return ancho} set{ancho = newValue}}
    init(){}
    func CalcularArea()-> Double{
        return (altura * ancho) / 2
    }
}
var rectangulo = Rectangulo()
rectangulo.Largo = 6
rectangulo.Ancho = 8
print(rectangulo.CalcularArea())
var circunferencia = Circunferencia()
circunferencia.Radio = 14
print(circunferencia.CalcularArea())
var triangulo = Triangulo()
triangulo.Altura = 4
triangulo.Ancho = 3
print(triangulo.CalcularArea())

```

```
53  
54     var rectangulo = Rectangulo()  
55     rectangulo.Largo = 6  
56     rectangulo.Ancho = 8  
57     print(rectangulo.CalcularArea())  
58     var circunferencia = Circunferencia()  
59     circunferencia.Radio = 14  
60     print(circunferencia.CalcularArea())  
61     var triangulo = Triangulo()  
62     triangulo.Altura = 4  
63     triangulo.Ancho = 3  
64     print(triangulo.CalcularArea())  
65  
66  
67  
  
48.0  
615.7536  
6.0
```

```
53  
54     var rectangulo = Rectangulo()  
55     rectangulo.Largo = 15  
56     rectangulo.Ancho = 4  
57     print(rectangulo.CalcularArea())  
58     var circunferencia = Circunferencia()  
59     circunferencia.Radio = 8  
60     print(circunferencia.CalcularArea())  
61     var triangulo = Triangulo()  
62     triangulo.Altura = 12  
63     triangulo.Ancho = 10  
64     print(triangulo.CalcularArea())  
65  
66  
67  
  
60.0  
201.0624  
60.0
```

2.3.46. Una agencia vendedora de autos desea administrar los datos de sus vehículos y clasificarlos por tipo. Todos los autos tienen los siguientes datos:

- Número de serie del motor
- Marca
- Año
- Precio

Los vehículos se clasifican en autos compactos, autos de lujo, camionetas y vagonetas. Para los autos y vagonetas, también es importante almacenar la cantidad de pasajeros; mientras que para las camionetas se debe controlar la capacidad de carga en kgs. y la cantidad de ejes y de rodadas. Modele este sistema e instancie cada una de las clases, asignándole datos mediante sus respectivas propiedades. Agregue un constructor con parámetros a cada clase para inicializar sus datos e invoque el constructor de la clase base desde el constructor de cada clase derivada.

```

class Vehiculo{
    var NumeroDeSerieMotor : String = ""
    var Marca: String = ""
    var Año: Int = 0
    var Precio : Double = 0
    init(_ numeroDeSerieMotor:String, _ marca:String,_ año:Int,_ precio:Double){
        self.NumeroDeSerieMotor = numeroDeSerieMotor
        self.Marca = marca
        self.Año = año
        self.Precio = precio
    }
}
class Auto : Vehiculo{
    var CantidadDePasajeros : Int = 0
    init(_ numeroDeSerieMotor:String, _ marca:String,_ año:Int,_ precio:Double, _ cantidadDePasajeros: Int){
        self.CantidadDePasajeros = cantidadDePasajeros
        super.init(numeroDeSerieMotor, marca, año, precio)
    }
}
class AutoCompacto : Auto{
    override init(_ numeroDeSerieMotor:String, _ marca:String,_ año:Int,_ precio:Double, _ cantidadDePasajeros: Int){
        super.init(numeroDeSerieMotor, marca, año, precio, cantidadDePasajeros)
    }
}
class AutoDeLujo: Auto{
    override init(_ numeroDeSerieMotor:String, _ marca:String,_ año:Int,_ precio:Double, _ cantidadDePasajeros: Int){
        super.init(numeroDeSerieMotor, marca, año, precio, cantidadDePasajeros)
    }
}
class Vagoneta: Vehiculo{
    var CantidadDePasajeros : Int = 0
    init(_ numeroDeSerieMotor:String, _ marca:String,_ año:Int,_ precio:Double, _ cantidadDePasajeros: Int){
        self.CantidadDePasajeros = cantidadDePasajeros
        super.init(numeroDeSerieMotor, marca, año, precio)
    }
}
class Camioneta: Vehiculo{
    var CantidadDeCargaKG : Int = 0
    var CantidadDeEjes : Int = 0
    var CantidadDeRodadas : Int = 0
    init(_ numeroDeSerieMotor:String, _ marca:String,_ año:Int,_ precio:Double, _ cantidadDeCargaKG: Int, _ cantidadDeEjes: Int, _ cantidadDeRodadas: Int){
        self.CantidadDeCargaKG = cantidadDeCargaKG
        self.CantidadDeEjes = cantidadDeEjes
        self.CantidadDeRodadas = cantidadDeRodadas
        super.init(numeroDeSerieMotor, marca, año, precio)
    }
}
var autoCompacto = AutoCompacto("NSerie", "Ford", 2020, 20000, 5)
print(autoCompacto.Precio)

```

```

53
54     var autoCompacto = AutoCompacto("NSerie", "Ford", 2020, 20000, 5)
55     print(autoCompacto.Precio)

```

20000.0

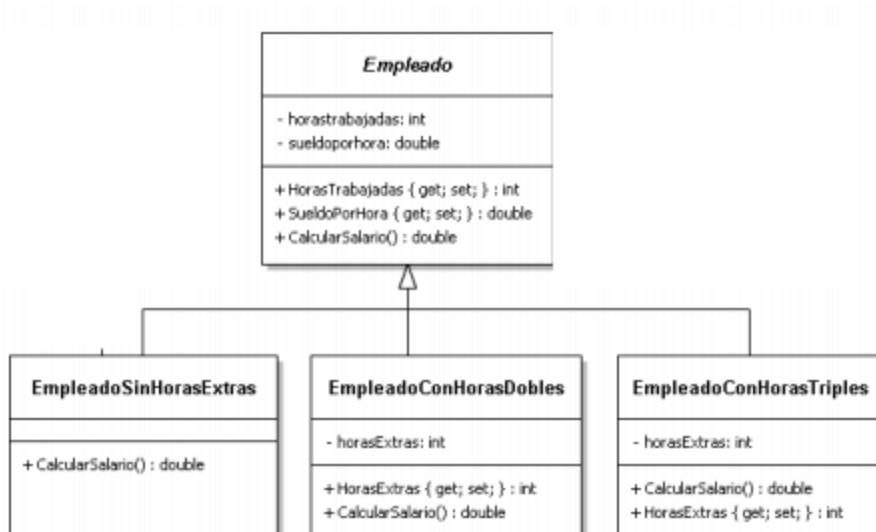
Por medio de la herencia y el llamado del constructor de la clase padre, los cuales se vieron en clase, se pudo clasificar cada uno de los vehículos que pide el ejercicio, dejando 3 grupos principales, 1 para Auto (desciende auto de lujo y auto compacto), Camioneta y Vagoneta. Todos ellos derivados de la clase Vehículo que es la que tiene todas las características en común.

2.3.47. Calcule el salario semanal de un empleado en base a las horas trabajadas, el sueldo por hora y los siguientes criterios:

- Si las horas trabajadas son más de 40, entonces el excedente se considera hora extra
- Si las horas totales se ubican entre 41 y 45, entonces cada hora extra se paga el doble
- Si las horas totales son más de 45, entonces cada hora extra se paga el triple

Pruebas

horas trabajadas	salario p/hora	horas extra	salario semanal
40	2	0	80
43	2	3	92
46	2	6	116



```

import Foundation

protocol Empleado{
    var HorasTrabajadas: Int{get set}
    var SueldoPorHora: Double{get set}
    func CalcularSalario() -> Double
}

class EmpleadoSinHorasExtras: Empleado{
    var HorasTrabajadas: Int = 0
    var SueldoPorHora: Double = 0
    init(horasTrabajadas : Int, sueldoPorHora: Double){
        self.HorasTrabajadas = horasTrabajadas
        self.SueldoPorHora = sueldoPorHora
    }
    public func CalcularSalario() -> Double{
        return Double(HorasTrabajadas) * SueldoPorHora
    }
}

class EmpleadoConHorasDobles: Empleado{
    var HorasTrabajadas: Int = 0
    var SueldoPorHora: Double = 0
    public var HorasExtras: Int = 0
    init(horasTrabajadas : Int, sueldoPorHora: Double, horasExtras : Int){
        self.HorasTrabajadas = horasTrabajadas
        self.SueldoPorHora = sueldoPorHora
        self.HorasExtras = horasExtras
    }
    public func CalcularSalario() -> Double{
        return (Double(HorasTrabajadas) * SueldoPorHora) + (Double(HorasExtras) * 2 * SueldoPorHora)
    }
}

class EmpleadoConHorasTriples: Empleado{
    var HorasTrabajadas: Int = 0
    var SueldoPorHora: Double = 0
    var HorasExtras: Int = 0
    init(horasTrabajadas : Int, sueldoPorHora: Double, horasExtras : Int){
        self.HorasTrabajadas = horasTrabajadas
        self.SueldoPorHora = sueldoPorHora
        self.HorasExtras = horasExtras
    }
    public func CalcularSalario() -> Double{
        return (Double(HorasTrabajadas) * SueldoPorHora) + (Double(HorasExtras) * 3 * SueldoPorHora)
    }
}

class EmpleadoFactory{
    public static func GenerarEmpleado( horas: Int, salario: Double) -> Empleado{
        var empleado : Empleado
        switch(horas){
            case 41...45:
                empleado = EmpleadoConHorasDobles(horasTrabajadas : 40, sueldoPorHora: salario, horasExtras : horas-40)
            case 46...:
                empleado = EmpleadoConHorasTriples(horasTrabajadas : 40, sueldoPorHora: salario, horasExtras : horas-40)
            default:
                empleado = EmpleadoSinHorasExtras(horasTrabajadas : 40, sueldoPorHora: salario)
        }
        return empleado
    }
}

print(EmpleadoFactory.GenerarEmpleado(horas: 46, salario: 2).CalcularSalario());

```

```
70  
71     print(EmppleadoFactory.GenerarEmpleado(horas: 40, salario: 2).CalcularSalario());  
72  
73
```

```
80.0
```

```
71     print(EmppleadoFactory.GenerarEmpleado(horas: 43, salario: 2).CalcularSalario());  
72  
73
```

```
92.0
```

```
70  
71     print(EmppleadoFactory.GenerarEmpleado(horas: 46, salario: 2).CalcularSalario());  
72  
73
```

```
116.0
```

Teniendo en cuenta la jerarquía de las clases presentada en el ejercicio, lo más conveniente para la selección de la clase para la responsabilidad del cálculo del salario fue hacer uso del patrón de diseño FactoryMethod, dejando a esta la responsabilidad de decidir qué clase debe de realizar el cálculo del salario por medio de los datos de entrada dados (Horas trabajadas y salario)

2.3.48. Una agencia de renta de vehículos dispone de autobuses y tractores.

Debe crear una clase abstracta llamada Vehículo que contenga:

- Propiedades:
 - Número de placa
- Métodos
 - Calcular importe
 - Mostrar datos

La renta de autobuses se factura por kilómetros. Debido a esto, los datos de la clase Autobús son:

- El precio por kilómetro.
- La cantidad de kilómetros que tiene el autobús cuando se renta.
- La cantidad de kilómetros que tiene el autobús cuando se devuelve.

En cambio, la renta de tractores se factura por días. Debido a esto, los datos de la clase Tractor son:

- El precio por día.
- El día del mes actual en que lo está rentando.
- El día del mes actual en que lo está devolviendo.
 - Suponga que la renta de un tractor solo debe realizarse en el mismo mes

Cuando se renta un vehículo, se deben capturar sus placas, sus datos correspondientes (de acuerdo con el tipo de vehículo).

Cuando se devuelve un vehículo, se calcula el importe a pagar por la renta y se muestran sus datos.

```

import Foundation

class Vehiculo{
    var numeroDePlaca: String = ""

    init(numeroDePlaca: String){
        self.numeroDePlaca = numeroDePlaca
    }
    func CalcularImporte() -> Double{
        return 0
    }
    func MostrarDatos(){}
}

class Autobus : Vehiculo{
    var precioKM: Double = 0
    var cantidadKMINicial: Double = 0
    var cantidadKMFfinal: Double = 0
    init(numeroDePlaca: String ,precioKM : Double, cantidadKMINicial : Double, cantidadKMFfinal: Double)
        super.init(numeroDePlaca : numeroDePlaca )
        self.precioKM = precioKM
        self.cantidadKMINicial = cantidadKMINicial
        self.cantidadKMFfinal = cantidadKMFfinal
    }
    override func CalcularImporte() -> Double{
        return    precioKM * (cantidadKMFfinal - cantidadKMINicial)
    }
    override func MostrarDatos(){
        print( "Numero de placa: \(String(self.numeroDePlaca)) ")
        print( "Precio por KM: \(String(self.precioKM)) ")
        print( "KM iniciales : \(String(self.cantidadKMINicial)) ")
        print( "KM finales: \(String(self.cantidadKMFfinal)) ")
        print( "Importe calculado: \(String(self.CalcularImporte())) ")
    }
}

```

```

class Tractor: Vehiculo{
    var precioDia: Double = 0
    var numDiaInicial: Double = 0
    var numDiaFinal: Double = 0
    init(numeroDePlaca:String, precioDia:Double, numDiaInicial:Double, numDiaFinal:Double ){
        super.init(numeroDePlaca : numeroDePlaca )
        self.precioDia = precioDia
        self.numDiaInicial = numDiaInicial
        self.numDiaFinal = numDiaFinal
    }
    override func CalcularImporte() -> Double{
        return  precioDia * (numDiaFinal - numDiaInicial)
    }
    override func MostrarDatos(){
        print( "Numero de placa: \(String(self.numeroDePlaca )) ")
        print( "Precio por dia: \(String(self.precioDia)) ")
        print( "Dia inicial del mes: \(String(self.numDiaInicial)) ")
        print( "Dia final del mes: \(String(self.numDiaFinal)) ")
        print( "Importe calculado: \(String(self.CalcularImporte())) ") 
    }
}
var autoBus = Autobus(numeroDePlaca: "DSFFASD223", precioKM: 100,cantidadKMI inicial : 20, cantidadKMFinal: 30 )
print(autoBus.MostrarDatos())
var tractor = Tractor(numeroDePlaca: "FWEWEF2332", precioDia: 100, numDiaInicial: 1, numDiaFinal: 22)
print(tractor.MostrarDatos())

```

Se crea una clase abstracta Vehículo la cual sus descendientes van a ser Tractor y Autobus, la diferencia entre estos dos es la manera en que se calcula el importe, sin embargo, ambos tienen que realizar esta acción, así como la acción del mostrado de datos.

2.3 PRACTICAS DE PROGRAMACIÓN

Jessica Janet Grajeda Castellanos



2.3.1. Se tiene la edad de Loky (en años perro) en una variable. Determine la edad de Loky en años humanos, considerando que 1 año humano equivale a 7 años perro.

Pruebas

edadLokyPerro	edadLokyHumano
50	7
14	2
32	4

```

1 import Foundation
2
3 var edadLokyPerro = 50
4 let edadLokyHumano = edadLokyPerro / 7
5
6 print("Edad de Loky en años perro: \(edadLokyPerro)\nEdad de Loky en años humanos: \(edadLokyHumano)")

```

2.3.2. Se tiene el tiempo de un recorrido, almacenado en tres variables: una para la hora, otra para los minutos y otra para los segundos; así también, se tiene la distancia (en metros) en otra variable. Despliegue la velocidad en metros/segundo, kilómetros/hora y millas/hora.

Pruebas

distancia	horas	minutos	segundos	Mts/seg	Km/h	Millas/h
2500	5	56	23	0.116915	0.420895	0.261588
50000	1	35	56	8.686588	31.27172	19.4355

```

1 import Foundation
2
3 var hora: Float = 1
4 var minutos: Float = 35
5 var segundos: Float = 56
6 var distancia: Float = 50000
7
8 var metrosPorSeg: Float = distancia / (hora * 3600 + minutos * 60 + segundos)
9 var kmPorHora: Float = metrosPorSeg * (1 / 1000) * (3600)
10 var millPorHora: Float = kmPorHora / 1.609
11
12
13 print("La velocidad en m/s es: \(metrosPorSeg)")
14 print("La velocidad en km/h es: \(kmPorHora)")
15 print("La velocidad en millas/h es: \(millPorHora)")

```

2.3.3. Dadas la suma y la diferencia entre dos números, encuentre los valores de dichos números y almacénelos en variables llamadas a y b . Imprima los resultados.

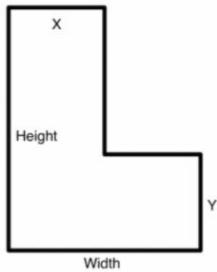
Pruebas

sum	dif	a	b
16	4	10	6
11	3	7	4
4	2	3	1

```

1 import Foundation
2
3 var sum = 4
4 var dif = 2
5
6 var b = (sum - dif) / 2
7 var a = sum - b
8
9 print("a = \(a)")
10 print("b = \(b)")
```

2.3.4. Se tienen cuatro variables ($ancho$, $alto$, x , y), que describen las dimensiones de una figura en forma de **L**. Calcule, almacene e imprima el perímetro y el área de dicha figura.



Pruebas

ancho	alto	x	y	perímetro	área
8	12	4	3	40	60
8	4	2	2	24	20

```

1 import Foundation
2
3 var ancho = 8
4 var alto = 4
5 var x = 2
6 var y = 2
7
8 var perimetro = ancho + alto + (alto - y) + (ancho - x) + x + y
9 var area = (ancho * y) + ((alto - y) * x)
10
11 print("Perímetro: \(perimetro)")
12 print("Área: \(area)")
```

2.3.5. Dado un número de tres dígitos almacenado en *a*, calcule aritméticamente e imprima el último dígito.

Pruebas

a	ultimo digito
123	3
337	7
100	0

```
1 import Foundation
2
3 var a = 100
4
5 print("El último dígito de \$(a) es \$(a % 10)")
```

2.3.6. En *x* años a partir de ahora, Anahí será *y* veces mayor en edad que su hermana Abril. Conociendo la edad actual de Abril, calcule la edad de Anahí.

Pruebas

x	y	abril	anahi
3	2	12	27
1	3	12	38

```
1 import Foundation
2
3 var x = 3
4 var y = 2
5 var abril = 12
6
7 var anahi = abril * y + x
8
9 print("La edad de Anahí será \$(anahi)")
```

2.3.7. Suponga que usted tiene x manzanas. Lizbeth intercambia 3 naranjas por 5 manzanas. ¿Cuántas naranjas puede obtener de Lizbeth y cuantas manzanas le quedarían después del intercambio? Imprima los resultados.

Pruebas

x	manzanas	naranjas
17	2	9
25	0	15
4	4	0

```
1 import Foundation
2
3 var x = 17
4
5 var manzanas = x - 5 * (x / 5)
6 var naranjas = (x / 5) * 3
7
8 print("Cantidad de manzanas: \(manzanas)")
9 print("Cantidad de naranjas: \(naranjas)")
```

2.3.8. Dados dos números en a y b , determine e imprima el mayor de ellos o indique si son iguales.

Pruebas

a	b	mayor
11	22	22
23	12	23
8	8	Iguales

```
1 import Foundation
2
3 var a = 8
4 var b = 8
5
6 if a == b {
7     print("Iguales")
8 } else if a > b {
9     print("El mayor es \(a)")
10 } else {
11     print("El mayor es \(b)")
12 }
```

2.3.9. Imprima si el valor en *x* es par o impar.

Pruebas

x	resultado
31	Impar
38	Par

```
1 import Foundation
2
3 var x = 38
4
5 if (x % 2) == 0 {
6     print("\n(x) es par")
7 } else {
8     print("\n(x) es impar")
9 }
```

2.3.10. Tiene dos números en *a* y *b*. Indique si *a* es divisible entre *b*.

Pruebas

a	b	resultado
22	11	Divisible
12	5	No divisible
18	9	Divisible

```
1 import Foundation
2
3 var a = 18
4 var b = 9
5
6 if a % b == 0 {
7     print("\n(a) si es divisible entre \n(b)")
8 } else {
9     print("\n(a) no es divisible entre \n(b)")
10 }
```

2.3.11. Se tienen tres variables (a, b, c). Imprima la leyenda “Al menos dos variables son iguales” o bien la leyenda “Todas las variables son diferentes”, según sea el caso.

Pruebas

a	b	c	resultado
1	2	3	Todas las variables son diferentes
1	2	1	Al menos dos variables son iguales
8	8	8	Al menos dos variables son iguales

```

1 import Foundation
2
3 var a = 8
4 var b = 8
5 var c = 8
6
7 if a == b {
8     print("Al menos dos variables son iguales")
9 } else if a == c {
10    print("Al menos dos variables son iguales")
11 } else if b == c {
12    print("Al menos dos variables son iguales")
13 } else {
14     print("Todas las variables son diferentes")
15 }
```

2.3.12. Antes de preparar el desayuno, debemos saber si aún se pueden cocinar los ingredientes. Considere que los blanquillos se echan a perder después de tres semanas (21 días) y el tocino después de una semana (7 días). Dados los días de los blanquillos y el tocino en su refrigerador, determine si ambos se pueden cocinar, o si hay alguno que deba desecharse.

Pruebas

blanquillos	tocino	resultado
2	3	Ambos se pueden cocinar
20	9	Debe desechar el tocino
23	11	Debe desechar ambos
25	6	Debe desechar los blanquillos

```

1 import Foundation
2
3 var blanquillos = 25
4 var tocino = 6
5
6 if blanquillos < 21 && tocino < 7 {
7     print("Ambos se pueden cocinar")
8 } else if blanquillos > 21 && tocino > 7 {
9     print("Se deben desechar ambos")
10 } else if blanquillos > 21 {
11     print("Se debe desechar los blanquillos")
12 } else {
13     print("Se debe desechar el tocino")
14 }
```

2.3.13. Dado un año, imprima si es o no es bisiesto. Las reglas (para este ejercicio) para determinar un año bisiesto, son las siguientes:

- Divisible entre 4
- Cada inicio de siglo, si el año es divisible entre 400

Pruebas

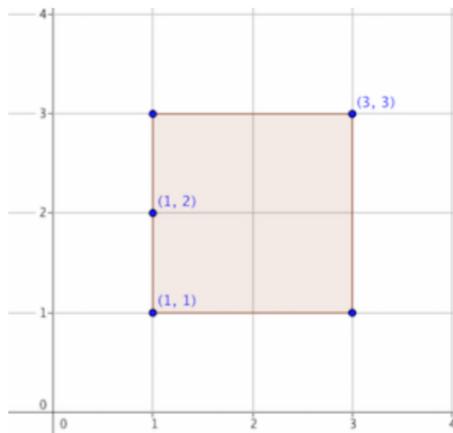
año	resultado
2000	Es bisiesto
1985	No es bisiesto
1900	No es bisiesto
2016	Es bisiesto

```
1 import Foundation
2
3 var año = 2016
4
5 if (año % 4 == 0 && año % 100 != 0) || año % 400 == 0 {
6     print("El año \(año) es bisiesto")
7 } else {
8     print("El año \(año) no es bisiesto")
9 }
```

2.3.14. La función `Int.random(in: mínimo...máximo)` entrega un valor aleatorio, comprendido entre mínimo y máximo. Utilice dicha función para simular que lanza un volado con una moneda y determine si cayó águila (par) o sello (impar).

```
1 import Foundation
2
3 let random = Int.random(in: 1...100)
4
5 if random % 2 == 0 {
6     print("Aleatorio: \(random). Es águila")
7 } else {
8     print("Aleatorio: \(random). Es sello")
9 }
```

2.3.15. Determine si un punto o coordenada (x, y) se encuentra dentro de un rectángulo definido por las coordenadas de la esquina inferior izquierda $(x1, y1)$ y la esquina superior derecha $(x2, y2)$.



```
1 import Foundation
2
3 let x = 2, y = 2
4 let x1 = 1, y1 = 1
5 let x2 = 3, y2 = 3
6
7 if (x1...x2 ~= x) && (y1...y2 ~= y) {
8     print("Dentro")
9 } else {
10    print("Fuera")
11 }
```

2.3.16. Dado un carácter, determine si es vocal, consonante, dígito, operador aritmético u otro.

```
1 import Foundation
2
3 let char: UnicodeScalar = "+"
4 let valor = char.value
5
6 if char == "a" || char == "e" || char == "i" || char == "o" || char == "u" ||
7 char == "A" || char == "E" || char == "I" || char == "O" || char == "U" {
8     print("El carácter '\(char)' es vocal")
9 } else if 65...90 ~= valor || 97...122 ~= valor {
10    print("El carácter '\(char)' es consonante")
11 } else if 48...57 ~= valor {
12     print("El carácter '\(char)' es un dígito")
13 } else if ["+", "-", "/", "*"].contains(char) {
14     print("El carácter '\(char)' es un operador aritmético")
15 } else {
16     print("Otro")
17 }
```

2.3.17. En base al mes y día de nacimiento, determine el signo zodiacal de una persona.

```
1 import Foundation
2
3 let mes = 10
4 let dia = 22
5
6 if (mes == 1 && 20...31 ~= dia) || (mes == 2 && 1...18 ~= dia) {
7     print("Signo zodiacal: Acuario")
8 } else if (mes == 2 && 19...29 ~= dia) || (mes == 3 && 1...20 ~= dia) {
9     print("Signo zodiacal: Piscis")
10 } else if (mes == 3 && 21...31 ~= dia) || (mes == 4 && 1...19 ~= dia) {
11     print("Signo zodiacal: Aries")
12 } else if (mes == 4 && 20...30 ~= dia) || (mes == 5 && 1...20 ~= dia) {
13     print("Signo zodiacal: Tauro")
14 } else if (mes == 5 && 21...31 ~= dia) || (mes == 6 && 1...20 ~= dia) {
15     print("Signo zodiacal: Géminis")
16 } else if (mes == 6 && 21...30 ~= dia) || (mes == 7 && 1...22 ~= dia) {
17     print("Signo zodiacal: Cáncer")
18 } else if (mes == 7 && 23...31 ~= dia) || (mes == 8 && 1...22 ~= dia) {
19     print("Signo zodiacal: Leo")
20 } else if (mes == 8 && 23...31 ~= dia) || (mes == 9 && 1...22 ~= dia) {
21     print("Signo zodiacal: Virgo")
22 } else if (mes == 9 && 23...30 ~= dia) || (mes == 10 && 1...22 ~= dia) {
23     print("Signo zodiacal: Libra")
24 } else if (mes == 10 && 23...31 ~= dia) || (mes == 11 && 1...21 ~= dia) {
25     print("Signo zodiacal: Escorpio")
26 } else if (mes == 11 && 22...30 ~= dia) || (mes == 12 && 1...21 ~= dia) {
27     print("Signo zodiacal: Sagitario")
28 } else {
29     print("Signo zodiacal: Capricornio")
30 }
```

2.3.18. Un número cuadrado perfecto es el producto de multiplicar un entero por sí mismo. Imprima los N cuadrados perfectos.

Pruebas	
N	salida
2	1 4
5	1 4 9 16 25

```
1 import Foundation
2
3 let n = 5
4
5 for cuadrado in 1...n {
6     print(cuadrado * cuadrado)
7 }
```

2.3.19. Imprima la serie de números de 1 a N alternando su orden, es decir, el primer número siempre es el 1, seguido de N , luego 2, luego $N-1$ y así sucesivamente. La serie debe imprimirse en un solo renglón, separando los números con espacios.

Pruebas

N	salida
4	1 4 2 3
9	1 9 2 8 3 7 4 6 5

```
1 import Foundation
2
3 let n = 9
4 var aux = 0
5 var salida: String = ""
6
7 for i in 1...n {
8     if i % 2 == 0 {
9         salida += "\((n - (aux - 1)) "
10    } else {
11        salida += "\((aux + 1) "
12        aux = aux + 1
13    }
14 }
15
16 print(salida)
```

- 2.3.20.** Dado N , dibuje una pirámide de asteriscos. La pirámide debe tener N líneas. En la i -ésima línea debe haber $N-i$ espacios, seguido de i^2-1 asteriscos.

Pruebas

N	salida
1	*
2	 ***
3	 *** *****
4	 *** ***** ******

```
1 import Foundation
2
3 var n = 4
4 print("Pirámide")
5
6 for i in 1...n {
7     let espacios = n - i
8     let asteriscos = (i * 2) - 1
9     var linea = ""
10
11     if espacios > 0 {
12         linea = String(repeating: " ", count: espacios)
13     }
14     linea += String(repeating: "*", count: asteriscos)
15     print(linea)
16 }
```

2.3.21. Imprima los primeros N números de la serie de *Fibonacci*. Los primeros dos números de la serie siempre son 1, el resto son la suma de los dos anteriores. Imprima la serie en un solo renglón con los valores separados por coma.

Pruebas

N	salida
3	1, 1, 2
6	1, 1, 2, 3, 5, 8
10	1, 1, 2, 3, 5, 8, 13, 21, 34, 55

```
1 import Foundation
2
3 var n = 10
4 var aux1 = 0, aux2 = 1
5 var fibonacci: String = "1 "
6
7 for _ in 1..<n {
8     let aux = aux1 + aux2
9     aux1 = aux2
10    aux2 = aux
11
12    fibonacci += "\(aux) "
13 }
14
15 print(fibonacci)
```

2.3.22. Dado un número en x , determine si es un número primo. Los números primos solo pueden dividirse entre 1 y entre sí mismos (división exacta).

Pruebas

x	resultado
2	Es primo
3	Es primo
15	No es primo
1	No es primo

```
1 import Foundation
2
3 var numero = 15
4 var contador = 2
5 var primo = true
6
7 while (primo) && (contador != numero) {
8     if numero == 1 [
9         primo = false
10    } else if numero % contador == 0 {
11        primo = false
12    }
13
14    contador += 1
15 }
16
17 if primo {
18     print("\(numero) es primo")
19 } else {
20     print("\(numero) no es primo")
21 }
```

2.3.23. Convierta un número de decimal a binario con el método de escalera.

Pruebas

decimal	resultado
78	1001110
92	1011100
2015	11111011111

```
1 import Foundation
2
3 var decimal = 92
4 var binario: String = ""
5 while decimal > 0 {
6     binario = decimal % 2 == 0 ? "0" + binario : "1" + binario;
7     decimal = decimal / 2
8 }
9 print(binario)
```

2.3.24. Imprima el elemento mayor de un arreglo de N números.

Pruebas

Arreglo	mayor
[1, 2, 3, 10, 100]	100
[10, 12, 33, 11, 1, 8]	33

```
1 import Foundation
2
3 let arreglo = [1, 2, 3, 10, 100]
4 let mayor = arreglo.max()
5
6 print(mayor!)
```

2.3.25. Imprima los elementos de un vector en orden inverso al que se encuentran almacenados.

Pruebas

vector	salida
[1, 2, 3, 10, 100]	100, 10, 3, 2, 1
[10, 12, 33, 11, 1, 8]	8, 1, 11, 33, 12, 10

```
1 import Foundation
2
3 let vector = [10, 12, 33, 11, 1, 8]
4 var salida = ""
5
6 ∵ for i in vector.reversed() {
7   |
8     salida += i == vector[0] ? "\u2028(i)" : "\u2028(i), "
9 }
10
11 print(salida)
```

2.3.26. Invierta los elementos de un arreglo sin crear otro.

Pruebas

arreglo	salida
[1, 2, 3, 10, 100]	[100, 10, 3, 2, 1]
[10, 12, 33, 11, 1, 8]	[8, 1, 11, 33, 12, 10]

```
1 import Foundation
2
3 var vector = [10, 12, 33, 11, 1, 8]
4 vector.reverse()
5
6 print(vector)
```

2.3.27. Dados los vectores A y B , imprima todos los elementos de B que se encuentren en A . Si no existen elementos en común, no debe imprimir nada.

Pruebas

A	B	salida
[1, 2, 3, 10, 100]	[1, 2, 3, 4, 5, 6]	1 2 3
[1, 2, 3, 10, 100]	[5, 2, 3, 10, 13]	2 3 10
[1, 2, 3, 10, 100]	[5, 6]	

```

1 import Foundation
2
3 let a = [1, 2, 3, 10, 100]
4 let b = [5, 2, 3, 10, 13]
5
6 for i in b {
7     if a.contains(i) {
8         print(i)
9     }
10 }
```

2.3.28. Extraiga cada dígito, de izquierda a derecha, de un número dado en n y almacénelo en un vector. Resuélvalo aritméticamente, es decir, sin convertir n a cadena de caracteres.

Pruebas

n	vector
12345	[1, 2, 3, 4, 5]
20143831	[2, 0, 1, 4, 3, 8, 3, 1]

```

1 import Foundation
2
3 var n = 20143831
4 var arreglo: [Int] = []
5
6 arreglo.append(n % 10)
7
8 while n >= 10 {
9     n = n / 10
10    arreglo.insert(n % 10, at: 0)
11 }
12
13 print(arreglo)
```

2.3.29. Dadas las matrices

$$A = \begin{pmatrix} 2 & 0 & 1 \\ 3 & 0 & 0 \\ 5 & 1 & 1 \end{pmatrix}; \quad B = \begin{pmatrix} 1 & 0 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 0 \end{pmatrix}$$

Calcular:

a) $A + B$

$$A + B = \begin{pmatrix} 2 & 0 & 1 \\ 3 & 0 & 0 \\ 5 & 1 & 1 \end{pmatrix} + \begin{pmatrix} 1 & 0 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 2+1 & 0+0 & 1+1 \\ 3+1 & 0+2 & 0+1 \\ 5+1 & 1+1 & 1+0 \end{pmatrix} = \begin{pmatrix} 3 & 0 & 2 \\ 4 & 2 & 1 \\ 6 & 2 & 1 \end{pmatrix}$$

```
1 import Foundation
2
3 let a: [[Int]] = [[2, 0, 1],
4 | | | [3, 0, 0],
5 | | | [5, 1, 1]]
6
7 let b: [[Int]] = [[1, 0, 1],
8 | | | [1, 2, 1],
9 | | | [1, 1, 0]]
10
11 let c: [[Int]] = [[[a[0][0] + b[0][0], a[0][1] + b[0][1], a[0][2] + b[0][2]],
12 | | | [a[1][0] + b[1][0], a[1][1] + b[1][1], a[1][2] + b[1][2]],
13 | | | [a[2][0] + b[2][0], a[2][1] + b[2][1], a[2][2] + b[2][2]]]
14
15 print(c)
```

b) $A - B$

$$A - B = \begin{pmatrix} 2 & 0 & 1 \\ 3 & 0 & 0 \\ 5 & 1 & 1 \end{pmatrix} - \begin{pmatrix} 1 & 0 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 2-1 & 0-0 & 1-1 \\ 3-1 & 0-2 & 0-1 \\ 5-1 & 1-1 & 1-0 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 2 & -2 & -1 \\ 4 & 0 & 1 \end{pmatrix}$$

```
1 import Foundation
2
3 let a: [[Int]] = [[2, 0, 1],
4 | | | [3, 0, 0],
5 | | | [5, 1, 1]]
6
7 let b: [[Int]] = [[1, 0, 1],
8 | | | [1, 2, 1],
9 | | | [1, 1, 0]]
10
11 let c: [[Int]] = [[a[0][0] - b[0][0], a[0][1] - b[0][1], a[0][2] - b[0][2]],
12 | | | [a[1][0] - b[1][0], a[1][1] - b[1][1], a[1][2] - b[1][2]],
13 | | | [a[2][0] - b[2][0], a[2][1] - b[2][1], a[2][2] - b[2][2]]]
14
15 print(c)
```

c) A x B

$$A \cdot B = \begin{pmatrix} 2 & 0 & 1 \\ 3 & 0 & 0 \\ 5 & 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 2 \cdot 1 + 0 \cdot 1 + 1 \cdot 1 & 2 \cdot 0 + 0 \cdot 2 + 1 \cdot 1 & 2 \cdot 1 + 0 \cdot 1 + 1 \cdot 0 \\ 3 \cdot 1 + 0 \cdot 1 + 0 \cdot 1 & 3 \cdot 0 + 0 \cdot 2 + 0 \cdot 1 & 3 \cdot 1 + 0 \cdot 1 + 0 \cdot 0 \\ 5 \cdot 1 + 1 \cdot 1 + 1 \cdot 1 & 5 \cdot 0 + 1 \cdot 2 + 1 \cdot 1 & 5 \cdot 1 + 1 \cdot 1 + 1 \cdot 0 \end{pmatrix} = \begin{pmatrix} 3 & 1 & 2 \\ 3 & 0 & 3 \\ 7 & 3 & 6 \end{pmatrix}$$

```
1 import Foundation
2
3 let a: [[Int]] = [[2, 0, 1],
4 | | | | [3, 0, 0],
5 | | | | [5, 1, 1]]
6
7 let b: [[Int]] = [[1, 0, 1],
8 | | | | [1, 2, 1],
9 | | | | [1, 1, 0]]
10
11 let c: [[Int]] = [
12   [
13     a[0][0] * b[0][0] + a[0][1] * b[1][0] + a[0][2] * b[2][0],
14     a[0][0] * b[0][1] + a[0][1] * b[1][1] + a[0][2] * b[2][1],
15     a[0][0] * b[0][2] + a[0][1] * b[1][2] + a[0][2] * b[2][2],
16   ],
17   [
18     a[1][0] * b[0][0] + a[1][1] * b[1][0] + a[1][2] * b[2][0],
19     a[1][0] * b[0][1] + a[1][1] * b[1][1] + a[1][2] * b[2][1],
20     a[1][0] * b[0][2] + a[1][1] * b[1][2] + a[1][2] * b[2][2],
21   ],
22   [
23     a[2][0] * b[0][0] + a[2][1] * b[1][0] + a[2][2] * b[2][0],
24     a[2][0] * b[0][1] + a[2][1] * b[1][1] + a[2][2] * b[2][1],
25     a[2][0] * b[0][2] + a[2][1] * b[1][2] + a[2][2] * b[2][2],
26   ]
27 ]
28
29 print(c)
```

2.3.30. Dada una matriz cuadrada A , almacene los elementos de la diagonal principal y los de la diagonal inversa, en vectores llamados DP y DI respectivamente.

Pruebas

A	DP	DI
3 5 8 2	3	2
2 7 9 5	7	9
2 8 9 2	9	8
4 6 7 1	1	4

```
1 import Foundation
2
3 let a: [[Int]] = [
4     [3, 5, 8, 2],
5     [2, 7, 9, 5],
6     [2, 8, 9, 2],
7     [4, 6, 7, 1]
8 ]
9 var dp: [Int] = []
10 var di: [Int] = []
11 var aux: Int = 1
12
13 for (i, items) in a.enumerated() {
14     dp.append(items[i])
15     di.append(items[items.endIndex - 1 * aux])
16     aux += 1
17 }
18 print(dp)
19 print(di)
```

- 2.3.31.** Dada una matriz cuadrada A , imprima el resultado de sumar los elementos que no corresponden a la periferia de la matriz.

Pruebas

A

3	5	8	9	2
1	4	2	1	0
4	5	4	8	1
9	8	1	0	3
7	2	1	1	3

Suma=33

```
1 import Foundation
2
3 let a: [[Int]] = [
4     [3, 5, 8, 9, 2],
5     [1, 4, 2, 1, 0],
6     [4, 5, 4, 8, 1],
7     [9, 8, 1, 0, 3],
8     [7, 2, 1, 1, 3]
9 ]
10
11 var suma: Int = 0
12
13 for (i, items) in a.enumerated() {
14     if i != 0 && i != a.endIndex - 1 {
15         var indice = 0
16         for item in items {
17             if indice != 0 && indice != items.endIndex - 1 {
18                 suma += item
19             }
20             indice += 1
21         }
22     }
23 }
24
25 print("La suma de los elementos distintos a la periferia es: \(suma)")
```

2.3.32. Escriba un método que reciba dos valores enteros y devuelva el mayor de ellos. Utilice el `_` para ignorar el nombre externo de cada parámetro.

Pruebas

a	b	mayor
2	5	5
8	3	8

```
1 import Foundation
2
3 let a = 8
4 let b = 3
5
6 func mayor(_ a: Int, _ b: Int) -> Int {
7     return a > b ? a : b
8 }
9
10 print("El número mayor entre \(a) y \(b) es \(mayor(a, b))")
```

2.3.33. Escriba un método que reciba un número entero y devuelva un vector con los valores de 1 a N .

Pruebas

N	vector
8	[1, 2, 3, 4, 5, 6, 7, 8]
3	[1, 2, 3]

```
1 import Foundation
2
3 func vector(_ n: Int) -> [Int] {
4     var arreglo: [Int] = []
5     if n == 0 {
6         return arreglo
7     }
8
9     for i in 1...n {
10        arreglo.append(i)
11    }
12
13    return arreglo
14 }
15
16 print(vector(0))
```

2.3.34. Programe un método que reciba cuatro parámetros: hora inicial, minutos iniciales, hora final y minutos finales. La función debe calcular los minutos transcurridos entre los dos horarios compuestos de horas y minutos. Si los minutos se omiten en la invocación, se considerará que el valor es cero.

Pruebas

horaInicial	minutoInicial	horaFinal	minutoFinal	diferencia
12	3	13	10	67
8	10	17	30	560

```
1 import Foundation
2
3 func tiempoTranscurrido(horaInicial: Int, minInicial: Int = 0, horaFinal: Int, minFinal: Int = 0) -> Int {
4     let desde: Int = (horaInicial * 60) + minInicial
5     let hasta: Int = (horaFinal * 60) + minFinal
6
7     return hasta - desde
8 }
9
10 print("La diferencia es de: \(tiempoTranscurrido(horaInicial: 12, minInicial: 3, horaFinal: 13, minFinal: 10))")
```

2.3.35. Una pila es una estructura de datos (arreglo unidimensional), que puede realizar lo siguiente:

- Agregar un elemento al principio del arreglo
- Obtener el valor del primer elemento del arreglo
- Entregar el primer elemento del arreglo y removerlo (sacarlo de la pila)

Debe crear un método para cada operación descrita anteriormente, mismas que deberán trabajar sobre un arreglo pasado como parámetro por referencia.

```
1 import Foundation
2
3 //Aregar un valor al inicio del arreglo
4 func push(_ arreglo: inout [Int], _ valor: Int) {
5     arreglo.insert(valor, at: 0)
6 }
7
8 //Eliminar el primero
9 func pop(_ arreglo: inout [Int]) -> Int {
10    if arreglo.count > 0 {
11        let elemento: Int = arreglo[0]
12        arreglo.remove(at: 0)
13        return elemento
14    }
15    return 0
16 }
17
18 //Obtener el primero
19 func obtener(_ arreglo: inout [Int]) -> Int {
20    return arreglo[0]
21 }
22
23 var array: [Int] = []
24
25
26 push(&array, 6)
27 push(&array, 7)
28 push(&array, 2)
29 print(array)
30 print(pop(&array))
31 print(obtener(&array))
```

2.3.36. Escriba los métodos para agregar y entregar un elemento bajo el concepto de la estructura de dato llamada cola. Utilice el arreglo como parámetro por referencia.

```
1 import Foundation
2
3 //Agregar un valor al final del arreglo
4 func agregar(_ arreglo: inout [Int], _ valor: Int) {
5     arreglo.append(valor)
6 }
7
8 //Eliminar el primero
9 func eliminar(_ arreglo: inout [Int]) {
10    if arreglo.count > 0 {
11        arreglo.remove(at: 0)
12    }
13 }
14
15 //Obtener el primero
16 func obtener(_ arreglo: inout [Int]) -> Int {
17     return arreglo[0]
18 }
19
20 var array: [Int] = []
21
22
23 agregar(&array, 6)
24 agregar(&array, 7)
25 agregar(&array, 2)
26 print(array)
27 eliminar(&array)
28 print(array)
29 print(obtener(&array))
```

2.3.37. Escriba un método que calcule la factorial de un número dado como parámetro. Utilice recursividad.

Pruebas

n	factorial
3	6
5	120
10	3628800

```
1 import Foundation
2
3 func factorial(_ n: Int) -> Int {
4     if n == 0 {
5         return 1
6     }
7
8     return n * factorial(n - 1)
9 }
10
11 print(factorial(10))
```

2.3.38. Programe un método recursivo que calcule la multiplicación de dos números dados como parámetros por referencia (A y B). Deberá utilizar el método ruso para multiplicar, el cual (en papel) consiste en escribir una serie de pares números bajo dos columnas. La primera columna la encabeza el valor A y la segunda columna la encabeza el valor B. El valor A se divide entre dos y debajo se coloca el cociente entero de la división (ignorando el residuo), repetir sucesivamente hasta llegar al resultado 1. Por cada vez que se divide la columna A, deberá multiplicar el valor de la columna B por dos. Al final se suman todos los valores bajo la columna B, que se encuentren al nivel de un numero impar de la columna A. Cree e invoque una función booleana de apoyo, que determine si un numero es par. Caso de ejemplo: $27 \times 82 = \mathbf{2214}$

A	B	sumandos
27	82	82
13	164	164
6	328	
3	656	656
1	1312	1312
$82 + 164 + 656 + 1312 = \mathbf{2214}$		

```

1 import Foundation
2
3 func esPar(_ n: Int) -> Bool {
4     return n % 2 == 0
5 }
6
7 func multiplicacionRusa(_ a: inout Int, _ b: inout Int) -> Int {
8     var suma = b
9     if a == 1 {
10         return suma
11     }
12     if !esPar(a) {
13         a = a / 2
14         b = b * 2
15         suma += multiplicacionRusa(&a, &b)
16     } else {
17         a = a / 2
18         b = b * 2
19         return multiplicacionRusa(&a, &b)
20     }
21
22     return suma
23 }
24
25 var a = 27, b = 82
26 print(multiplicacionRusa(&a, &b))

```

2.3.39. Escriba una función que reciba tres valores enteros y devuelva al mismo tiempo el mayor y el menor de ellos. Utilice el `_` para ignorar el nombre externo de cada parámetro.

Pruebas

a	b	c	mayor	menor
2	5	8	8	2
6	3	6	6	3

```
1 import Foundation
2
3 func mayorMenor(_ a: Int, _ b: Int, _ c: Int) -> (Int, Int) {
4     var mayor: Int = 0
5     var menor: Int = 0
6
7     if a > b {
8         if a > c {
9             mayor = a
10            if b > c {
11                menor = c
12            } else {
13                menor = b
14            }
15        } else {
16            mayor = c
17            menor = b
18        }
19    } else if b > a {
20        if b > c {
21            mayor = b
22            if a > c {
23                menor = c
24            } else {
25                menor = a
26            }
27        } else {
28            mayor = c
29            menor = a
30        }
31    } else {
32        //A y B son iguales
33        if a > c {
34            mayor = a
35            menor = c
36        } else {
37            mayor = c
38            menor = a
39        }
40    }
41
42    return (mayor, menor)
43 }
44
45 var tupla = mayorMenor(6, 3, 6)
46
47 print("Mayor: \(tupla.0) \nMenor: \(tupla.1)")
```

2.3.40. Suponga que se encuentra programando un juego en el cual el personaje camina en una cuadrícula, es decir, puede desplazarse un paso a la vez, ya sea hacia la derecha, izquierda, arriba o abajo. La posición inicial siempre es $(0, 0)$ y mediante un arreglo que contenga los pasos, debe calcular la posición final del personaje después de dar dichos pasos. Utilice una enumeración para definir cada una de las cuatro direcciones.

Pruebas

pasos	posición
arriba, arriba, izquierda, abajo, izquierda	$(-2, 1)$
arriba, arriba, izquierda, abajo, izquierda, abajo, abajo, derecha, derecha, abajo, derecha	$(1, -2)$
* Comience en $(5, 2)$ arriba, derecha, arriba, derecha, arriba, derecha, abajo, derecha	$(9, 4)$

```

1 import Foundation
2
3 enum Desplazamiento {
4     case arriba, abajo, izquierda, derecha
5 }
6
7 func moverPersonaje(_ pasos: [Desplazamiento], _ inicio: (Int, Int) = (0, 0)) -> (Int, Int) {
8     var x = inicio.0, y = inicio.1
9
10    for paso in pasos {
11        switch paso {
12            case .arriba: y += 1
13            case .abajo: y -= 1
14            case .derecha: x += 1
15            case .izquierda: x -= 1
16        }
17    }
18
19    return (x, y)
20 }
21
22 let pasos: [Desplazamiento] = [.arriba, .derecha, .arriba, .derecha, .arriba, .derecha, .abajo, .derecha]
23
24 var posicion = moverPersonaje(pasos, (5, 2))
25
26 print(posicion)

```

- 2.3.41.** 1) Defina una enumeración con tres miembros: *piedra*, *papel*, *tijera*.
 2) Defina otra enumeración con tres miembros: *gana*, *pierde*, *empata*.
 3) Escriba una función llamada *juego* que reciba lo que “elige” cada uno de los jugadores para jugar y que devuelva la respuesta correspondiente (de la segunda enumeración), respecto al primer jugador. Notas: Siempre juegan dos jugadores. No utilice valoraciones numéricas.

Pruebas

juego	resultado
piedra, tijera	gana
piedra, papel	pierde
tijera, tijera	empata

```

1 import Foundation
2
3 enum Chinchanpun {
4     case piedra, papel, tijera
5 }
6
7 enum Resultado {
8     case gana, pierde, empata
9 }
10
11 func juego(_ j1: Chinchanpun, _ j2: Chinchanpun) -> Resultado {
12
13     if j1 == .piedra && j2 == .papel {
14         return .pierde
15     } else if j1 == .piedra && j2 == .tijera {
16         return .gana
17     } else if j1 == .papel && j2 == .piedra {
18         return .gana
19     } else if j1 == .papel && j2 == .tijera {
20         return .pierde
21     } else if j1 == .tijera && j2 == .piedra {
22         return .pierde
23     } else if j1 == .tijera && j2 == .papel {
24         return .gana
25     } else {
26         return .empata
27     }
28 }
29
30 var resultado = juego(.tijera, .tijera)
31
32 print(resultado)

```

2.3.42. Tiene un arreglo de diccionarios. Cada diccionario tiene un par de valores, uno con el apellido y otro con el nombre de una persona. Debe entregar un arreglo que contenga cada nombre completo.

Pruebas

diccionarios	resultado
[{"apellido": "Perea", "nombre": "Alicia"}, {"apellido": "Flores", "nombre": "Norma"}, {"apellido": "Noriega", "nombre": "Roberto"}]	["Alicia Perea", "Norma Flores", "Roberto Noriega"]

```
1 import Foundation
2
3 let diccionarios: [[String: String]] =
4 [
5     ["apellido": "Perea", "nombre": "Alicia"],
6     ["apellido": "Flores", "nombre": "Norma"],
7     ["apellido": "Noriega", "nombre": "Roberto"]
8 ]
9
10 var resultado: [String] = []
11
12 for diccionario in diccionarios {
13     resultado.append("\(diccionario["nombre"]!) \(diccionario["apellido"]!)")
14 }
15
16 print(resultado)
```

2.3.43. Se tienen dos tuplas, cada una representa el numerador y el denominador de una fracción (en ese orden dentro de la tupla). Escriba una función que reciba ambas tuplas y que devuelva otra con la fracción resultante. Ejemplo: f1 = (34, 3); f2 = (11, 2); suma = (101, 6). Simplifique el resultado (cuando aplique)

$$\frac{a \cancel{\times} c}{\cancel{b} \cancel{\times} d} = \frac{(a \times d) + (b \times c)}{b \times d}$$

Pruebas

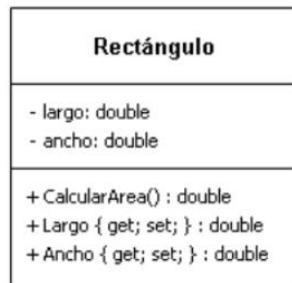
fracción 1	fracción 2	suma
5 / 8	17 / 9	181 / 72
1 / 5	1 / 6	11 / 30
3 / 12	3 / 6	3 / 4

```

1 import Foundation
2
3 func sumar(_ fraccion1: (numerador: Int, denominador: Int), _ fraccion2: (numerador: Int, denominador: Int)) -> (Int, Int) {
4     return ((fraccion1.numerador * fraccion2.denominador) + (fraccion1.denominador * fraccion2.numerador),
5             fraccion1.denominador * fraccion2.denominador)
6 }
7
8 func simplificar(_ n: Int, _ d: Int) -> (Int, Int) {
9     var divisor = 2, numerador = n, denominador = d
10    while((divisor <= numerador) && (divisor <= denominador)) {
11        while((numerador % divisor == 0) && (denominador % divisor == 0)) {
12            numerador = numerador / divisor;
13            denominador = denominador / divisor;
14        }
15        divisor += 1
16    }
17    return (numerador, denominador)
18 }
19
20 let fraccion1 = (3, 12), fraccion2 = (3, 6)
21 var resultado = sumar(fraccion1, fraccion2)
22 resultado = simplificar(resultado.0, resultado.1)
23
24 print("\(fraccion1.0) / \(fraccion1.1) + \(fraccion2.0) / \(fraccion2.1) = \(resultado.0) / \(resultado.1)" )
25

```

2.3.44. Hay una pared rectangular que tiene una ventana rectangular; se requiere un programa que dadas las dimensiones (largo y ancho) de la ventana y de la pared, imprima los minutos necesarios para pintar la pared, sabiendo que se puede pintar 1 m² en 10 minutos. Utilice la siguiente clase:



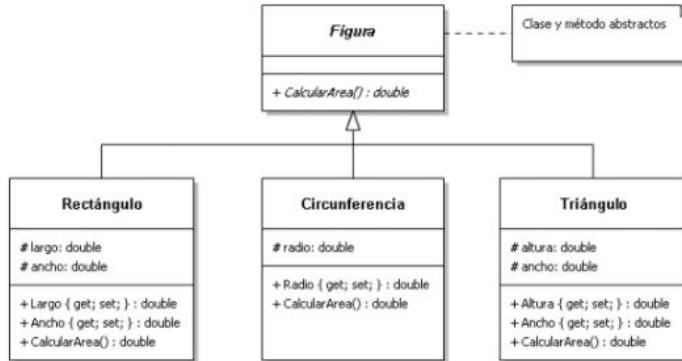
Pruebas				
pared		ventana		resultado
largo	ancho	largo	ancho	
10	5	2	1	480
3	3	1	1	80

```

1 import Foundation
2
3 class Rectangulo {
4     private var largo: Int = 0
5     private var ancho: Int = 0
6
7     var Largo: Int {
8         get { return largo }
9         set { largo = newValue }
10    }
11    var Ancho: Int {
12        get { return ancho }
13        set { ancho = newValue }
14    }
15
16    init(_ largo: Int, _ ancho: Int) {
17        self.largo = largo
18        self.ancho = ancho
19    }
20
21    func calcularArea() -> Int{
22        return largo * ancho
23    }
24 }
25
26 var pared = Rectangulo(10, 5)
27 var ventana = Rectangulo(2, 1)
28 let tiempoTotal = (pared.calcularArea() - ventana.calcularArea()) * 10
29
30 print("El tiempo necesario para pintar la pared es de \$(tiempoTotal) min")

```

2.3.45. Diseñe un programa que calcule el área de las figuras geométricas señaladas en el diagrama UML que se muestra a continuación. Utilice polimorfismo para la reutilización de métodos abstractos.



Pruebas

rectángulo			circunferencia		triángulo		
largo	ancho	área	radio	área	altura	ancho	área
6	8	48	14	615.75	4	3	6
15	4	60	8	201.06	12	10	60

```

1 import Foundation
2
3 class Figura {
4     func calcularArea() -> Double{
5         fatalError("Se debe sobreescribir este método")
6     }
7 }
8
9 class Rectangulo : Figura {
10    private var largo: Double = 0.0
11    private var ancho: Double = 0.0
12
13    var Largo: Double {
14        get { return largo }
15        set { largo = newValue }
16    }
17    var Ancho: Double {
18        get { return ancho }
19        set { ancho = newValue}
20    }
21
22    init(_ largo: Double, _ ancho: Double) {
23        self.largo = largo
24        self.ancho = ancho
25    }
26
27    override func calcularArea() -> Double{
28        return largo * ancho
29    }
30 }
31
32 class Circunferencia : Figura {
33     private var radio: Double = 0.0
  
```

```

32  class Circunferencia : Figura {
33      private var radio: Double = 0.0
34
35      var Radio: Double {
36          get { return radio }
37          set { radio = newValue }
38      }
39
40      init(_ radio: Double) {
41          self.radio = radio
42      }
43
44      override func calcularArea() -> Double {
45          return Double.pi * radio * radio
46      }
47  }
48
49  class Triangulo : Figura {
50      private var altura: Double = 0.0
51      private var ancho: Double = 0.0
52
53      var Altura: Double {
54          get { return altura }
55          set { altura = newValue }
56      }
57      var Ancho: Double {
58          get { return ancho }
59          set { ancho = newValue }
60      }
61
62      init(_ altura: Double, _ ancho: Double) {
63          self.altura = altura
64          self.ancho = ancho
65      }
66
67      override func calcularArea() -> Double {
68          return (altura * ancho) / 2
69      }
70  }
71
72  var rectangulo = Rectangulo(15, 4)
73  var circunferencia = Circunferencia(8)
74  var triangulo = Triangulo(12, 10)
75
76  print("Área del rectángulo: \(rectangulo.calcularArea())")
77  print("Área de la circunferencia: \(circunferencia.calcularArea())")
78  print("Área del triángulo: \(triangulo.calcularArea())")
79

```

2.3.46. Una agencia vendedora de autos desea administrar los datos de sus vehículos y clasificarlos por tipo. Todos los autos tienen los siguientes datos:

- Número de serie del motor
- Marca
- Año
- Precio

Los vehículos se clasifican en autos compactos, autos de lujo, camionetas y vagonetas. Para los autos y vagonetas, también es importante almacenar la cantidad de pasajeros; mientras que para las camionetas se debe controlar la capacidad de carga en kgs. y la cantidad de ejes y de rodadas. Modele este sistema e instancie cada una de las clases, asignándole datos mediante sus respectivas propiedades. Agregue un constructor con parámetros a cada clase para inicializar sus datos e invoque el constructor de la clase base desde el constructor de cada clase derivada.

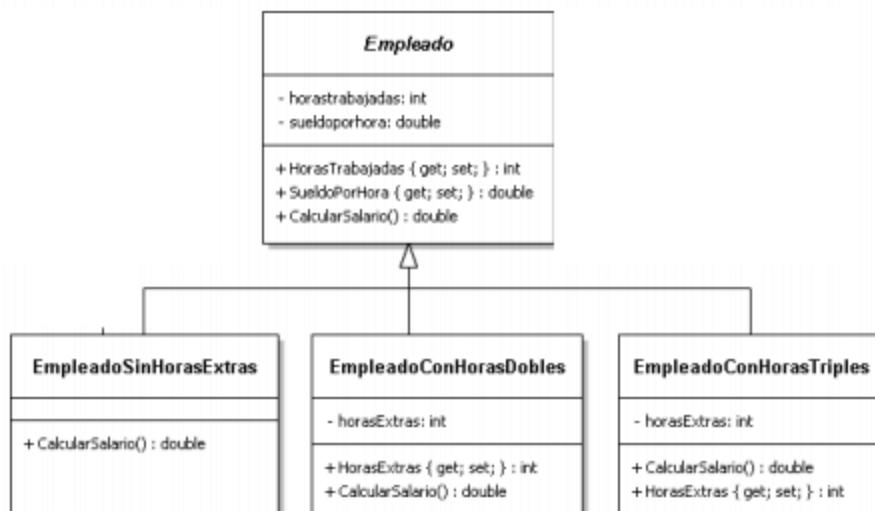
```
1 import Foundation
2
3 class Auto {
4     private var numeroSerieMotor: String = ""
5     private var marca: String = ""
6     private var año: Int = 0
7     private var precio: Double = 0.0
8
9     var NumeroSerieMotor: String {
10         get { return numeroSerieMotor }
11         set { numeroSerieMotor = newValue }
12     }
13     var Marca: String {
14         get { return marca }
15         set { marca = newValue }
16     }
17     var Año: Int {
18         get { return año }
19         set { año = newValue }
20     }
21     var Precio: Double {
22         get { return precio }
23         set { precio = newValue }
24     }
25
26     init(_ numeroSerieMotor: String, _ marca: String, _ año: Int, _ precio: Double) {
27         self.numeroSerieMotor = numeroSerieMotor
28         self.marca = marca
29         self.año = año
30         self.precio = precio
31     }
32 }
```

2.3.47. Calcule el salario semanal de un empleado en base a las horas trabajadas, el sueldo por hora y los siguientes criterios:

- Si las horas trabajadas son más de 40, entonces el excedente se considera hora extra
- Si las horas totales se ubican entre 41 y 45, entonces cada hora extra se paga el doble
- Si las horas totales son más de 45, entonces cada hora extra se paga el triple

Pruebas

horas trabajadas	salario p/hora	horas extra	salario semanal
40	2	0	80
43	2	3	92
46	2	6	116



```
1 import Foundation
2
3 class Empleado {
4     private var horasTrabajadas: Int
5     private var sueldoPorHora: Double
6
7     var HorasTrabajadas: Int {
8         get { return horasTrabajadas }
9         set { horasTrabajadas = newValue}
10    }
11    var SueldoPorHora: Double {
12        get { return sueldoPorHora }
13        set { sueldoPorHora = newValue}
14    }
15
16    init(_ horasTrabajadas: Int, _ sueldoPorHora: Double) {
17        self.horasTrabajadas = horasTrabajadas
18        self.sueldoPorHora = sueldoPorHora
19    }
20
21    func calcularSueldo() -> Double {
22        return Double(horasTrabajadas) * sueldoPorHora
23    }
24}
25
26 class EmpleadoSinHorasExtra : Empleado {
27     override func calcularSueldo() -> Double {
28         return super.calcularSueldo()
29     }
30 }
31
```

```

31
32 class EmpleadoConHorasDobles : Empleado {
33     var horasExtra: Int {
34         get { return horasTrabajadas - 40 }
35     }
36
37     override func calcularSueldo() -> Double {
38         return Double(horasTrabajadas - horasExtra) * 2 + (Double(horasExtra) * 2 * 2)
39     }
40 }
41
42 class EmpleadoConHorasTriples : Empleado {
43     var horasExtra: Int {
44         get { return horasTrabajadas - 40 }
45     }
46
47     override func calcularSueldo() -> Double {
48         return Double(horasTrabajadas - horasExtra) * 2 + (Double(horasExtra) * 2 * 3)
49     }
50 }
51
52 var horasTrabajadas = 46, sueldoPorHora = 2.0
53
54
55 if horasTrabajadas <= 40 {
56     var empleado = EmpleadoSinHorasExtra(horasTrabajadas, sueldoPorHora)
57     print("Horas extra: 0\nSalario Semanal: \(empleado.calcularSueldo())")
58 } else if 41...45 ~= horasTrabajadas {
59     var empleado = EmpleadoConHorasDobles(horasTrabajadas, sueldoPorHora)
60     print("Horas extra: \(empleado.horasExtra)\nSalario Semanal: \(empleado.calcularSueldo())")
61 } else {
62     var empleado = EmpleadoConHorasTriples(horasTrabajadas, sueldoPorHora)

```

2.3.48. Una agencia de renta de vehículos dispone de autobuses y tractores.

Debe crear una clase abstracta llamada Vehículo que contenga:

- Propiedades:
 - Número de placa
- Métodos
 - Calcular importe
 - Mostrar datos

La renta de autobuses se factura por kilómetros. Debido a esto, los datos de la clase Autobús son:

- El precio por kilómetro.
- La cantidad de kilómetros que tiene el autobús cuando se renta.
- La cantidad de kilómetros que tiene el autobús cuando se devuelve.

En cambio, la renta de tractores se factura por días. Debido a esto, los datos de la clase Tractor son:

- El precio por día.
- El día del mes actual en que lo está rentando.
- El día del mes actual en que lo está devolviendo.
 - Suponga que la renta de un tractor solo debe realizarse en el mismo mes

Cuando se renta un vehículo, se deben capturar sus placas, sus datos correspondientes (de acuerdo con el tipo de vehículo).

Cuando se devuelve un vehículo, se calcula el importe a pagar por la renta y se muestran sus datos.

```
1 import Foundation
2
3 class Vehiculo {
4     private var numeroPlaca: String
5
6     var NumeroPlaca: String {
7         get { return numeroPlaca }
8         set { numeroPlaca = newValue }
9     }
10
11    init(_ numeroPlaca: String) {
12        self.numeroPlaca = numeroPlaca
13    }
14
15    func calcularImporte() -> Double {
16        fatalError("Se debe sobreescribir este método")
17    }
18
19    func mostrarDatos() -> String {
20        return "Número de Placa: \(numeroPlaca)"
21    }
22 }
23
24 class Autobus : Vehiculo {
25     private var precioPorKm: Double
26     private var kmIniciales: Double
27     private var kmFinales: Double
28
29     var PrecioPorKm: Double {
30         get { return precioPorKm }
31         set { precioPorKm = newValue }
32     }
33     var KmIniciales: Double {
```

```

52
53     var KmIniciales: Double {
54         get { return kmIniciales }
55         set { kmIniciales = newValue }
56     }
57     var KmFinales: Double {
58         get { return kmFinales }
59         set { kmFinales = newValue }
60     }
61
62     init(_ precioPorKm: Double, _ kmIniciales: Double, _ kmFinales: Double, _ numeroPlaca: String) {
63         self.precioPorKm = precioPorKm
64         self.kmIniciales = kmIniciales
65         self.kmFinales = kmFinales
66         super.init(numeroPlaca)
67     }
68
69     override func calcularImporte() -> Double {
70         return (kmFinales - kmIniciales) * precioPorKm
71     }
72
73     override func mostrarDatos() -> String {
74         return "\u{super.mostrarDatos()}\tPrecio por Km: \u{(precioPorKm)}\tKm Iniciales: \u(kmIniciales)\tKm Finales: \u(kmFinales)\nCantidad de Km recorridos: \u(kmFinales - kmIniciales)"
75     }
76 }

```

```

59     class Tractor : Vehiculo {
60         private var precioPorDia: Double
61         private var diaInicial: Int
62         private var diaFinal: Int
63
64         var PrecioPorDia: Double {
65             get { return precioPorDia }
66             set { precioPorDia = newValue }
67         }
68         var DiaInicial: Int {
69             get { return diaInicial }
70             set { diaInicial = newValue }
71         }
72         var DiaFinal: Int {
73             get { return diaFinal }
74             set { diaFinal = newValue }
75         }
76
77         init(_ precioPorDia: Double, _ diaInicial: Int, _ diaFinal: Int, _ numeroPlaca: String) {
78             self.precioPorDia = precioPorDia
79             self.diaInicial = diaInicial
80             self.diaFinal = diaFinal
81             super.init(numeroPlaca)
82         }
83
84         override func calcularImporte() -> Double {
85             return Double(diaFinal - diaInicial) * precioPorDia
86         }
87
88         override func mostrarDatos() -> String {
89             return "\u{super.mostrarDatos()}\tPrecio por Día: \u{(precioPorDia)}\tDía de Inicio de Renta: \u(diaInicial)\tDía de Devolución: \u(diaFinal)\nDías Totales: \u(diaFinal - diaInicial)"
90         }
91     }

```

```
92     }
93
94     var vehiculo: Vehiculo
95     let tipoVehiculo = false
96
97     //Cuando es true es autobus, sino es tractor
98     if tipoVehiculo {
99         vehiculo = Autobus(100.50, 10.2, 15.0, "Autobus")
100    } else {
101        vehiculo = Tractor(54.32, 5, 8, "Tractor")
102    }
103
104    print("Importe total a pagar: \\" + (vehiculo.calcularImporte()))
105    print(vehiculo.mostrarDatos())
106
```

2.3 PRACTICAS DE PROGRAMACIÓN

Miguel Ángel Méndez Cruz



Problema:

2.3.1. Se tiene la edad de Loky (en años perro) en una variable. Determine la edad de Loky en años humanos, considerando que 1 año humano equivale a 7 años perro.

Solución:

```
import Foundation

let EdadLokyPerro = 50
var EdadLokyHumano = EdadLokyPerro / 7
print(EdadLokyHumano)
```

Problema:

2.3.2. Se tiene el tiempo de un recorrido, almacenado en tres variables: una para la hora, otra para los minutos y otra para los segundos; así también, se tiene la distancia (en metros) en otra variable. Despliegue la velocidad en metros/segundo, kilómetros/hora y millas/hora.

Solución:

```
let Horas: Float
Horas = 1
let Minutos: Float
Minutos = 35
let Segundos: Float
Segundos = 56
let DistanciaMetros: Float
DistanciaMetros = 50000
//Metros sobre segundos
let Velocidad: Float
Velocidad = (DistanciaMetros) / ((Horas * 60 * 60)+(Minutos * 60) + (Segundos))
print(Velocidad)
//Kilometros sobre horas
let VelocidadKmPH: Float
VelocidadKmPH = (DistanciaMetros / 1000) / ((Horas) + (Minutos / 60) + ((Segundos / 60) / 60))
print(VelocidadKmPH)
//Millas sobre horas
let VelocidadMillas: Float
VelocidadMillas = (DistanciaMetros / 1609.34) / ((Horas) + (Minutos / 60) + ((Segundos / 60) / 60))
print(VelocidadMillas)
```

Problema:

2.3.3. Dadas la suma y la diferencia entre dos números, encuentre los valores de dichos números y almacénelos en variables llamadas a y b . Imprima los resultados.

Solución:

```
let Suma = 11
let Diferencia = 3
var VarA = (Suma + Diferencia) / 2
var VarB = VarA - Diferencia
print(VarA)
print(VarB)
```

Problema:

2.3.4. Se tienen cuatro variables ($ancho$, $alto$, x , y), que describen las dimensiones de una figura en forma de **L**. Calcule, almacene e imprima el perímetro y el área de dicha figura.

Solución:

```
let Ancho = 8
let Alto = 12
let X = 4
let Y = 3

var perimetro = Ancho + Alto + (Alto - Y) + (Ancho - X) + X + Y
var area = (Ancho * Y) + ((Alto - Y) * X)

print(perimetro)
print(area)
```

Problema:

2.3.5. Dado un número de tres dígitos almacenado en *a*, calcule aritméticamente e imprima el último dígito.

Solución:

```
var a = 123
var res = a % 10
print(res)
```

Problema:

2.3.6. En *x* años a partir de ahora, Anahí será *y* veces mayor en edad que su hermana Abril. Conociendo la edad actual de Abril, calcule la edad de Anahí.

Solución:

```
var equis = 3
var ygriega = 2
var edadAbril = 12

var edadAnahi = edadAbril * ygriega + equis
print(edadAnahi)
```

Problema:

2.3.7. Suponga que usted tiene x manzanas. Lizbeth intercambia 3 naranjas por 5 manzanas. ¿Cuántas naranjas puede obtener de Lizbeth y cuantas manzanas le quedarían después del intercambio? Imprima los resultados.

Solución:

```
var manzanas = 17
var intercambio = (manzanas / 5) * 3

print(intercambio)
```

Problema:

2.3.8. Dados dos números en a y b , determine e imprima el mayor de ellos o indique si son iguales.

Solución:

```
var a = 11
var b = 22

if ( a == b)
{
    print("Iguales")
} else if (a > b)
{
    print("El mayor es a")
} else
{
    print("El mayor es b")
```

Problema:

2.3.9. Imprima si el valor en *x* es par o impar.

Solución:

```
var x = 38
var division = x % 2

if division == 0
{
    print("Es par")
}
else
{
    print("Es impar")
}
```

Problema:

2.3.10. Tiene dos números en *a* y *b*. Indique si *a* es divisible entre *b*.

Solución:

```
var letA = 22
var letB = 11

if(letA % letB) == 0
{
    print("Es divisible")
}
else
{
    print("No es divisible")
}
```

Problema:

2.3.11. Se tienen tres variables (a, b, c). Imprima la leyenda “Al menos dos variables son iguales” o bien la leyenda “Todas las variables son diferentes”, según sea el caso.

Solución:

```
var aaa = 1
var bbb = 2
var ccc = 3

if(aaa == bbb || bbb == ccc || aaa == ccc )
{
    print("al menos dos variables son iguales")
}
else
{
    print("todas las variables son diferentes")
}
```

Problema:

2.3.12. Antes de preparar el desayuno, debemos saber si aún se pueden cocinar los ingredientes. Considere que los blanquillos se echan a perder después de tres semanas (21 días) y el tocino después de una semana (7 días). Dados los días de los blanquillos y el tocino en su refrigerador, determine si ambos se pueden cocinar, o si hay alguno que deba desecharse.

Solución:

```
var blanquillos = 2
var tocino = 3

if(blanquillos < 21 && tocino < 7)
{
    print("ambos se pueden cocinar")
}else if(blanquillos >= 21 && tocino < 7 )
{
    print("debe desechar el blanquillo")
} else if( blanquillos < 21 && tocino >= 7)
{
    print("debe desechar el tocino")
}
else
{
    print("debe desechar ambas")
}
```

Problema:

2.3.13. Dado un año, imprima si es o no es bisiesto. Las reglas (para este ejercicio) para determinar un año bisiesto, son las siguientes:

- Divisible entre 4
- Cada inicio de siglo, si el año es divisible entre 400

Solución:

```
var bisiesto = 2016

if( bisiesto % 4 == 0 && bisiesto % 100 != 0 && bisiesto % 400 != 0)
{
    print("Es bisiesto")
} else
{
    print ("No es bisiesto")
}
```

Problema:

2.3.14. La función *Int.random(in: mínimo...máximo)* entrega un valor aleatorio, comprendido entre mínimo y máximo. Utilice dicha función para simular que lanza un volado con una moneda y determine si cayó águila (par) o sello (impar).

Solución:

```
let random = Int.random(in: 1...100)

if random % 2 == 0 {
    print("Es aguila")
} else {
    print("Es sello")
}
```

Problema:

2.3.15. Determine si un punto o coordenada (x, y) se encuentra dentro de un rectángulo definido por las coordenadas de la esquina inferior izquierda ($x1, y1$) y la esquina superior derecha ($x2, y2$).

Solución:

```

let xx = 1
let yy = 2
let x1 = 1
let y1 = 1
let x2 = 3
let y2 = 3

if( xx >= x1 && xx <= x2 && yy >= y1 && yy <= y2)
{
    | print("Dentro")
} else {
    print("fuera")
}

```

Problema:

2.3.16. Dado un carácter, determine si es vocal, consonante, dígito, operador aritmético u otro.

Solución:

```

let char: UnicodeScalar = "a"
let valor = char.value

if (char == "a" || char == "e" || char == "i" || char == "o" || char == "u" ||
char == "A" || char == "E" || char == "I" || char == "O" || char == "U")
{
    print("\u{char} es vocal")
} else if 65...90 ~= valor || 97...122 ~= valor
{
    print("\u{char} es consonante")
} else if 48...57 ~= valor
{
    print("\u{char} es un dígito")
} else if ["+", "-", "/", "*"].contains(char)
{
    print("\u{char} es un operador aritmético")
} else
{
    print("otro")
}

```

Problemas:

2.3.17. En base al mes y día de nacimiento, determine el signo zodiacal de una persona.

Solución:

```
var mes = 1
var dia = 28

if mes == 1 && 20...31 ~= dia || mes == 2 && 1...18 ~= dia
{
    print("Acuario")
}
else if mes == 2 && 19...28 ~= dia || mes == 3 && 1...20 ~= dia
{
    print("Piscis")
}
else if mes == 3 && 21...31 ~= dia || mes == 4 && 1...20 ~= dia
{
    print("Aries")
}
else if mes == 4 && 20...30 ~= dia || mes == 5 && 1...20 ~= dia
{
    print("Tauro")
}

else if mes == 5 && 21...31 ~= dia || mes == 6 && 1...20 ~= dia
{
    print("Géminis")
}
else if mes == 6 && 21...30 ~= dia || mes == 7 && 1...22 ~= dia
{
    print("Cáncer")
}
else if mes == 7 && 23...31 ~= dia || mes == 8 && 1...22 ~= dia
{
    print("Leo")
}
else if mes == 8 && 23...31 ~= dia || mes == 9 && 1...22 ~= dia
{
    print("Virgo")
}
```

```
else if mes == 9 && 23...30 ~= dia || mes == 10 && 1...22 ~= dia
{
    print("Libra")
}
else if mes == 10 && 23...31 ~= dia || mes == 11 && 1...21 ~= dia
{
    print("Escorpio")
}
else if mes == 11 && 22...30 ~= dia || mes == 12 && 1...21 ~= dia
{
    print("Sagitario")
}
else if mes == 12 && 22...31 ~= dia || mes == 1 && 1...19 ~= dia
{
    print("Capricornio")
}
```

Problema:

2.3.18. Un número cuadrado perfecto es el producto de multiplicar un entero por sí mismo. Imprima los N cuadrados perfectos.

Solución:

```
let N = 5

for cuadrado in 1...N {
    print(cuadrado * cuadrado)
}
```

Problema:

2.3.19. Imprima la serie de números de 1 a N alternando su orden, es decir, el primer número siempre es el 1, seguido de N , luego 2, luego $N-1$ y así sucesivamente. La serie debe imprimirse en un solo renglón, separando los números con espacios.

Solución:

```
let n = 4
var a = 0
var out: String = ""
for i in 1...n
{
    if i % 2 == 0
    {
        out += "\((n - (a - 1)) ) "
    } else
    {
        out += "\((a + 1) ) "
        a = a + 1
    }
}
print(out)
```

Problema:

2.3.20. Dado N , dibuje una pirámide de asteriscos. La pirámide debe tener N líneas. En la i -ésima línea debe haber $N-i$ espacios, seguido de i^2-1 asteriscos.

Solución:

```
var N = 4
print("piramide")
for i in 1...N
{
    let espacios = N - i
    let asteriscos = (i * 2) - 1
    var linea = ""

    if espacios > 0
    {
        linea = String(repeating: " ", count: espacios)
    }
    linea += String(repeating: "*", count: asteriscos)
    print(linea)
}
```

Problema:

2.3.21. Imprima los primeros N números de la serie de *Fibonacci*. Los primeros dos números de la serie siempre son 1, el resto son la suma de los dos anteriores. Imprima la serie en un solo renglón con los valores separados por coma.

Solución:

```
var N = 3
var a1 = 0
var a2 = 1
var Fibonacci: String = "1 "
for _ in 1..<N
{
    let aux = a1 + a2
    a1 = a2
    a2 = aux
    Fibonacci += "\,(aux) "
}
print(Fibonacci)
```

Problema:

2.3.22. Dado un número en x , determine si es un número primo. Los números primos solo pueden dividirse entre 1 y entre sí mismos (división exacta).

Solución:

```
var num = 2
var cont = 2
var primo = true
while (primo) && (cont != num)
{
    if (num == 1)
    {
        primo = false
    } else if (num % cont == 0)
    {
        primo = false
    }
    cont += 1
}
if primo {
    print("Es primo")
} else [
    print("No es primo")
]
```

Problema:

2.3.23. Convierta un número de decimal a binario con el método de escalera.

Solución:

```
var d = 78
var b: String = ""
while d > 0 {
    b = d % 2 == 0 ? "0" + b : "1" + b;
    d = d / 2
}
print(b)
```

Problema:

2.3.24. Imprima el elemento mayor de un arreglo de N números.

Solución:

```
let a = [10,12,33,11,1,8]
let mayor = a.max()
print(mayor!)
```

Problema:

2.3.25. Imprima los elementos de un vector en orden inverso al que se encuentran almacenados.

Solución:

```
let v = [10, 12, 33, 11, 1, 8]
var out = ""
for i in v.reversed()
{
    out += i == v[0] ? "\u2028(i)" : "\u2028(i), "
}
print(out)
```

Problema:

2.3.26. Invierta los elementos de un arreglo sin crear otro.

Solución:

```
let v = [1,2,3,10,100]
var out = ""
for i in v.reversed()
{
    out += i == v[0] ? "\\\(i)" : "\\(i), "
}
print(out)
```

Problema:

2.3.27. Dados los vectores A y B , imprima todos los elementos de B que se encuentren en A . Si no existen elementos en común, no debe imprimir nada.

Solución:

```
let v = [1,2,3,10,100]
var out = ""
for i in v.reversed()
{
    out += i == v[0] ? "\\\(i)" : "\\(i), "
}
print(out)
```

Problema:

2.3.28. Extraiga cada dígito, de izquierda a derecha, de un número dado en n y almacénelo en un vector. Resuélvalo aritméticamente, es decir, sin convertir n a cadena de caracteres.

Solución:

```
var n = 12345
var a: [Int] = []

a.append(n % 10)

while n >= 10
{
    n = n / 10
    a.insert(n % 10, at: 0)
}
print(a)
```

Problema:

2.3.29. Dadas las matrices

$$\mathbf{A} = \begin{pmatrix} 2 & 0 & 1 \\ 3 & 0 & 0 \\ 5 & 1 & 1 \end{pmatrix}; \quad \mathbf{B} = \begin{pmatrix} 1 & 0 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 0 \end{pmatrix}$$

Calcular:

a) $\mathbf{A} + \mathbf{B}$

Solución:

```
let a: [[Int]] = [[2, 0, 1],  
                  [3, 0, 0],  
                  [5, 1, 1]]  
  
let b: [[Int]] = [[1, 0, 1],  
                  [1, 2, 1],  
                  [1, 1, 0]]  
  
let c: [[Int]] = [[a[0][0] + b[0][0], a[0][1] + b[0][1], a[0][2] + b[0][2]],  
                  [a[1][0] + b[1][0], a[1][1] + b[1][1], a[1][2] + b[1][2]],  
                  [a[2][0] + b[2][0], a[2][1] + b[2][1], a[2][2] + b[2][2]]]  
  
print(c)
```

Problema:

b) $\mathbf{A} - \mathbf{B}$

Solución:

```
let a: [[Int]] = [[2, 0, 1],  
                  [3, 0, 0],  
                  [5, 1, 1]]  
  
let b: [[Int]] = [[1, 0, 1],  
                  [1, 2, 1],  
                  [1, 1, 0]]  
  
let c: [[Int]] = [[a[0][0] - b[0][0], a[0][1] - b[0][1], a[0][2] - b[0][2]],  
                  [a[1][0] - b[1][0], a[1][1] - b[1][1], a[1][2] - b[1][2]],  
                  [a[2][0] - b[2][0], a[2][1] - b[2][1], a[2][2] - b[2][2]]]  
  
print(c)
```

Problema:

c) $A \times B$

Solución:

```
let a: [[Int]] = [[2, 0, 1],  
                  [3, 0, 0],  
                  [5, 1, 1]]  
let b: [[Int]] = [[1, 0, 1],  
                  [1, 2, 1],  
                  [1, 1, 0]]  
let c: [[Int]] = [  
    [  
        a[0][0] * b[0][0] + a[0][1] * b[1][0] + a[0][2] * b[2][0],  
        a[0][0] * b[0][1] + a[0][1] * b[1][1] + a[0][2] * b[2][1],  
        a[0][0] * b[0][2] + a[0][1] * b[1][2] + a[0][2] * b[2][2],  
    ],  
    [  
        a[1][0] * b[0][0] + a[1][1] * b[1][0] + a[1][2] * b[2][0],  
        a[1][0] * b[0][1] + a[1][1] * b[1][1] + a[1][2] * b[2][1],  
        a[1][0] * b[0][2] + a[1][1] * b[1][2] + a[1][2] * b[2][2],  
    ],  
    [  
        a[2][0] * b[0][0] + a[2][1] * b[1][0] + a[2][2] * b[2][0],  
        a[2][0] * b[0][1] + a[2][1] * b[1][1] + a[2][2] * b[2][1],  
        a[2][0] * b[0][2] + a[2][1] * b[1][2] + a[2][2] * b[2][2],  
    ]]  
print(c)
```

Problema:

2.3.30. Dada una matriz cuadrada A , almacene los elementos de la diagonal principal y los de la diagonal inversa, en vectores llamados DP y DI respectivamente.

Solución:

```
let arreglo: [[Int]] = [
    [3, 5, 8, 2],
    [2, 7, 9, 5],
    [2, 8, 9, 2],
    [4, 6, 7, 1]
]
var dp: [Int] = []
var di: [Int] = []
var a: Int = 1
for (i, items) in arreglo.enumerated()
{
    dp.append(items[i])
    di.append(items[items.endIndex - 1 * a])
    a += 1
}
print(dp)
print(di)
```

Problema:

2.3.31. Dada una matriz cuadrada A , imprima el resultado de sumar los elementos que no corresponden a la periferia de la matriz.

Solución:

```
let a: [[Int]] = [
    [3, 5, 8, 9, 2],
    [1, 4, 2, 1, 0],
    [4, 5, 4, 8, 1],
    [9, 8, 1, 0, 3],
    [7, 2, 1, 1, 3]
]
var sum: Int = 0
for (i, items) in a.enumerated()
{
    if i != 0 && i != a.endIndex - 1
    {
        var b = 0
        for item in items
        {
            if b != 0 && b != items.endIndex - 1
            {
                sum += item
            }
            b += 1
        }
    }
}
print(sum)
```

Problema:

2.3.32. Escriba un método que reciba dos valores enteros y devuelva el mayor de ellos. Utilice el `_` para ignorar el nombre externo de cada parámetro.

Solución:

```
let num1 = 2
let num2 = 5
func mayor(_ num1: Int, _ num2: Int) -> Int
{
    if( num1 > num2 )
    {
        return num1
    }
    else
    {
        return num2
    }
}
print(mayor(num1, num2))
```

Problema:

2.3.33. Escriba un método que reciba un número entero y devuelva un vector con los valores de 1 a N .

Solución:

```
func v(_ n: Int) -> [Int]
{
    var a: [Int] = []
    if n == 0
    {
        return a
    }
    for i in 1...n
    {
        a.append(i)
    }
    return a
}
print(v(21))
```

Problema:

2.3.34. Programe un método que reciba cuatro parámetros: hora inicial, minutos iniciales, hora final y minutos finales. La función debe calcular los minutos transcurridos entre los dos horarios compuestos de horas y minutos. Si los minutos se omiten en la invocación, se considerará que el valor es cero.

Solución:

```
func tiempo(hi: Int, mihi: Int = 0, hf: Int, mifi: Int = 0) -> Int
{
    let desde: Int = (hi * 60) + mihi
    let hasta: Int = (hf * 60) + mifi

    return hasta - desde
}

print(tiempo(hi: 8, mihi: 10, hf: 17, mifi: 30))
```

Problema:

2.3.35. Una pila es una estructura de datos (arreglo unidimensional), que puede realizar lo siguiente:

- Agregar un elemento al principio del arreglo
- Obtener el valor del primer elemento del arreglo
- Entregar el primer elemento del arreglo y removerlo (sacarlo de la pila)

Debe crear un método para cada operación descrita anteriormente, mismas que deberán trabajar sobre un arreglo pasado como parámetro por referencia.

Solución:

```
func agregarPrimero(_ arreglo: inout [Int], _ valor: Int)
{
    arreglo.insert(valor, at: 0)
}
func eliminarPrimero(_ arreglo: inout [Int]) -> Int
{
    if arreglo.count > 0
    {
        let b: Int = arreglo[0]
        arreglo.remove(at: 0)
        return b
    }
    return 0
}
func obtenerPrimero(_ arreglo: inout [Int]) -> Int
{
    return arreglo[0]
}

var resultado: [Int] = []
agregarPrimero(&resultado, 2)
agregarPrimero(&resultado, 1)
agregarPrimero(&resultado, 8)
print(resultado)
print(eliminarPrimero(&resultado))
print(obtenerPrimero(&resultado))
```

Problema:

2.3.36. Escriba los métodos para agregar y entregar un elemento bajo el concepto de la estructura de dato llamada cola. Utilice el arreglo como parámetro por referencia.

Solución:

```
func agregarFinal(_ arreglo: inout [Int], _ valor: Int)
{
    arreglo.append(valor)
}
func eliminarPrimero(_ arreglo: inout [Int])
{
    if arreglo.count > 0 {
        arreglo.remove(at: 0)
    }
}
func obtener(_ arreglo: inout [Int]) -> Int
{
    return arreglo[0]
}

var resultado: [Int] = []
agregarFinal(&resultado, 2)
agregarFinal(&resultado, 1)
agregarFinal(&resultado, 8)
print(resultado)
eliminarPrimero(&resultado)
print(resultado)
print([obtener(&resultado)])
```

Problema:

2.3.37. Escriba un método que calcule la factorial de un número dado como parámetro. Utilice recursividad.

Solución:

```
func factorial(_ i: Int) -> Int
{
    if i == 0
    {
        return 1
    }
    return i * factorial(i - 1)
}
print(factorial(5))
```

Problema:

2.3.39. Escriba una función que reciba tres valores enteros y devuelva al mismo tiempo el mayor y el menor de ellos. Utilice el `_` para ignorar el nombre externo de cada parámetro.

Solución:

```
func mayorYmenor(_ a: Int, _ b: Int, _ c: Int) -> (Int, Int) {
    var mayor: Int
    var menor: Int
    //Comparaciones
    if a > b
    {
        if a > c
        {
            mayor = a
            if b > c
            {
                menor = c
            } else
            {
                menor = b
            }
        } else
        {
            mayor = b
        }
    } else
    {
        mayor = b
        if a > c
        {
            menor = c
        } else
        {
            menor = a
        }
    }
    return (mayor, menor)
}
```

```

        {
            mayor = c
            menor = b
        }
    } else if b > a
    {
        if b > c
        {
            mayor = b
            if a > c
            {
                menor = c
            } else
            {
                menor = a
            }
        } else
        {
            mayor = c
            menor = a
        }
    } else
    {
        if a > c
        {
            mayor = a
            menor = c
        } else
        {
            mayor = c
            menor = a
        }
    }
    return (mayor, menor)
}
var resultado = mayorYmenor(2, 1, 8)
print("El mayor es \$(resultado.0) y el menor es \$(resultado.1)")

```

Problema:

- 2.3.41.** 1) Defina una enumeración con tres miembros: *piedra*, *papel*, *tijera*.
2) Defina otra enumeración con tres miembros: *gana*, *pierde*, *empata*.
3) Escriba una función llamada *juego* que reciba lo que “elige” cada uno de los jugadores para jugar y que devuelva la respuesta correspondiente (de la segunda enumeración), respecto al primer jugador. Notas: Siempre juegan dos jugadores. No utilice valoraciones numéricas.

Solución:

```
enum Juego
{
    case piedra
    case papel
    case tijera
}
enum resultados
{
    case gana
    case empate
    case pierde
}
func partida(_ azul: Juego, _ rojo: Juego) -> resultados
{
    if azul == .piedra && rojo == .papel
    {
        return .pierde
    }
}
```

```
        } else if azul == .piedra && rojo == .tijera
    {
        return .gana
    } else if azul == .papel && rojo == .piedra
    {
        return .gana
    } else if azul == .papel && rojo == .tijera
    {
        return .pierde
    } else if azul == .tijera && rojo == .piedra
    {
        return .pierde
    } else if azul == .tijera && rojo == .papel
    {
        return .gana
    }
}
var resultado = partida(.piedra, .papel)
print(resultado)
```

Problema:

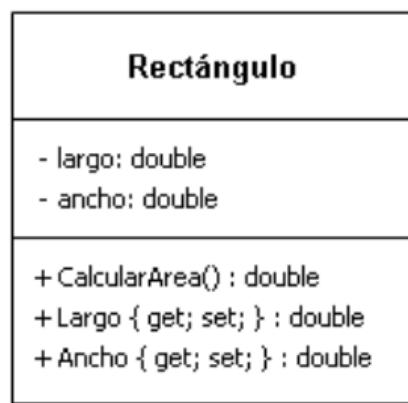
2.3.42. Tiene un arreglo de diccionarios. Cada diccionario tiene un par de valores, uno con el apellido y otro con el nombre de una persona. Debe entregar un arreglo que contenga cada nombre completo.

Solucion:

```
let diccionario: {[String: String]}[] =
[
    ["apellido": "Perea", "nombre": "Alicia"],
    ["apellido": "Flores", "nombre": "Norma"],
    ["apellido": "Noriega", "nombre": "Roberto"]
]
var nombres: [String] = []
for d in diccionario
{
    |   nombres.append("\u201c(d[\"nombre\"]!) \u201c(d[\"apellido\"]!)\u201d")
}
print(nombres)
```

Problema:

2.3.44. Hay una pared rectangular que tiene una ventana rectangular; se requiere un programa que dadas las dimensiones (largo y ancho) de la ventana y de la pared, imprima los minutos necesarios para pintar la pared, sabiendo que se puede pintar 1 m² en 10 minutos. Utilice la siguiente clase:



Solución:

```
class Rectangulo
{
    var ancho : Int
    var alto : Int

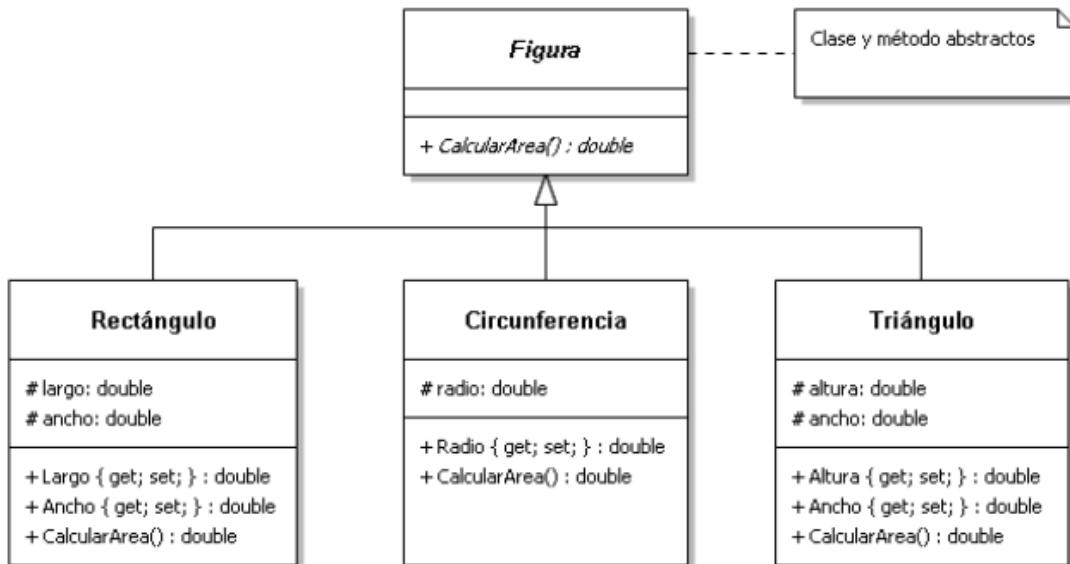
    init(al:Int, an:Int)
    {
        self.ancho = an
        self.alto = al
    }

    func Area( ) -> Int
    {
        return ancho * alto
    }
}

let pared = Rectangulo(al:10,an:5)
let ventana = Rectangulo(al:2,an:1)
let minutos = (pared.Area() - ventana.Area()) * 10
print(minutos)
```

Problema:

2.3.45. Diseñe un programa que calcule el área de las figuras geométricas señaladas en el diagrama UML que se muestra a continuación. Utilice polimorfismo para la reutilización de métodos abstractos.



Solución:

```
class Figura
{
func CalcularArea() -> Double
{
return 0.0
}
}
class Rectangulo : Figura
{
var ancho: Double
var alto: Double

init(al:Double, an:Double)
{
    self.ancho = an
    self.alto = al
}

override func CalcularArea() -> Double
{
    var area = ancho * alto
    return area
}
}
class Circunferencia : Figura
{
    var radio: Double
init(rad:Double)
{
    self.radio = rad
}
override func CalcularArea() -> Double
{
    var area = Double.pi * radio * radio
    return area
}
}
```

```
class Triangulo : Figura
{
    var altura: Double
    var ancho: Double
    init(al:Double,an:Double)
    {
        self.altura = al
        self.ancho = an
    }
    override func CalcularArea() -> Double
    {
        var area = (ancho * altura) / 2
        return area
    }
}
```

```
let rectangulo = Rectangulo(al:6,an:8)
let circunferencia = Circunferencia(rad: 14)
let triangulo = Triangulo(al:4,an:3)

let rectanguloResultado = rectangulo.CalcularArea()
let circunferenciaResultado = circunferencia.CalcularArea()
let trianguloResultado = triangulo.CalcularArea()
print(rectanguloResultado)
print(circunferenciaResultado)
print(trianguloResultado)
```

CONCLUSIONES

Juan Felipe Garza Sánchez

Por medio de las prácticas realizadas durante esta unidad que corresponde a conocer algunas de las funcionalidades que el lenguaje Swift nos provee y del cómo difiere o como tiene similitudes este lenguaje de programación respecto a los que ya habíamos manejado con anterioridad.

Viendo siempre desde un punto de vista de lógica por medio de ejercicios para la implementación desde la declaración de variables hasta la declaración de objetos complejos que requieran inicializaciones definidas en base a un problema, y apoyándonos también del conjunto de resultados esperados en base a unos datos de entrada para cerciorarnos de que nuestro programa está bien elaborado o cumple con lo esperado.

Sin duda alguna la unidad me permitirá tener una idea del cómo empezar a manejar distintos elementos del desarrollo IOS por medio de la programación y el cómo es la manera de trabajar en el lenguaje de programación Swift.

Jessica Janet Grajeda Castellanos

En mi opinión creo que este lenguaje tiene muchas ventajas ya que es bastante sencillo de aprender, en mi caso no tuve muchas complicaciones para realizar las prácticas además de que me parecieron muy interesantes también. Swift tiene la ventaja de tener su playground online que ayuda mucho a los que quieren aprender este lenguaje, pero no cuentan con una computadora con MacOS, este lenguaje tiene sus razones por las cuales es uno de los más populares, no es solo porque este casado con Apple y sus dispositivos.

Miguel Ángel Méndez Cruz

Swift es un lenguaje muy práctico a mi parecer, además de que también muy sencillo ya que su lenguaje está basado en varios lenguajes y eso lo hace interesante. Realizar las prácticas fue muy cómodo ya que no se presentaban tantos problemas de la sencillez que tiene el lenguaje. Me parece un lenguaje muy moderno visualmente por lo que yo pienso que atrae también a otros programadores a probar el lenguaje.

BIBLIOGRAFÍA | LINKOGRAFÍA

Apple Inc. Methods — The Swift Programming Language (Swift 5.4).
<https://docs.swift.org/swift-book/LanguageGuide/Methods.html>. Recuperado 20 Mar. 2021.

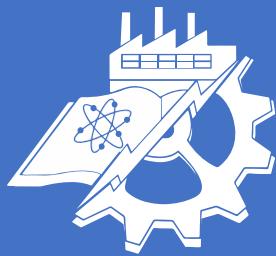
---. The Basics — The Swift Programming Language (Swift 5.4).
<https://docs.swift.org/swift-book/LanguageGuide/TheBasics.html>. Recuperado 22 Mar. 2021.

Fitzpatrick, Aidan. “Cómo Abrir Un Archivo IPA En Windows.” Reincubate: The App Data Company, Reincubate, 19 June 2014, <https://reincubate.com/es/support/how-to/open-ipa-file-on-windows/>. Recuperado 20 Mar. 2021.

Jacobs, Bart. “Namespaces in Swift.” Stop Writing Swift That Sucks,
<https://cocoacasts.com/namespaces-in-swift>. Recuperado 20 Mar. 2021..

“Swift Ya Es Código Abierto. Análisis | Apple Coding.” Apple Coding,
<http://www.facebook.com/applecoding>, 3 Dec. 2015,
<https://applecoding.com/analisis/swift-codigo-abierto>. Recuperado 20 Mar. 2021.

Theory, Geeky. “Swift - Geeky Theory.” Geeky Theory, <https://geekytheory.com/swift>. Recuperado 20 Mar. 2021.



INSTITUTO TECNOLÓGICO DE NUEVO LAREDO

CON LA CIENCIA POR LA HUMANIDAD

INGENIERÍA EN SISTEMAS COMPUTACIONALES



💻 Programación móvil 2

Tema 3

DOCENTE

Ing. Humberto Peña Valle M.T.I.

🎓 INTEGRANTES

Jessica Janet Grajeda Castellanos	17100229
Juan Felipe Garza Sánchez	17100218
Miguel Ángel Méndez Cruz	17100254

INDICE

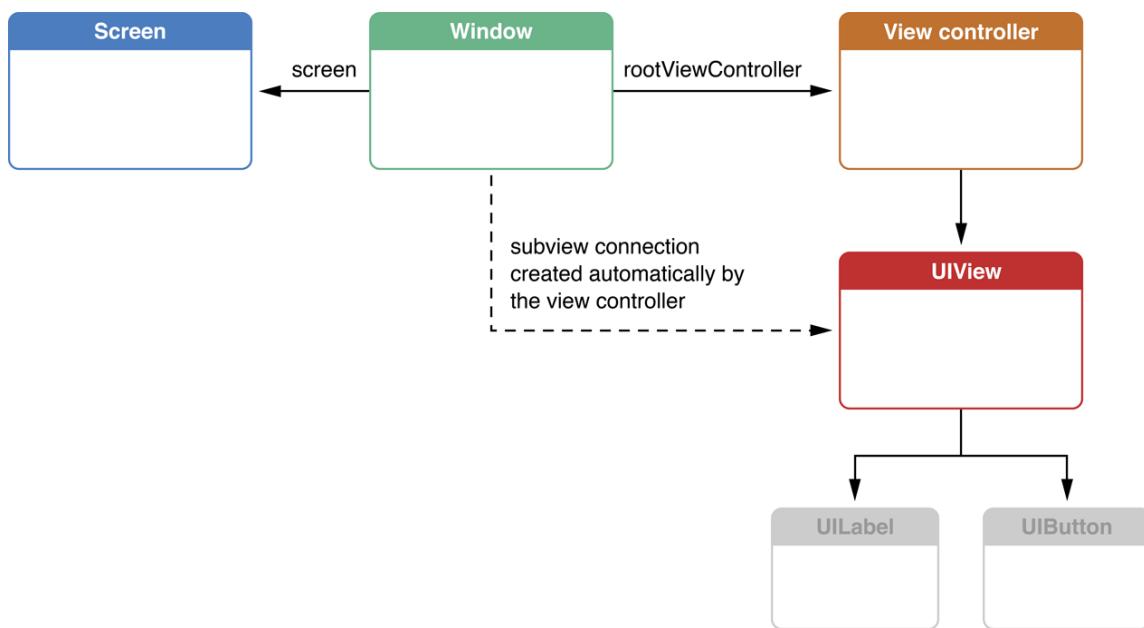
Explicación general del tema	284
3.1 Funcionalidad y propiedades comunes de los controles de Swift	285
3.2 Explicación de archivos creados a partir de una vista sencilla (Single View Application)	307
3.3 Diagrama de Jerarquía de clases de los controles del punto a partir de la clase NSObject	310
Conclusiones	312
Bibliografía	313

EXPLICACIÓN GENERAL DEL TEMA

En este tema se explica cada uno de los controladores de Swift utilizados para la programación móvil en dispositivos IOS, veremos desde los mas comunes hasta algunos avanzados, desde textos hasta barras de búsqueda, cada uno de los controladores con sus propiedades y herramientas

En iOS, los controladores de navegación son una de las principales herramientas para presentar múltiples pantallas de contenido. Y en este documento veremos todas las funcionalidades que pueden llegar a lograr los controladores en una aplicación

Veremos el diagrama de jerarquía de las clases de todos los controladores existentes en Swift, así como el orden de prioridad entre otras.



También en esta actividad se vera los archivos que se crean de manera predeterminada en una vista sencilla de Xcode, así como también sus categorías y tipos de archivos que se pueden agregar en el navegador del proyecto.

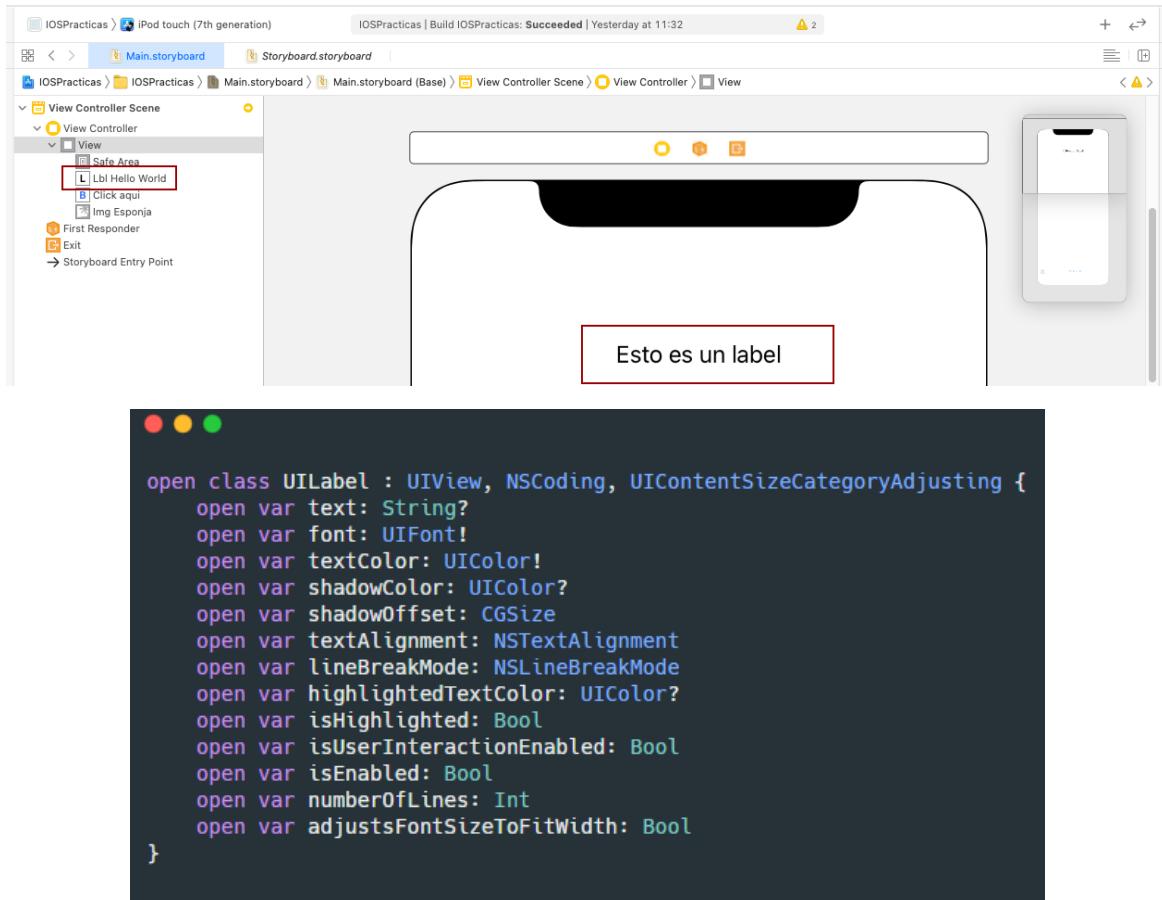
3.1 FUNCIONALIDAD Y PROPIEDADES COMUNES DE LOS CONTROLES DE



UILabel

Siendo su clase padre “UIView”, representa un texto de solo de lectura en nuestra pantalla, con fines puramente informativos.

Contiene una cantidad arbitraria de texto, sin embargo, puede llegar a truncar el texto dependiendo del tamaño del contenedor en el que se encuentra.

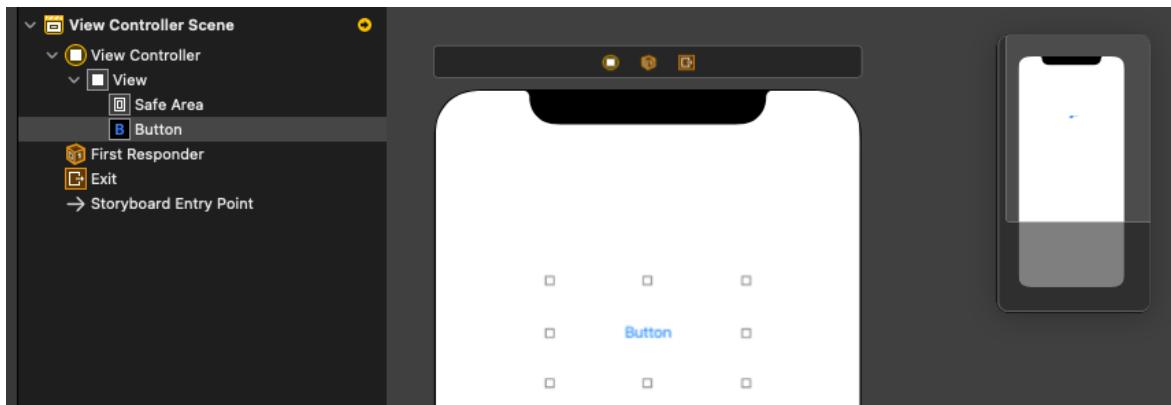


Las propiedades que componen este elemento son

- Texto o contenido
- Fuente de texto
- Color de texto
- Sombra del texto
- Colocar estilo de línea rota
- Alineamiento
- Rompimiento de línea
- Número de líneas
- Color de texto cuando es subrayado
- Saber si se encuentra habilitado
- Saber si es interactivo con el usuario
- Saber si se ajusta el tamaño de fuente respecto al ancho

UIButton

Este control ejecuta Código personalizado según las interacciones del usuario. Se puede agregar botones desde programación o utilizando el Interface Builder de Xcode.



Se puede usar las propiedades de este elemento para cambiar la apariencia del botón como asignarle un color o cambiar el formato del título. Las propiedades son las siguientes:

- Tipo de botón
- Título del botón
- Foto de primer plano del botón
- Foto de fondo del botón

UISegmentedControl

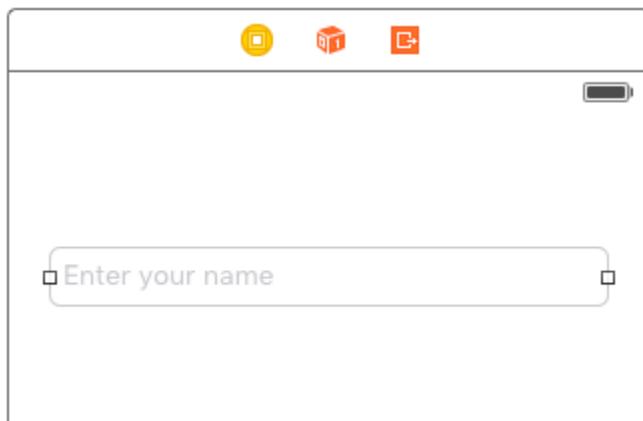
Es un control horizontal que contiene múltiples segmentos que funcionan como botones discretos.



Un control de este tipo puede mostrar un título o una imagen

UITextField

Es un objeto que muestra un área de texto la cual es editable

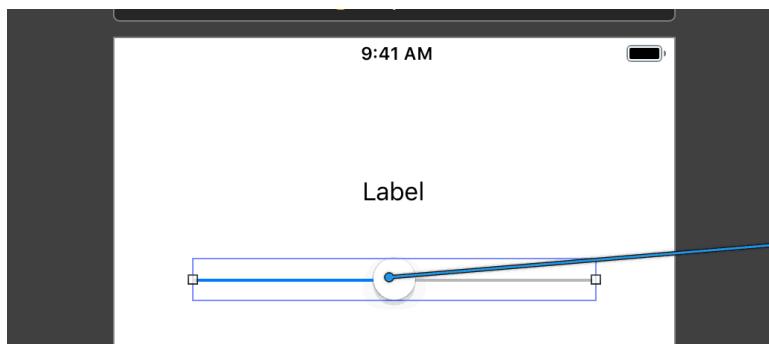


El teclado se puede configurar para muchos tipos diferentes de entrada, también puede agregar vistas para mostrar información adicional y dar controles adicionales como iconos de búsqueda. Las propiedades disponibles son las siguientes

- Color del texto del textfield
- La fuente utilizada en el textfield
- La alineación del texto del campo de texto dentro del área de escritura
- El texto placeholder que se muestra en el campo de texto.
- La imagen de fondo que se muestra cuando el campo de texto está habilitado
- La imagen de fondo que se muestra cuando el campo de texto está desactivado
- El estilo visual del borde del campo de texto

UISlider

Este control puede seleccionar un solo valor de un rango de valores

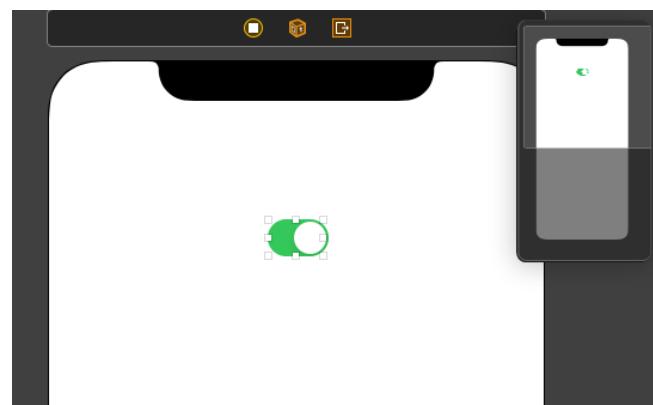


La apariencia del control es configurable ya que se puede cambiar el color del botón deslizante así como la línea de recorrido además de proporcionar imágenes para que aparezcan en los extremos del control. Es posible agregar este control mediante programación o mediante el Interface Builder. Las propiedades son los siguientes

- Especificar los valores adjuntos a los puntos finales del control deslizante, el mínimo representa el extremo anterior del control deslizante y el máximo representa el extremo final
- Valor inicial del slider

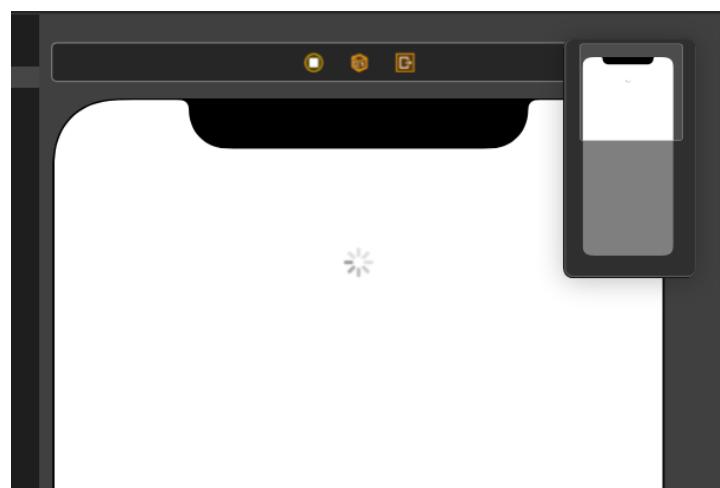
UISwitch

Este control que ofrece una opción binaria la cual puede ser encendido o apagado. La clase UISwitch declara una propiedad y un método para controlar su estado activado o desactivado. Al igual que con UISlider, cuando el usuario manipula el control, activa el evento valueChanged.



UIActivityIndicatorView

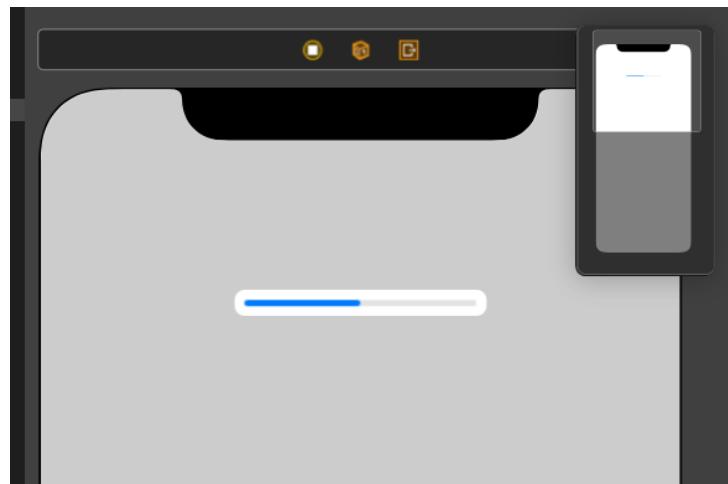
Esta es una vista que muestra que una tarea está en curso. Puede controlar cuándo se anima un indicador de actividad llamando a los métodos startAnimating() y stopAnimating().



Para ocultar automáticamente el indicador de actividad cuando se detiene la animación, debe establecerse la propiedad hidesWhenStopped en true. También puede establecer el color del indicador de actividad mediante la propiedad de color.

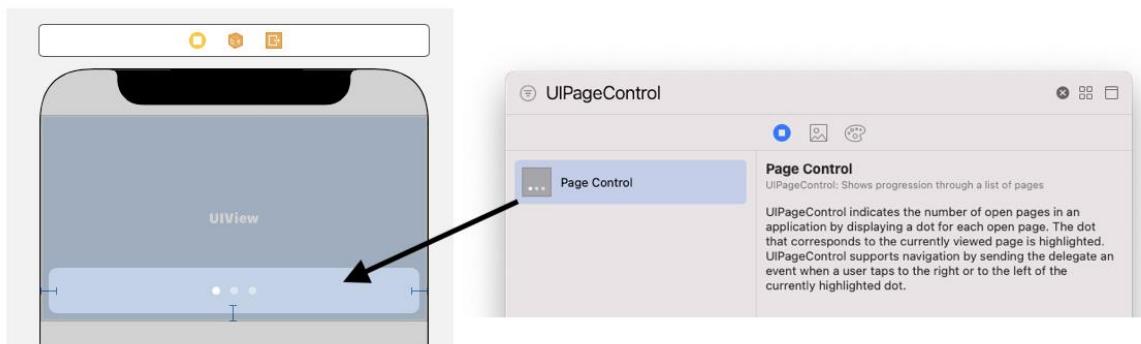
UIProgressView

Esta vista muestra el progreso de una tarea a lo largo del tiempo. La clase `UIProgressView` proporciona propiedades para administrar el estilo de la barra de progreso y para obtener y establecer valores que están anclados al progreso de una tarea.



UIPageControl

Este control muestra una serie horizontal de puntos, cada uno de los cuales corresponde a una página en el documento de la aplicación o un modelo de datos.

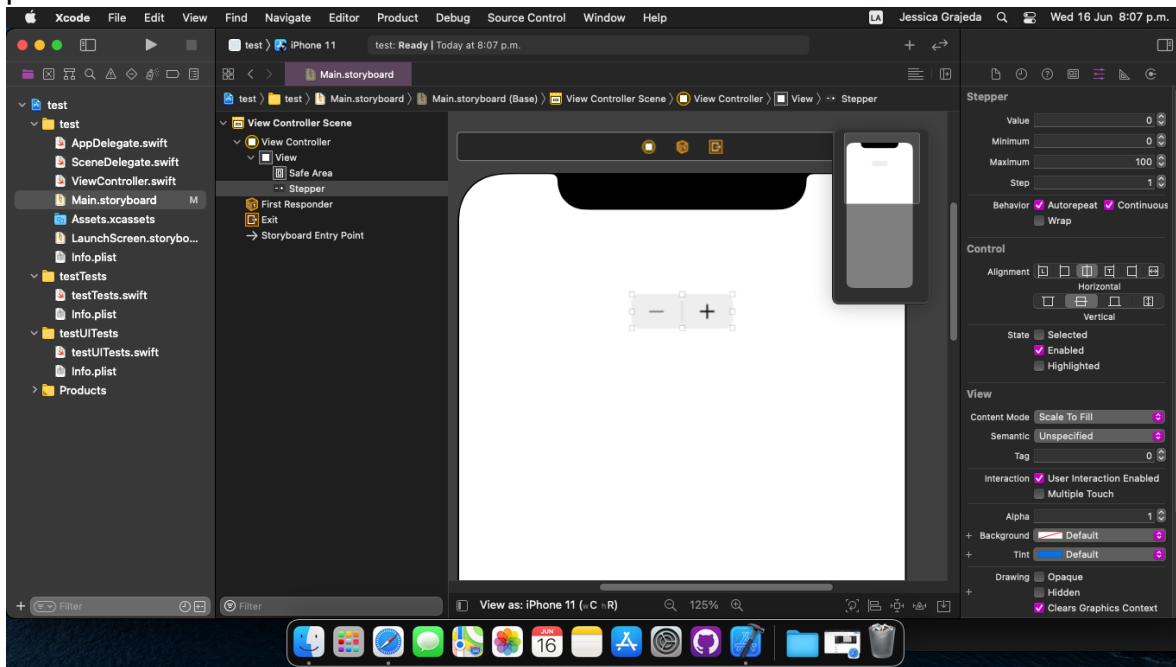


Cuando un usuario toca un control de página para pasar a la página siguiente o anterior, el control envía el evento `valueChanged` para que lo maneje el delegado. A continuación, el delegado puede evaluar la propiedad `currentPage` para determinar la página que se mostrará. El control de página avanza solo una página en cualquier dirección. La página visualizada actualmente se indica con un punto blanco.

UIStepper

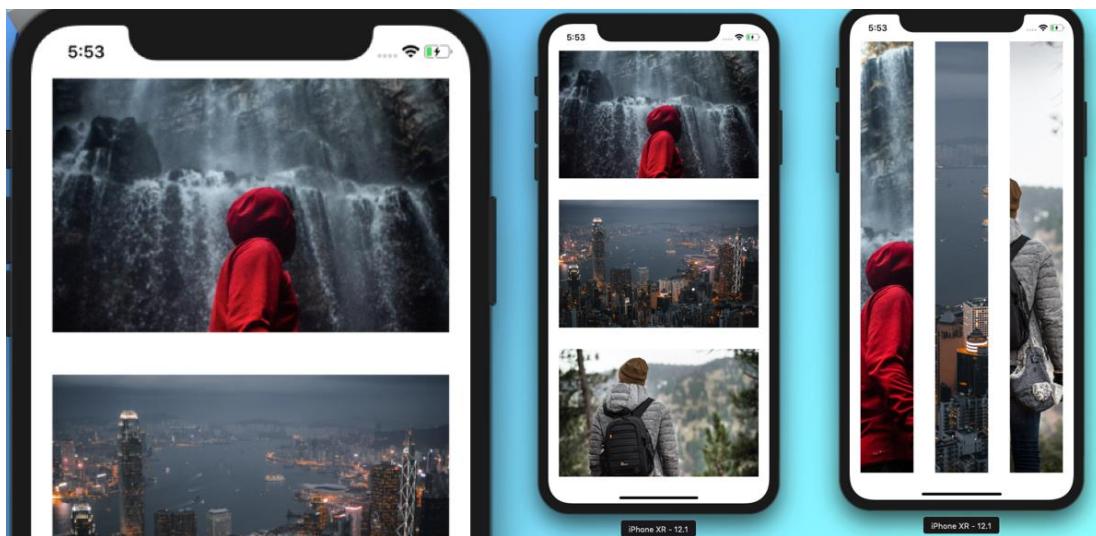
Este control sirve para incrementar o disminuir un valor. De forma predeterminada, presionar y mantener presionado el botón aumenta o disminuye el valor repetidamente. La tasa de cambio depende de cuánto tiempo el usuario continúe presionando el

control. Para desactivar este comportamiento, existe la propiedad autorepeat que debe ponerse en false de ser el caso.



UIStackView

Esta es una interfaz optimizada para diseñar una colección de vistas en una columna o una fila. Las vistas de pila permiten aprovechar el poder del diseño automático, creando interfaces de usuario que pueden adaptarse dinámicamente a la orientación del dispositivo, el tamaño de la pantalla y cualquier cambio en el espacio disponible.



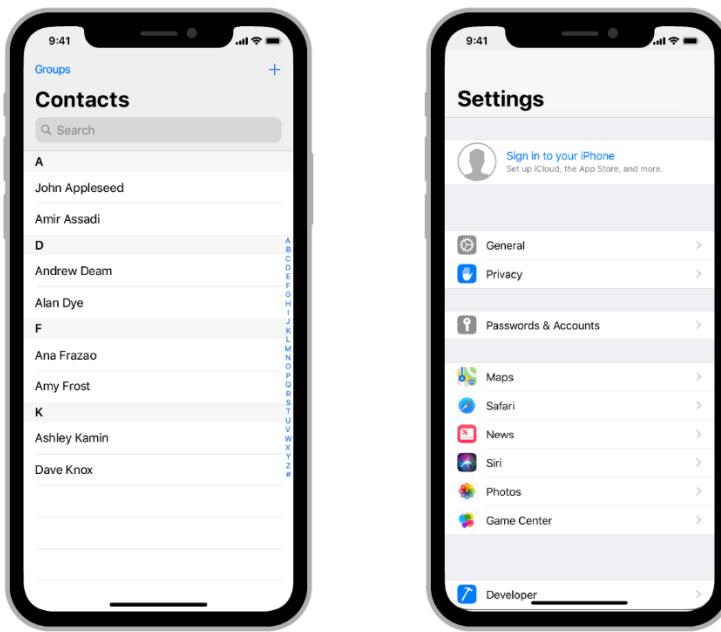
Las propiedades son las siguientes:

- La propiedad Axis determina si las views contenidas en la UIStackView deben colocarse en posición Vertical u Horizontal

- La propiedad Distribution define el tamaño y posición de las views contenidas en la UIStackView. Por defecto toma el valor Fill, lo que significa que cada una de las views contenidas ocupan todo el espacio disponible en la UIStackView.
- La propiedad Alignment especifica como se alinearán las views contenidas en la UIStackView.
- La propiedad Spacing permite especificar el espacio que habrá entre las views.

UITableView

Este es una vista que presenta datos usando filas en una sola columna. UITableView administra la apariencia básica de la tabla, pero en la aplicación se proporcionan celdas (objetos UITableViewCell) que muestran el contenido real.



UITableViewCell

Un objeto UITableViewCell es un tipo de vista especializado que administra el contenido de una sola fila de la tabla. Utiliza celdas principalmente para organizar y presentar el contenido personalizado de su aplicación, pero UITableViewCell proporciona algunas propiedades específicas para controlar comportamientos relacionados con la tabla:

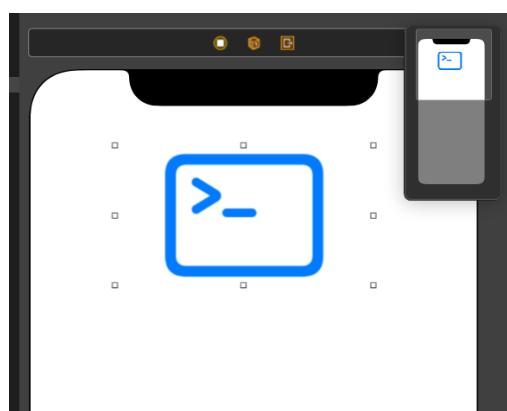
- Aplicar una selección o un color de resaltado a la celda.
- Agregar vistas de accesorios estándar, como un control de detalle o divulgación.
- Poner la celda en un estado editable.

- Aplicar sangría al contenido de la celda para crear una jerarquía visual en su tabla.



UIImageView

Este objeto que muestra una sola imagen o una secuencia de imágenes animadas en su interfaz. Permiten dibujar de manera eficiente cualquier imagen que se pueda especificar utilizando un objeto UIImage. clase UIImageView para mostrar el contenido de muchos archivos de imagen estándar, como archivos JPEG y PNG.



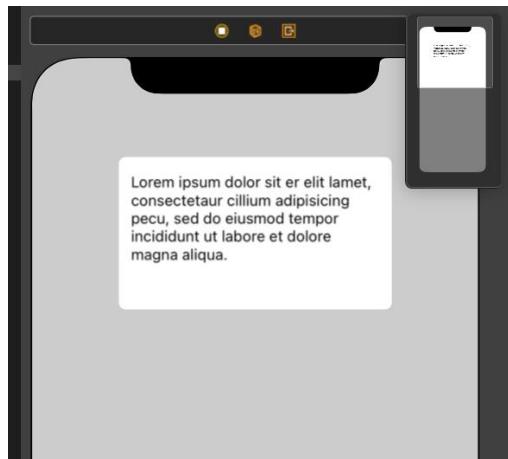
Las propiedades son las siguientes:

- La imagen a especificar, puede ser cualquier imagen dentro del proyecto de Xcode
- La imagen que se mostrará cuando se resalte la vista de imagen

- El estado inicial de la imagen (Puede ser resaltado)

UITextView

UITextView admite la visualización de texto utilizando información personalizada y también admite la edición de texto

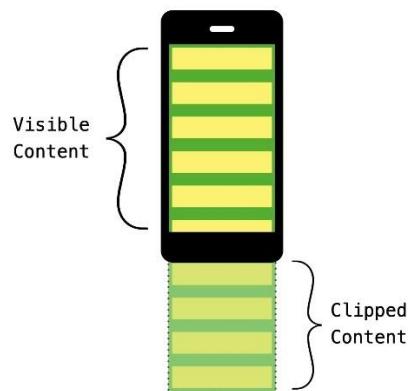


Las propiedades son:

- El texto predeterminado
- El color de los bordes del control
- El tipo de fuente a utilizar
- Si se puede ajustar el tamaño de la fuente dinámicamente
- La alineación del texto

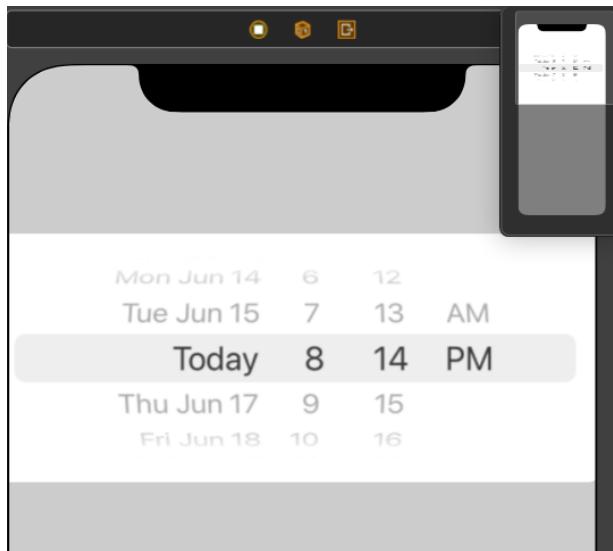
UIScrollView

Esta vista permite el desplazamiento y el zoom de sus vistas contenidas. UIScrollView es la superclase de varias clases de UIKit, incluidas UITableView y UITextView. Una vista de este tipo es una vista con un origen que se puede ajustar sobre la vista de contenido. Este recorta el contenido a su marco, que generalmente (pero no necesariamente) coincide con el de la ventana principal de la aplicación. También rastrea los movimientos de los dedos y ajusta el origen en consecuencia.



UIDatePicker

Este control sirve para la entrada de valores de fecha y hora.



El selector de fechas informa las interacciones a su objeto de destino asociado. Las propiedades son las siguientes:

- Estilo del picker
- Modo del picker (Puede ser hora o fecha o ambos)
- Localización asociada al picker

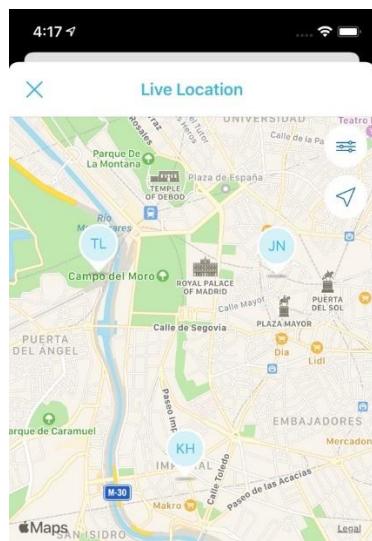
UIPickerView

Una vista de este estilo muestra una o más ruedas que el usuario manipula para seleccionar elementos. Cada rueda, conocida como componente, tiene una serie de filas indexadas que representan los elementos seleccionables. Cada fila muestra una cadena o vista para que el usuario pueda identificar y seleccionar el elemento en esa fila.



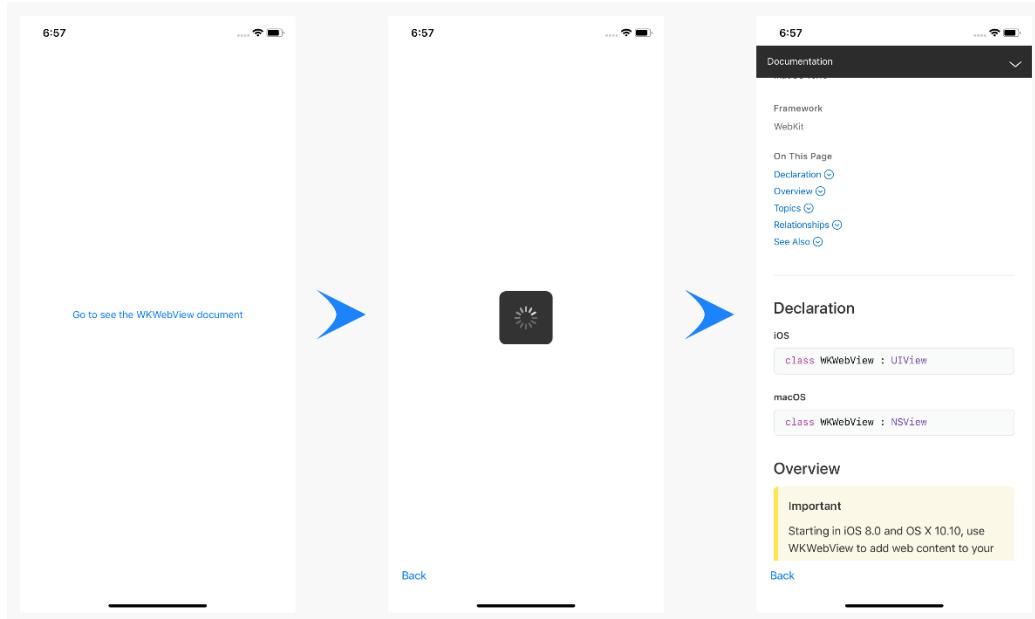
MKMapView

Este control es una interfaz de mapa integrable, similar a la proporcionada por la aplicación Maps. Se utiliza para mostrar información del mapa y manipular el contenido del mapa. Se puede centrar el mapa en una coordenada determinada, especificar el tamaño del área que desea mostrar y anotar el mapa con información personalizada.



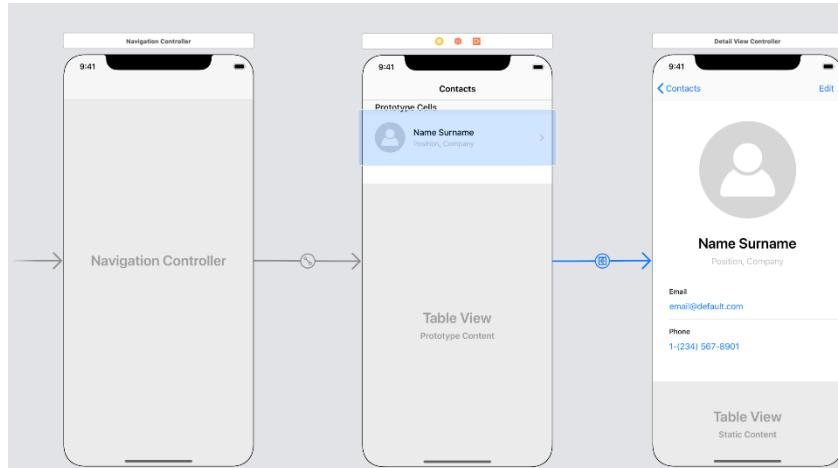
WKWebView

Un objeto WKWebView es una vista nativa utilizada para incorporar contenido web sin problemas en la interfaz una aplicación. Además, admite una experiencia de navegación web completa y presenta contenido HTML, CSS y JavaScript.



UIView

Descripción general Las vistas son los bloques de construcción fundamentales de la interfaz de usuario de su aplicación, y la clase UIView define los comportamientos que son comunes a todas las vistas. Un objeto de vista representa el contenido dentro de su rectángulo de límites y maneja cualquier interacción con ese contenido. La clase UIView es una clase concreta que puede crear instancias y usar para mostrar un color de fondo fijo. También puede subclásificarlo para dibujar contenido más sofisticado. Para mostrar etiquetas, imágenes, botones y otros elementos de la interfaz que se encuentran comúnmente en las aplicaciones, use las subclases de vista que proporciona el marco UIKit en lugar de intentar definir las suyas propias.



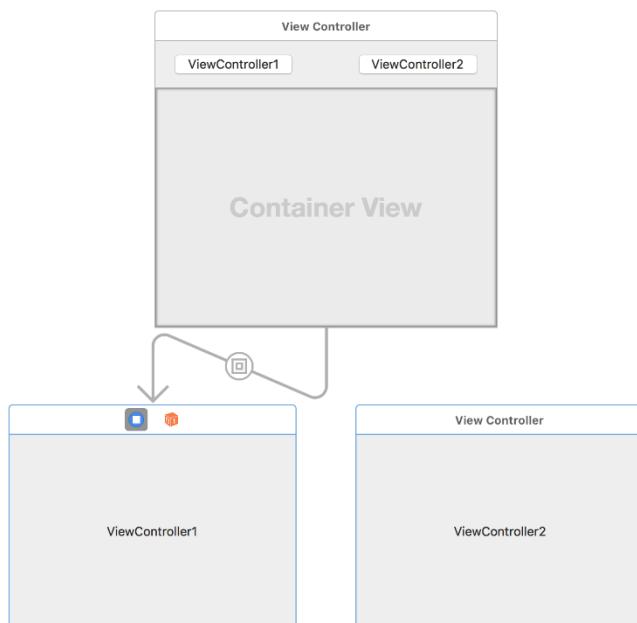
Debido a que los objetos de visualización son la forma principal en que su aplicación interactúa con el usuario, tienen una serie de responsabilidades. Éstos son solo algunos:

- Dibujo y animación
- Las vistas dibujan contenido en su área rectangular utilizando UIKit o Core Graphics.
- Puede animar algunas propiedades de la vista a nuevos valores.
- Gestión de diseño y subvista
- Las vistas pueden contener cero o más subvistas.
- Las vistas pueden ajustar el tamaño y la posición de sus subvistas.

UIContainerView

Los controladores de vista de contenedor promueven una mejor encapsulación al separar su contenido de cómo lo muestra en pantalla. A diferencia de un controlador de vista de contenido que muestra los datos de su aplicación, un controlador de vista de contenedor muestra otros controladores de vista, organizándolos en pantalla y manejando la navegación entre ellos. Un controlador de vista de contenedor sigue siendo un controlador de vista, por lo que lo muestra en una ventana o lo presenta como cualquier otro controlador de vista.

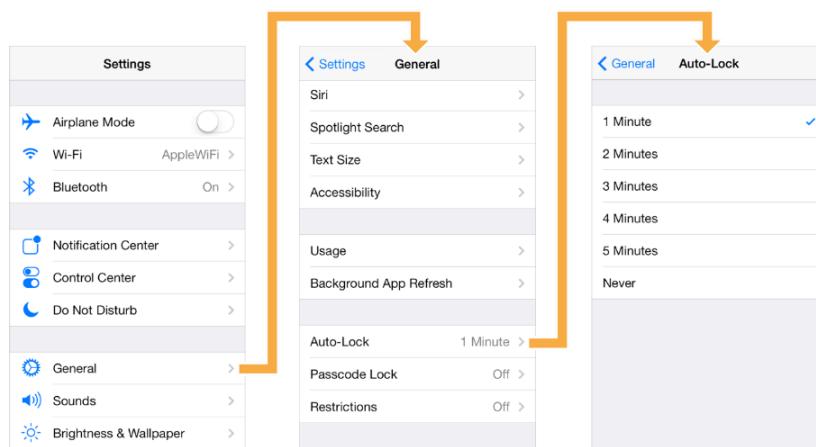
Un controlador de vista de contenedor también administra una interfaz compuesta, incorporando las vistas de uno o más controladores de vista secundarios en su propia jerarquía de vista. Cada niño continúa administrando su propia jerarquía de vistas, pero el contenedor administra la posición y el tamaño de la vista raíz de ese niño.



UINavigationBar

Un objeto UINavigationBar es una barra, que normalmente se muestra en la parte superior de la ventana, que contiene botones para navegar dentro de una jerarquía de pantallas. Los componentes principales son un botón izquierdo (atrás), un título central y un botón derecho opcional. Puede utilizar una barra de navegación como un objeto independiente o junto con un objeto de controlador de navegación.

Una barra de navegación se usa más comúnmente dentro de un controlador de navegación. El objeto UINavigationController crea, muestra y administra su barra de navegación asociada, y usa atributos de los controladores de vista que agrega para controlar el contenido que se muestra en la barra de navegación.



En el caso de UINavigationBar podemos personalizar los siguientes atributos:

- Imagen y color de fondo
- Sombra de la barra de navegación
- Tipografía del título
- Botones

Propiedades de UINavigationController :

- topViewController apunta al VC en el top de la pila.
- visibleViewController apunta al VC mostrado.
- viewControllers es el array con la pila de VCs.
- navigationBar es la barra de navegación mostrada.
- toolBar es la barra de herramientas mostrada.

UINavigationItem

Al crear una interfaz de navegación, cada controlador de vista que inserte en la pila de navegación debe tener un objeto UINavigationItem que contenga los botones y las vistas que desea mostrar en la barra de navegación. El objeto de administración UINavigationController usa los elementos de navegación de los dos controladores de vista superiores para llenar la barra de navegación con contenido.

Un elemento de navegación siempre refleja información sobre su controlador de vista asociado. El elemento de navegación debe proporcionar un título para mostrar cuando el controlador de vista está en la parte superior de la pila de navegación. El elemento también puede contener botones adicionales para mostrar en el lado derecho (o al final) de la barra de navegación. Puede especificar botones y vistas para que se muestren en el lado izquierdo (o inicial) de la barra de herramientas mediante la propiedad leftBarButtonItems, pero el controlador de navegación muestra esos botones solo cuando hay espacio disponible.



Algunas propiedades:

Cada objeto View Controller usado en este tipo de navegación tiene asociado un objeto UINavigationItem donde configura los elementos a mostrar en la barra de navegación cuando es visible.

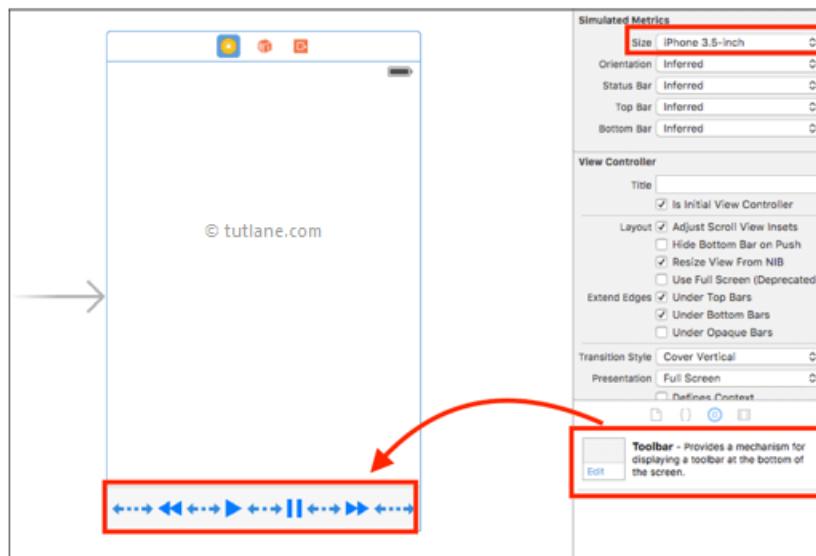
- title es el título mostrado en el centro de la barra de navegación.
- backBarButtonItem es el botón para retroceder a la pantalla anterior.
- prompt es una línea de texto situada en la parte superior de la barra de navegación
- titleView, leftBarButtonItems, leftBarButtonItem, rightBarButtonItems, rightBarButtonItem son views y botones personalizados

UIToolBar

Para crear elementos de la barra de herramientas, use la clase UIBarButtonItem. Para agregar elementos de la barra de herramientas a una barra de herramientas, use el método setItems (_: animated :).

Las imágenes de la barra de herramientas que representan los estados normales y resaltados de un elemento se derivan de la imagen que estableciste usando la propiedad de imagen heredada de la clase UIBarButtonItem. La imagen está coloreada con el tintColor de la barra de herramientas.

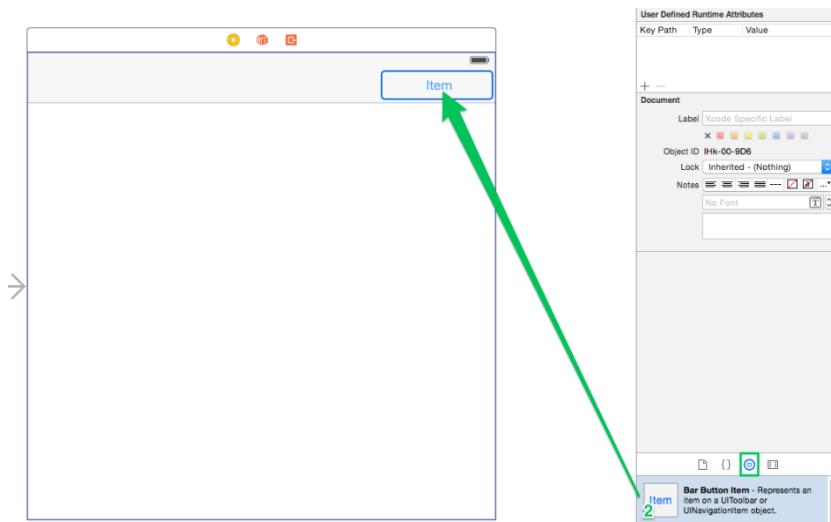
Si necesita controles de estilo de botón de opción, use la clase UITabBar en lugar de UIToolbar



UIBarButtonItem

Normalmente, utiliza Interface Builder para crear y configurar elementos de botones de barra. Sin embargo, puede personalizar la apariencia de los botones enviando los mensajes del configurador a UIBarButtonItemAppearance para personalizar todos los botones, o a una instancia específica de UIBarButtonItem. Puede utilizar botones personalizados en lugares estándar en un objeto UINavigationItem (backBarButtonItem, leftBarButtonItem, rightBarButtonItem) o una instancia de UIToolbar.

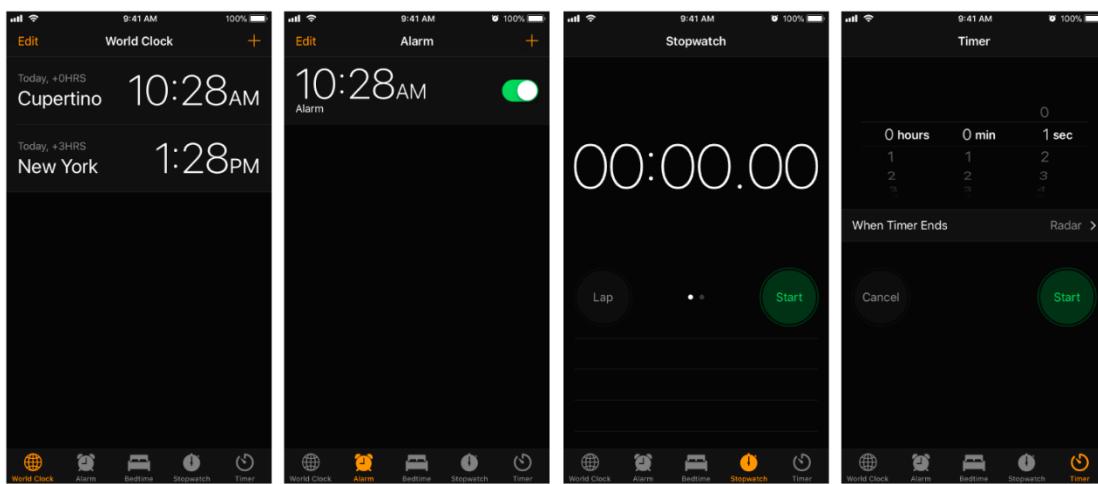
En general, debe especificar un valor para el estado normal para que otros estados sin un conjunto de valores personalizado puedan usarlo. De manera similar, cuando una propiedad depende de las métricas de la barra (por ejemplo, en el iPhone, en orientación horizontal, las barras tienen una altura diferente a la estándar), debe especificar un valor de UIBarMetrics.default.



UITabBar

Por lo general, usa barras de pestañas junto con un objeto UITabBarController, pero también puede usarlas como controles independientes en su aplicación. Las barras de pestañas siempre aparecen en el borde inferior de la pantalla y muestran el contenido de uno o más objetos UITabBarItem. La apariencia de una barra de pestañas se puede personalizar con una imagen de fondo o un color de tinte para adaptarse a las necesidades de su interfaz. Al tocar un elemento, se selecciona y resalta ese elemento, y usa la selección del elemento para habilitar el modo correspondiente para su aplicación.

Puede configurar barras de pestañas mediante programación o en Interface Builder. Un objeto UITabBarController proporciona su propio objeto de barra de pestañas y debe configurar el objeto que se le proporciona.



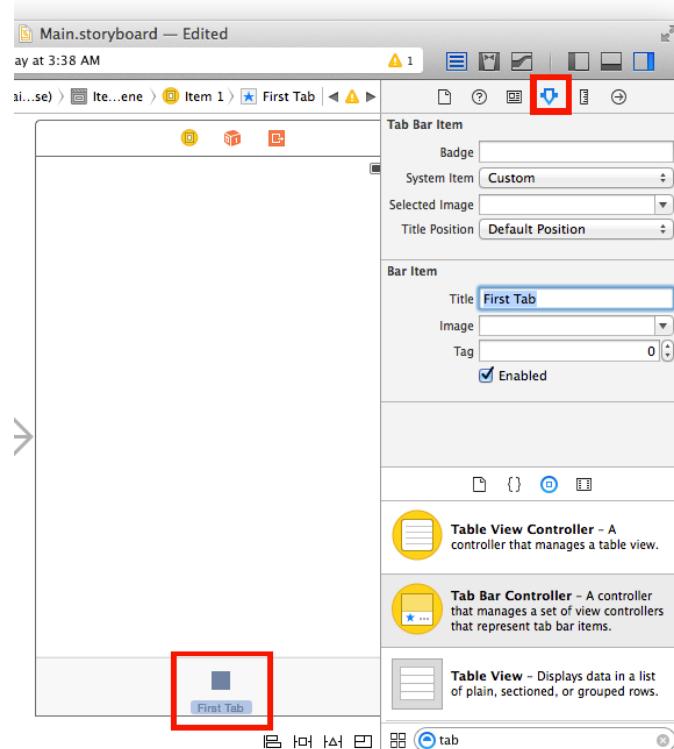
Algunos métodos y propiedades:

- Array con todos los view controllers que gestiona.
`var viewControllers: [UIViewController]?`
 - Los VC de este array, y todos los VC descendientes de estos, tienen una propiedad, llamada `tabBarController`, que apunta al objeto Tab Bar Controller que los contiene.
 - Esta propiedad es nil si el VC no está gestionado por un Tab Bar Controller
- Programáticamente los VCs pueden cambiarse por otros con el método:
`func setViewControllers(_ viewControllers: [UIViewController]?, animated: Bool)`
- El VC seleccionado se indica con:
`var selectedViewController: UIViewController? var selectedIndex: Int`
- Un delegado `UITabBarControllerDelegate` con el que se puede prohibir la selección de un VC, avisar cuando se selecciona un VC, etc...

UITabBarItem

Un elemento de la barra de pestañas es un segmento de una barra de pestañas que representa una sección específica de su aplicación. Una barra de pestañas muestra uno o más elementos que permiten al usuario cambiar entre las diferentes secciones. El usuario puede seleccionar un elemento a la vez.

El sistema proporciona varios elementos de la barra de pestañas para casos de uso comunes. Si necesita un elemento personalizado, cree uno con un título y una imagen. Puede personalizar aún más el elemento proporcionando una imagen alternativa que aparece cuando el usuario lo selecciona. De forma predeterminada, el artículo no muestra las imágenes que proporcionas. En cambio, genera nuevas imágenes a partir de los valores alfa de sus imágenes y las tiñe. Para evitar esto, proporcione imágenes que utilicen el modo de representación `UIImage.RenderingMode.alwaysOriginal`.

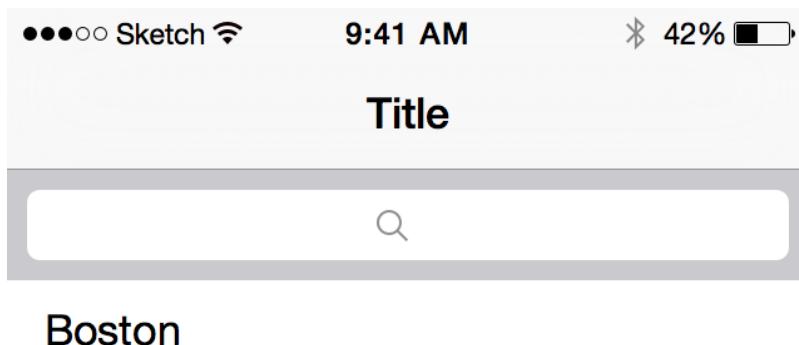


Algunas propiedades

- Title: el texto que sale en la pestaña
- Image: imagen asignada
- Tag: identificativo del ítem
- Enabled: posibilidad de habilitar o deshabilitar el elemento

UISearchBar

UISearchBar proporciona un campo de texto para ingresar texto, un botón de búsqueda, un botón de marcador y un botón de cancelación. Una barra de búsqueda en realidad no realiza ninguna búsqueda. Utiliza un delegado, un objeto conforme al protocolo UISearchBarDelegate, para implementar las acciones cuando el usuario ingresa texto o hace clic en botones. Para obtener detalles sobre la interacción con el campo de texto, el acceso a su contenido y el uso de tokens, consulte UISearchBarTextField y UISearchToken.



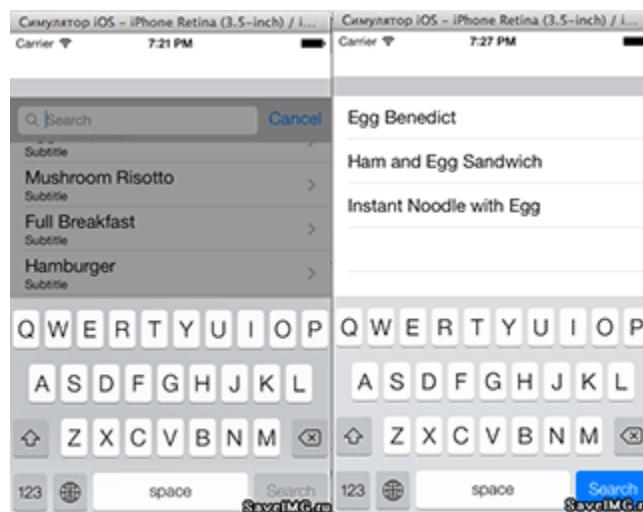
Propiedades:

- text: texto a buscar.
- placeholder: texto en gris que sirve de pista para el usuario.
- prompt: prompt mostrado encima de la barra de búsqueda.
- barStyle, searchBarStyle, barTintColor: apariencia.
- showsSearchResultsButton: botón ver búsquedas o resultados recientes.
- showsBookmarkButton: ícono para mostrar los bookmarks salvados.
- showsCancelButton: botón para cancelar.
- showsScopeBar, scopeButtonTitles, selectedScopeButtonIndex: mostrar la barra de scopes, títulos de los botones de esta barra, e índice del botón seleccionado.
- autocapitalizationType, autocorrectionType, keyboardType, spellCheckingType: configuración del campo de entrada de texto.

UISearchDisplayController

Un controlador de visualización de búsqueda gestiona la visualización de una barra de búsqueda, junto con una vista de tabla que muestra los resultados de la búsqueda.

Inicializa un controlador de visualización de búsqueda con una barra de búsqueda y un controlador de vista responsable de administrar los datos que se van a buscar. Cuando el usuario inicia una búsqueda, el controlador de visualización de búsqueda superpone la interfaz de búsqueda sobre la vista del controlador de vista original y muestra los resultados de la búsqueda en su vista de tabla.



Propiedades:

- searchBar - la SearchBar.
- searchContentsController - el VC propietario de los datos manejados.
- searchResultsTableView - la TableView donde se muestra el resultado de las búsquedas.
- searchResultsDataSource - el data source de la TableView de resultados.
- searchResultsDelegate - el delegado de la TableView de resultados.
- delegate - el delegado del propio SearchDisplayController.
- searchResultsTitle - el título para tabla con los resultados.
- displaysSearchBarInNavigationBar - booleano para indicar que la SearchBar se mostrará en la barra de navegación.

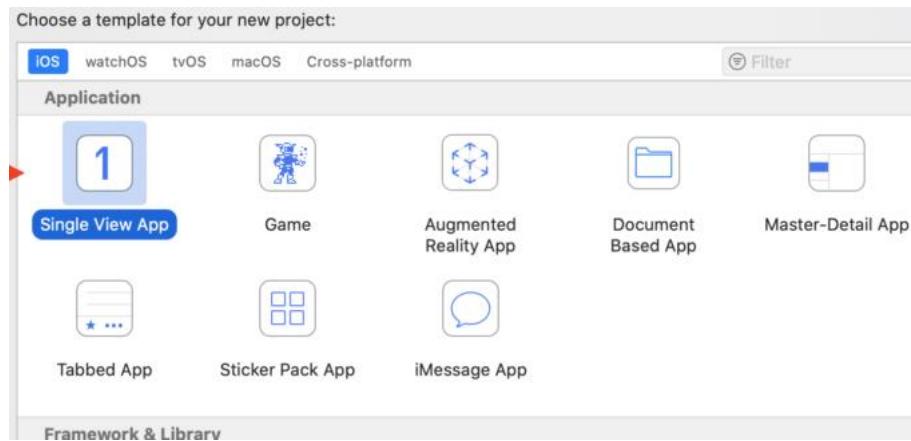
Las SearchBars presentadas en NavigationBars no pueden tener ScopeBar.

- navigationItem - NavigationItem usado por el SearchDisplayController.
- active - el estado de la interface de búsqueda.

3.2 Explicación de archivos creados a partir de una vista sencilla



Este tipo de template contiene un ViewController adicional el cual es personalizable, este tipo de template tiene dos opciones, el ser creado utilizando storyboards o no



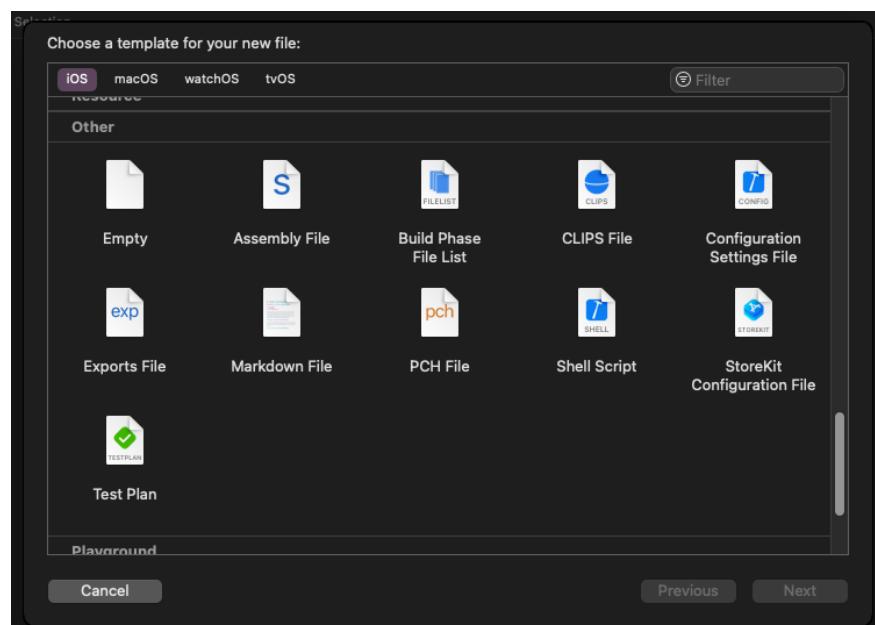
- Utilizando storyboards

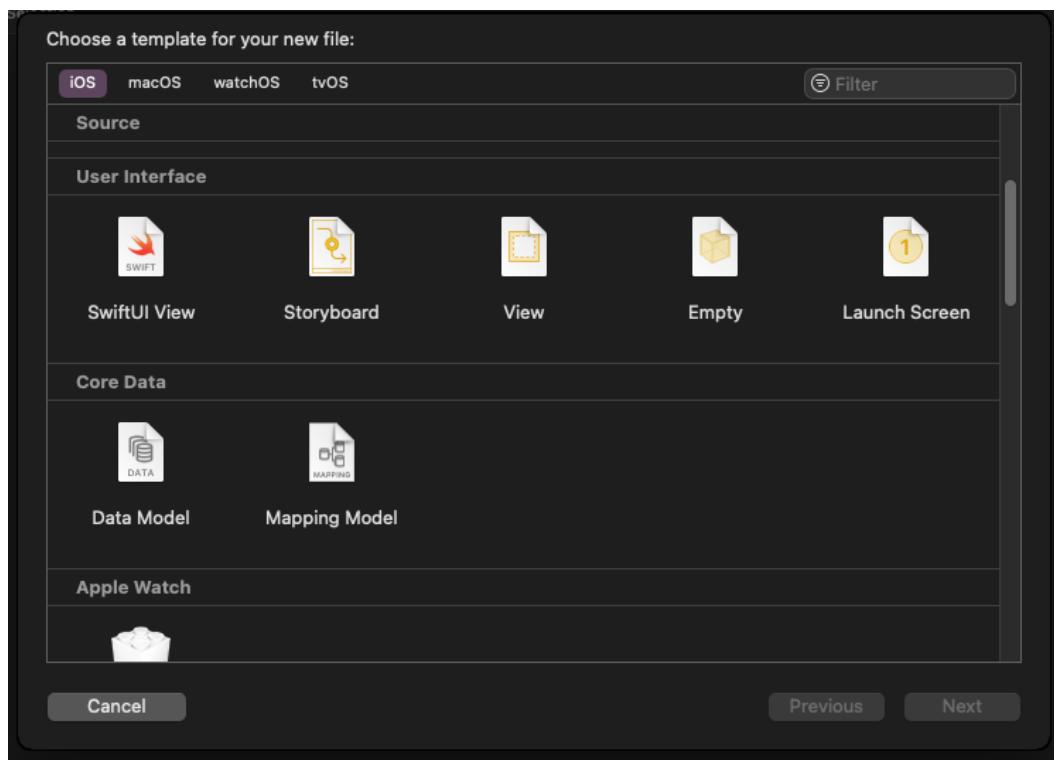
Esta versión del template contiene 3 nuevos archivos: el storyboard (MainStoryboard.storyboard), un header y un archivo de implementación para el ViewController. En el storyboard contiene una escena la cual representa el viewcontroller

- No utilizando storyboards

En caso de decidir esta opción, xcode creará archivos NIB/XIB, por ejemplo un .xib que contiene el viewcontroller principal del proyecto.

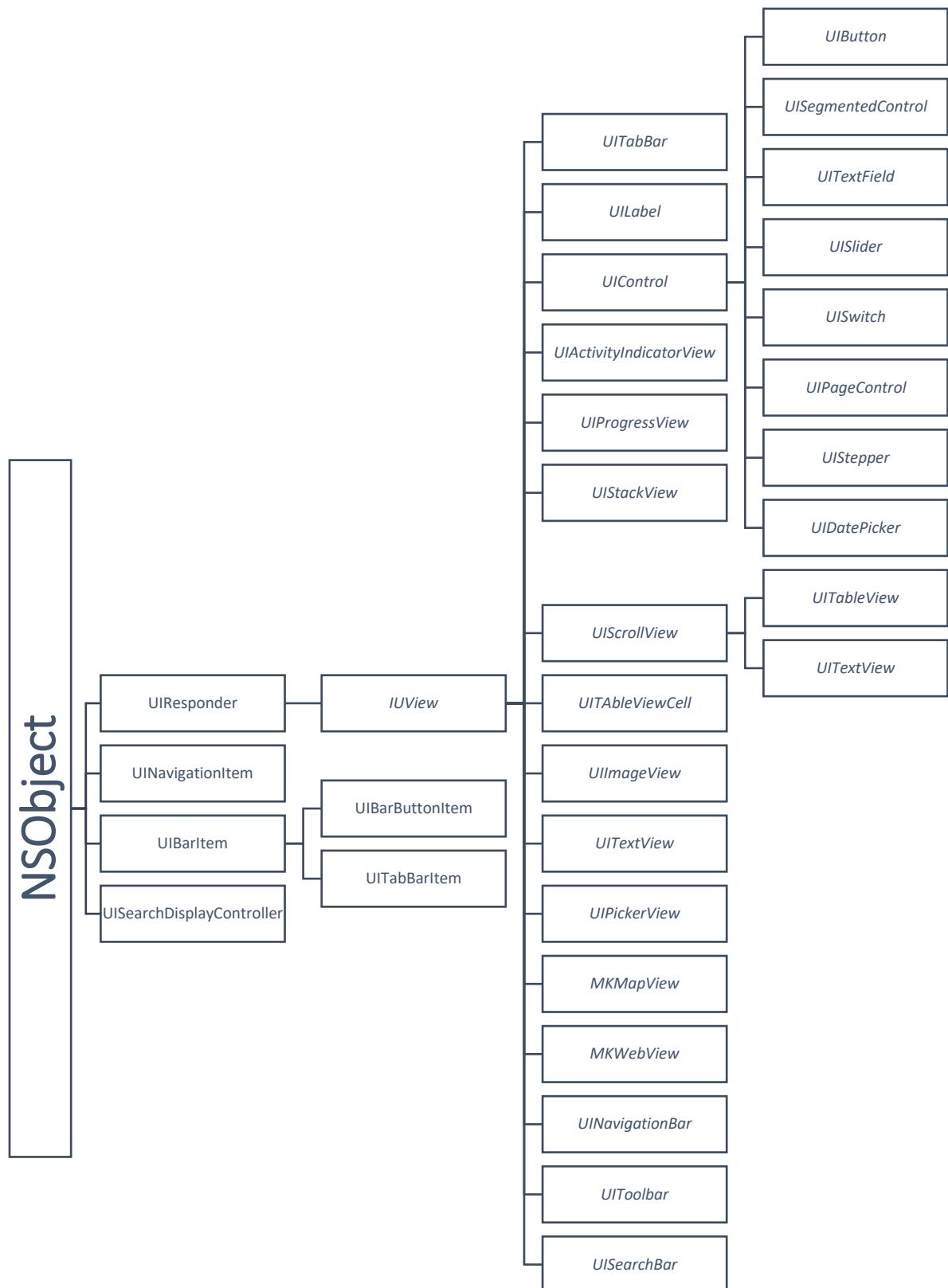
Los archivos que se pueden agregar al proyecto son los siguientes





3.3 Diagrama de jerarquía de clases de los controles del 3.1





CONCLUSIONES

Juan Felipe Garza Sánchez

En general siento que los conocimientos adquiridos son la base fundamental para empezar un desarrollo de aplicaciones de manera profesional, pues se ven temas sumamente importantes referentes a el manejo de los distintos controles que involucra una interfaz grafica en una aplicación del sistema operativos IOS, dando a conocer las distintas propiedades visuales y de programación que involucra cada control mencionado en esta investigación.

También y no menos importante la manera en que se maneja XCode y el cómo es que este IDE interactúa con nosotros para facilitarnos el trabajo durante el desarrollo de la aplicación en general, utilizando un conjunto de buenas prácticas por medio de protocolos ya definidos o incluso advertencias de algo totalmente redundante o innecesario, lo cual como desarrolladores nos ayuda incluso en tiempo de desarrollo para aprender algo nuevo cada día.

Si bien lo sé, que no basta con simplemente con conformarse con lo aprendido, sí que me deja una idea clara de el cómo es trabajar con los controles predeterminados que nos ofrece este Framework de desarrollo de aplicaciones en IOS y abriendo la posibilidad de incluso extender estos controles para facilitarnos el trabajo durante el desarrollo.

La manera de representar gráficamente desde el punto de vista de jerarquía de controles del Framework que involucra el kit de gráfico de controles, viendo desde su origen el cual es NSObject hasta controles muy específicos, nos dan a conocer las similitudes y diferencias que se tienen entre sí y el cuál debemos elegir en ciertas circunstancias.

Jessica Janet Grajeda Castellanos

Personalmente pienso que el contenido de este tema es de los más importantes e interesantes que hemos visto a hasta la fecha, esto debido a que aprendimos cuales son los controles que podemos utilizar en aplicaciones, así como la utilización de los mismos. Me llevo de aprendizaje estos controles, así como la técnica de utilización de estos en Xcode para usarlos en el futuro con el proyecto final de este curso, así como en situaciones donde me vea en la oportunidad de crear aplicaciones en este sistema operativo.

Miguel Ángel Méndez Cruz

Ya viendo todos los controladores existentes es muy interesante la cantidad de aplicaciones interactivas y funcionales que se puede llegar a crear, además de las herramientas y propiedades que tiene cada una de ellas como para hacer diseños amigables con el usuario, cada uno de los controladores me parece muy elegante hablando del diseño a comparación de otros sistemas operativos.

BIBLIOGRAFÍA | LINKOGRAFÍA

<https://www.dit.upm.es/~santiago/docencia/ios/2014-16/054-TabBarController-20151030.pdf>

<https://www.dit.upm.es/~santiago/docencia/ios/2019-20/035-ViewController-20191106.pdf>

<https://www.dit.upm.es/~santiago/docencia/ios/2016-17/055-NavigationController-20170904.pdf>

<https://www.dit.upm.es/~santiago/docencia/ios/2000-14/057-SearchDisplayController/057-SearchDisplayController-20131019.pdf>

<https://www.iosapptemplates.com/blog/ios-programming/uisearchcontroller-swift>

<https://developer.apple.com/library/archive/featuredarticles/ViewControllerPGforiPhoneOS/ImplementingaContainerViewController.html>

<https://developer.apple.com/documentation/uikit>

<https://developer.apple.com/documentation/uikit/uiview>

<https://developer.apple.com/documentation/uikit/uinavigationbar>

<https://developer.apple.com/documentation/uikit/uinavigationitem>

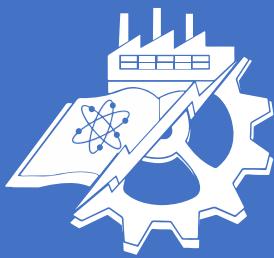
<https://developer.apple.com/documentation/uikit/uitoolbar>

<https://developer.apple.com/documentation/uikit/uitabbar>

<https://developer.apple.com/documentation/uikit/uitabbaritem>

<https://developer.apple.com/documentation/uikit/uisearchbar>

<https://developer.apple.com/documentation/uikit/uisearchdisplaycontroller>



INSTITUTO TECNOLÓGICO DE NUEVO LAREDO

CON LA CIENCIA POR LA HUMANIDAD

INGENIERÍA EN SISTEMAS COMPUTACIONALES



Programación móvil 2

Tema 4 y 5



DOCENTE

Ing. Humberto Peña Valle M.T.I.



INTEGRANTES

Jessica Janet Grajeda Castellanos

17100229

Juan Felipe Garza Sánchez

17100218

Miguel Ángel Méndez Cruz

17100254

INDICE

Explicación general del contenido del tema	316
4.1 Funcionalidad del modelado y manejo de datos locales con Core Data	317
5.1 Sensores en dispositivos móviles de iOS	325
5.2 Gestos en dispositivos móviles de iOS	331
Conclusiones	335
Bibliografía	336

EXPLICACIÓN GENERAL DEL CONTENIDO DEL TEMA

En estos dos temas se verá el manejo de datos en donde en el primero se vera la funcionalidad del modelado y el manejo de datos locales con Core Data, en donde veremos el fundamento teórico donde se explica el funcionamiento del framework de Core Data, de igual manera se verán todos componentes que forman parte de Core Data, la descripción de cada uno de ellos y como se relacionan con los demás componentes, esta información incluye diagramas, por ultimo del primer tema se da una explicación documentada con código de altas y bajas con Core Data.

El siguiente tema se habla sobre los sensores y gestos que incluye un dispositivo IOS, y la funcionalidad de cada uno de ellos, así como también en el caso de los gestos se muestra con imágenes.



4.1 FUNCIONALIDAD DEL MODELADO Y MANEJO DE DATOS LOCALES



Fundamento teórico

Core Data es el principal framework de persistencia de iOS. Nos proporciona un mecanismo para poder almacenar de forma persistente los objetos de nuestra aplicación y luego recuperarlos. Es capaz de guardar automáticamente un grafo entero de objetos con las relaciones (uno a uno, uno a muchos, muchos a muchos) que hay entre ellos. Si bien es un API complejo y tiene una curva de aprendizaje pronunciada, para cualquier aplicación medianamente grande usar Core Data nos va a resultar mucho más sencillo que implementar nosotros la persistencia de manera manual con, digamos, SQLite.

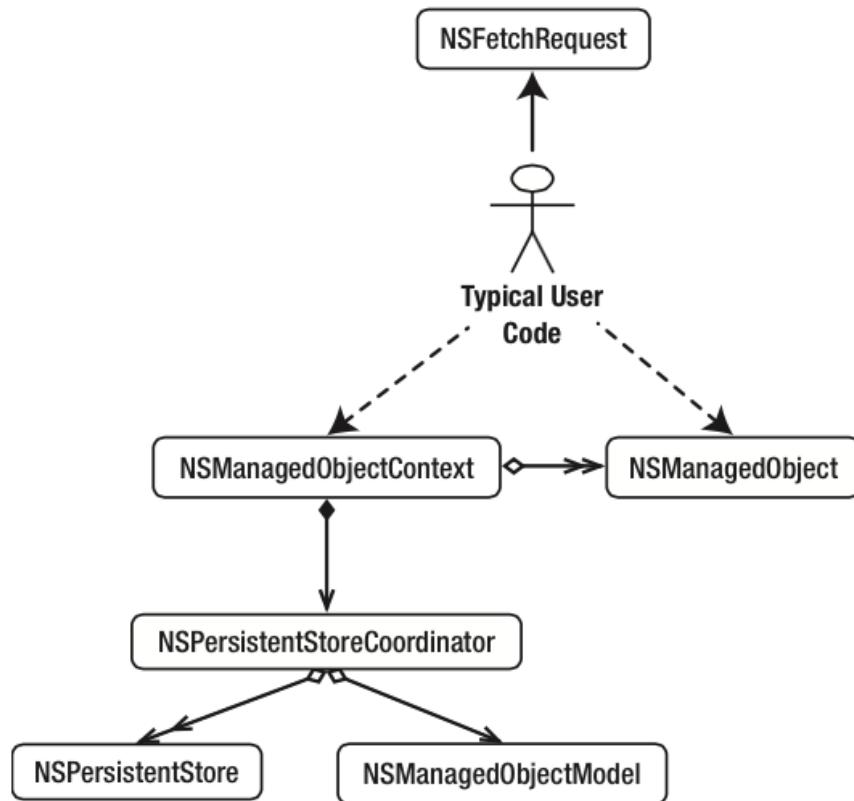
Core Data se puede considerar en cierto modo como un ORM (object relational mapper). Pero no es un ORM estrictamente hablando, ya que su objetivo no es la persistencia únicamente en bases de datos relacionales. En principio el backend de persistencia podría ser cualquiera. Con la implementación actual Core Data admite como única BD relacional para persistencia SQLite, y también puede almacenar datos en memoria y en XML (esto último solo en OSX).

Algunas de las ventajas y herramientas que tiene core data:

- Herramientas de migración de esquemas que simplifican los cambios de esquemas y le permiten realizar una migración de esquemas.
- Integración opcional con la capa del controlador de la aplicación para admitir la sincronización de la interfaz de usuario.
- Agrupar, filtrar y organizar datos en la memoria y en la interfaz de usuario.
- Soporte automático para almacenar objetos en repositorios de datos externos.

Compilación de consultas sofisticadas. En lugar de escribir SQL, puede crear consultas complejas asociando un objeto NSPredicate con una solicitud de recuperación

Componentes del Core Data



Core Data Stack

Managed Object (NSManagedObject)

Es una representación de un registro de una table de la base de datos

- Representa un registro de datos.
- Podemos implementar clases fuerte mente tipeadas usando subclases
- Podemos modificar las clases al gusto, por ejemplo, añadiendo nuevos métodos

Managed Object Context (NSManagedObjectContext)

- Es una colección de Managed Objects
- Altas, bajas y modificaciones son realizadas por este objeto
- Undo/Redo, validaciones, etc.

Managed Object Model (NSManagedObjectModel)

- Define el modelo de datos
- Un conjunto de entidades
- Permite definir relaciones entre entidades
- Se crea a través de XCode de modo visual (fichero .xcdatamodel)
- Ejecuta las consultas, borrados, actualizaciones, inserciones, etc...

Persistent Store Coordinator (NSPersistentStoreCoordinator)

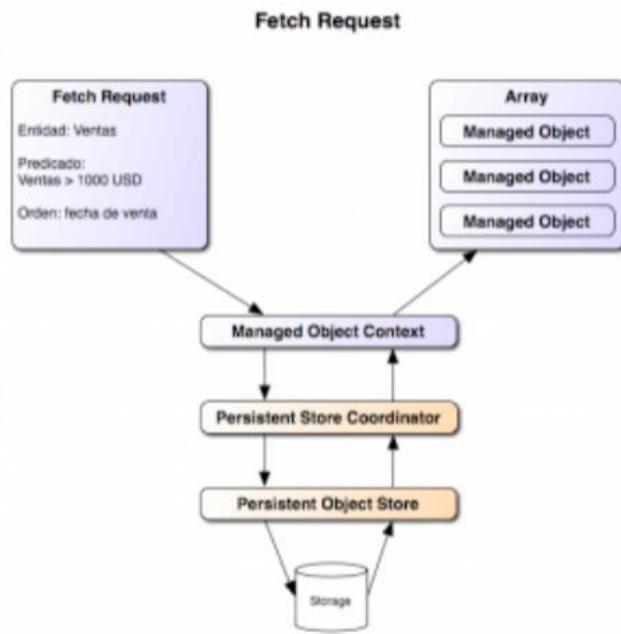
Podría considerarse como el núcleo de Core Data. Es la clase responsable de gestionar la persistencia y por eso tiene que interactuar con el NSPersistentStore.

- Colección de Persistent Object Store
- No interactuamos directamente

Persistent Object Store (NSPersistentObjectStore)

- Representa un repositorio
- Podemos implementar uno propio

Esquema de funcionamiento de las consultas (FetchRequest)



NSFetchRequest:

- Es una consulta de **ManagedObject**s sobre un **ManagedObjectContext** (Equivale a un **select**)
- Podemos usar predicados para hacer filtros: **NSPredicate**
- Podemos usar descriptores para ordenar los datos: **NSSortDescriptor**
- El resultado de un **FetchRequest** será un array de **ManagedObject**s

NSFetchResultsController:

- Controlador que permite que la aplicación funcione de forma semiautomática cuando hay cambios en los datos de CoreData.
- Muy útil cuando trabajamos con tablas (UITableView)

Explicación documentada de un ejemplo de altas y bajas

Objetos gestionados

Las instancias de NSManagedObject representan un registro en la tienda de respaldo de Core Data. Para revisar la analogía de la base de datos, una instancia de NSManagedObject contiene la información de una fila en una tabla de base de datos.

La razón por la que Core Data usa NSManagedObject en lugar de NSObject como su clase base para modelar registros tendrá más sentido en la documentación que veremos.

NSEntityDescription

Cada instancia de NSManagedObject está asociada a una instancia de NSEntityDescription. La descripción de la entidad incluye información sobre el objeto administrado, como la entidad del objeto administrado, así como sus atributos y relaciones.

NSManagedObjectContext

Un objeto administrado también está vinculado a una instancia de NSManagedObjectContext. El contexto de objeto administrado al que pertenece un objeto administrado supervisa el objeto administrado para detectar cambios.

Crear un registro

Para asegurarte de que un objeto administrado está configurado correctamente, se recomienda usar el inicializador designado para crear nuevas instancias de NSManagedObject.

Se trabajará en la clase de delegado de la aplicación, AppDelegate. Abre AppDelegate.swift y actualiza la implementación de application(_: didFinishLaunchingWithOptions:) como se muestra a continuación.

```
func application(application: UIApplication, didFinishLaunchingWithOptions launchOptions: [NSObject: AnyObject]?) -> Bool {
    // Create Managed Object
    let entityDescription = NSEntityDescription.entityForName("Person",
inManagedObjectContext: self.managedObjectContext)
    let newPerson = NSManagedObject(entity: entityDescription!,
insertIntoManagedObjectContext: self.managedObjectContext)

    return true
}
```

Lo primero que hacemos es crear una instancia de la clase `NSEntityDescription` invocando `entityForName(_:inManagedObjectContext:)`. Pasamos el nombre de la entidad para la que queremos crear un objeto

Necesitamos indicarle a Core Data dónde puede encontrar el modelo de datos para esa entidad. Un contexto de objeto administrado está vinculado a un coordinador de tienda persistente y un coordinador de tienda persistente mantiene una referencia a un modelo de datos. Cuando pasamos en un contexto de objeto administrado, Core Data pide a su coordinador de tienda persistente su modelo de datos para encontrar la entidad que estamos buscando.

En el segundo paso, invocamos el inicializador designado de la clase `NSManagedObject`, `init(entity:insertIntoManagedObjectContext:)`. Pasamos la descripción de la entidad y una instancia de `NSManagedObjectContext`.

```
// Configure New Person
newPerson.setValue("Bart", forKey: "first")
newPerson.setValue("Jacobs", forKey: "last")
```

Guardar un registro

Ya tenemos una nueva instancia de persona, Core Data aún no ha guardado a la persona en su tienda de respaldo. El objeto administrado que creamos solo vive en el contexto de objeto administrado en el que se insertó. Para guardar el objeto persona en la tienda de respaldo, necesitamos guardar los cambios del contexto del objeto administrado llamando a `save()` en él.

El método `save()` es un método de lanzamiento y devuelve un booleano para indicar el resultado de la operación de guardar.

```
do {
    try newPerson.managedObjectContext?.save()
} catch {
    print(error)
}
```

Eliminación de registros

La eliminación de un registro sigue el mismo patrón. Le indicamos al contexto del objeto administrado que un registro debe eliminarse del almacén persistente invocando `deleteObject(_:)` y pasar el objeto administrado que necesita ser eliminado.

En nuestro proyecto, elimina el objeto persona que obtuvimos anteriormente pasándolo al método `deleteObject(_:)` del contexto del objeto administrado. Ten en cuenta que la operación de eliminación no se confirma en la tienda de respaldo hasta que llamemos a `save()` en el contexto del objeto administrado.

```
let person = result[0] as! NSManagedObject  
  
self.managedObjectContext.deleteObject(person)  
  
do {  
    try self.managedObjectContext.save()  
} catch {  
    let saveError = error as NSError  
    print(saveError)  
}
```

5.1.1 SENsoRES EN DISPOSITIVOS MÓVILES DE IOS



Acelerómetro

El acelerómetro es uno de los sensores más destacados de los dispositivos iOS, mediante el podemos controlar los distintos movimientos que hagamos. Según la definición es: "Usado para determinar la posición de un cuerpo, pues al conocerse su aceleración en todo momento, es posible calcular los desplazamientos que tuvo. Considerando que se conocen la posición y velocidad original del cuerpo bajo análisis, y sumando los desplazamientos medidos se determina la posición.".



Para obtener los datos del acelerómetro debemos usar la clase `UIAccelerometer` de la API. Esta clase es la responsable de registrar los movimientos en los ejes x, y, z que se producen en nuestro dispositivo iOS. En la definición de la clase, como podemos ver, hemos definido el protocolo `UIAccelerometerDelegate`. También hemos definido dos propiedades, una es la imagen de la pelota que se va a mover en la pantalla y la otra es el punto x,y que nos devuelve la clase `UIAccelerometer`.

Giroscopio

Un giroscopio es un torno de hilar, en un alojamiento que puede girar en una dirección perpendicular a una base fija. Las leyes físicas del momento angular permiten un giroscopio para proporcionar información precisa acerca de las fuerzas aplicadas en contra de ella, como por ejemplo cuando el iPad se mueve o gira. Un giroscopio proporciona información a lo largo de un eje único (por ejemplo, horizontal o vertical), por lo que un IPAD utiliza múltiples giroscopios para mantener la información en tres dimensiones sobre sí mismo.

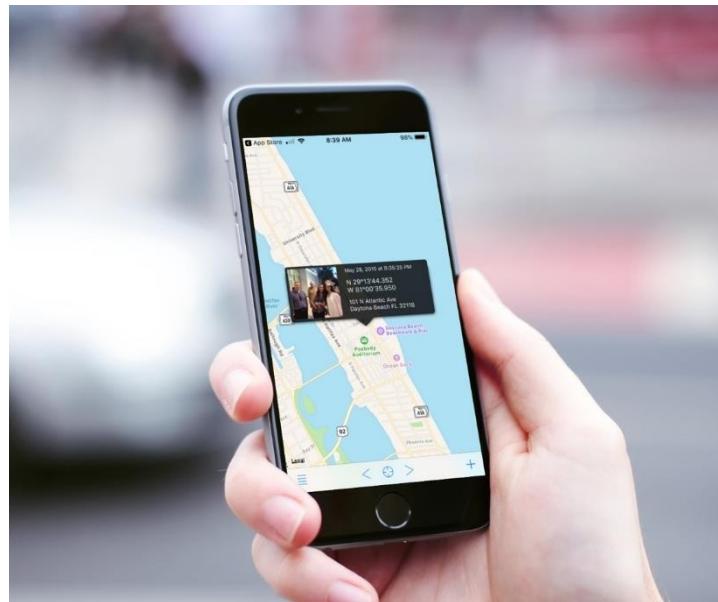


La orientación giroscópica también es importante para muchos tipos de aplicaciones de iPad. Muchos juegos ofrecen una superficie de control adicional mediante la medición de todo el iPad, además de toques de pantalla; un juego de conducción, por ejemplo, puede dirigir el coche mediante la rotación de todo el dispositivo. La orientación también se utiliza para medir ángulos de algo de fotografía avanzada y aplicaciones 3-D, que graba los datos sobre la posición del iPad con las fotos se rompió.

GPS

La Localización permite que las aplicaciones y sitios web basados en la ubicación (incluyendo Mapas, Cámara, Safari y otras apps de Apple y de terceros) utilicen información de redes móviles, Wi-Fi y el sistema de posicionamiento global (GPS) para determinar tu ubicación aproximada.

Por ejemplo, una aplicación podría utilizar tus datos de localización y consultas de búsqueda de localización para ayudarte a encontrar cafeterías o cines cercanos, o tu dispositivo podría configurar su zona horaria automáticamente según tu ubicación actual.



Tanto el giroscopio y acelerómetro son cruciales para servicios de localización, el sistema IOS para proporcionar su ubicación a la cartografía, y otro software de geolocalización basada en la localización. La radio GPS puede identificar una ubicación en la red de satélites, pero este proceso lleva tiempo y está limitado a una precisión de 10 metros. Servicios de Localización combina estos datos con información sobre los puntos calientes Wi-Fi para llegar a una solución precisa y luego utiliza el giroscopio y acelerómetro para seguir el movimiento desde la última ubicación fija. La ubicación de edad, cuando se añade a los cálculos de la velocidad y dirección, inmediatamente proporciona la ubicación actual en el IPAD, que puede entonces ser doble cotejarse con más lenta, el seguimiento de ubicación de red.

Brújula

El sensor de la brújula (compass) apareció a partir de la versión de iPhone 3GS. Mediante ella podemos situarnos en el mapa de una forma más precisa. En la aplicación de Mapas la brújula rotará el mapa según la dirección en donde estemos mirando. En aplicaciones de realidad aumentada también es útil la brújula. Si combinamos los datos obtenidos con la brújula (yaw) junto con los que obtenemos del acelerómetro (roll y pitch) conseguiremos determinar de una manera bastante precisa la orientación del dispositivo en tiempo real.



Sensor de Proximidad y Luz ambiental

El sensor de proximidad y sensor de luz ambiente son los mismos. Este sensor es el encargado de reducir la intensidad de la pantalla o incluso apagarla cuando acercamos algún objeto a él. Se utiliza por ejemplo al hablar por teléfono: cuando recibimos una llamada al iPhone y acercamos la oreja al dispositivo, este se apaga.

Hasta hace poco no existía una API pública que permitiera obtener información de este sensor, pero ahora sí, aunque no se suele utilizar demasiado sí que puede servirnos para determinados casos sobre todo en la programación de videojuegos. Para hacer uso del sensor de proximidad deberemos acceder al mediante la propiedad `proximityState` de la clase singleton del dispositivo `UIDevice`. Esta propiedad nos devolverá un valor simple (`true/false`) no un valor variable y sólo está disponible en iPhone e iPad, no en las versiones de iPod Touch.



5.2 GESTOS EN DISPOSITIVOS MÓVILES DE IOS



Tap

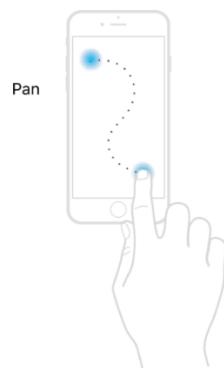
Tap (UITapGestureRecognizer): cuando el usuario toca con uno o varios dedos la pantalla en un punto. Permite que el toque sea único, doble, etc. Sería el clásico clic



Tap

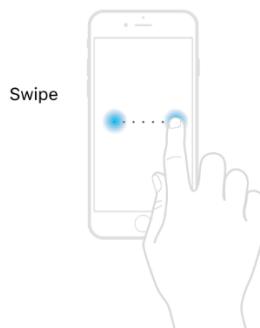
Pan/Drag

Pan / Drag (UIPanGestureRecognizer): cuando el usuario toca con uno o varios dedos y sin soltar arrastra el dedo por la pantalla. Sería el gesto que haríamos cuando queremos arrastrar un objeto de un sitio a otro.



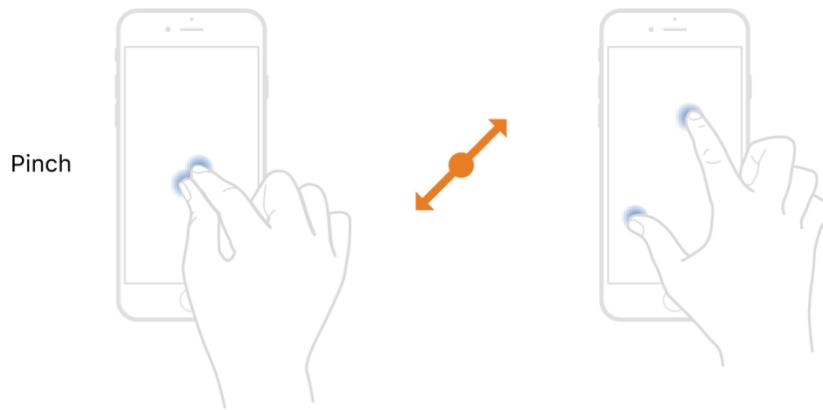
Swipe

Swipe (UISwipeGestureRecognizer): cuando el usuario pulsa con su dedo y desplaza de forma rápida hacia la derecha o hacia la izquierda. Es el gesto que utilizamos en una UITableView cuando queremos que se muestre el botón borrar de dicha fila.



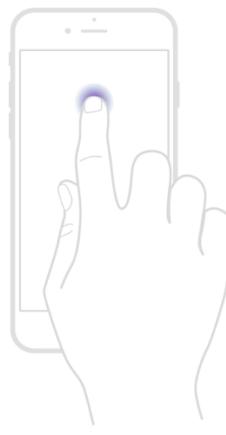
Pinch

Pinch (UIPinchGestureRecognizer): cuando el usuario, normalmente con una mano pulsa con dos dedos la pantalla y a continuación los separa el uno del otro, o los aproxima. Es el gesto utilizado para hacer zoom in y out en las fotografías.



Touch and hold

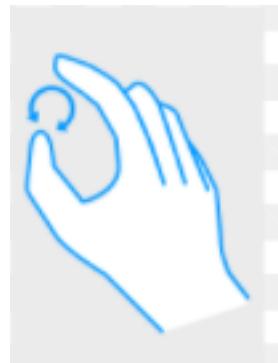
Touch and hold (UILongPressGestureRecognizer): significa que el usuario pulsa sobre la pantalla y mantiene la pulsación en ese punto durante un periodo de tiempo. Es el gesto que se utiliza si estamos en Safari donde al dejar una fotografía de la web pulsada, el sistema nos muestra un menú para poder guardar la imagen



Long press
(>0.5 seconds)

Rotation

Rotation (UIRotationGestureRecognizer): es cuando dos dedos se mueven en dirección contraria de las agujas del reloj. Es el gesto que utilizamos para rogar imágenes en algunas aplicaciones.



CONCLUSIONES

Juan Felipe Garza Sánchez

A mi parecer es de suma importancia conocer por lo menos la existencia de cada uno de los sensores que involucra los dispositivos móviles de Apple, pues es aquí donde surgen las distintas propuestas que nosotros podremos tener como ingenieros en sistemas computacionales ante una problemática en específica, dando a conocer las distintas alternativas y el cuál es la mejor opción a la hora de desarrollar una aplicación para atender una necesidad del cliente.

Ahora hablando del software y el gran soporte que tienen los dispositivos móviles de Apple al día de hoy para captar los distintos gestos que podemos realizar en la pantalla nos vemos en la necesidad de nosotros como desarrolladores de aplicaciones móviles el saber conocer las diversas formas que tenemos para capturar un evento de gesto en nuestra aplicación pues nos permitirá dar un alcance más profesional a la aplicación ya que estamos evitando el agregar controles de manera sumamente excesiva y terminar dando un mal terminado en la interfaz gráfica del usuario.

El conocer la manera en que Core Data permite el almacenamiento interno de datos, tanto los beneficios que puede traer y sus debidas limitantes, me permitieron a mi saber cuándo y cómo utilizar este tipo de estrategias de almacenamiento en una aplicación móvil.

Jessica Janet Grajeda Castellanos

Como conclusión a mi parecer me resultó muy interesante la cantidad de gestos que es posible habilitar en nuestras aplicaciones, ya que conocía algunas sin embargo no tenía conocimiento de otras. Además de esto, Core Data me pareció uno de los temas más importantes en mi opinión de este curso, si bien es indispensable poner controles y programarlos, uno de los recursos más importantes de una aplicación si no es el más importante, son los datos que se manejan ya sea por la misma aplicación o por los datos que el usuario genera o proporciona. Esto también nos ayuda a crear aplicaciones completas y que sean útiles para los usuarios.

Miguel Ángel Méndez Cruz

Me parece que Core Data es una herramienta muy útil para el manejo de datos en Swift, como el uso de bases de datos y acerca de los sensores me parece que tiene mucha variedad como hacer aplicaciones muy útiles para la vida diaria de los usuarios de Swift, hoy en día muchas aplicaciones utilizan muchas de ellas y eso las hace interesantes y funcionales, así como atractivas, también los gestos son herramientas muy útiles para hacer mucho más interactivas las aplicaciones.

BIBLIOGRAFÍA | LINKOGRAFÍA

- <http://www.migueliazrubio.com/desarrollo-ios-introducir-gestos-en-tu-aplicacion/>
- <http://jtech.ua.es/dadm/restringido/persistencia/traspas/sesion04-persistencia.pdf>
- https://mastermoviles.gitbook.io/persistencia-en-dispositivos-m-viles-ios/intro-2/3.1_intro
- <https://code.tutsplus.com/es/tutorials/core-data-and-swift-managed-objects-and-fetch-requests--cms-25068>
- <https://lamanzanamordida.net/tutoriales/apple-watch/sensores-apple-watch-explicados/>
- https://cincodias.elpais.com/cincodias/2021/04/06/lifestyle/1617704176_647606.html
- <https://mastermoviles.gitbook.io/interfaz-de-usuario-en-dispositivos-moviles-ios-a/sesion-4-multitouch>
- <https://www.iphoneworld.com.es/2013/10/tap-scroll-swipe-pinch-gestos-tactiles.html>
- <http://www.jtech.ua.es/dadm/restringido/sensores/sesion03-apuntes.pdf>