

Metodología y Ciclo de Vida Desarrollo de Software

Metodología, ciclo de vida y herramientas de desarrollo en la WEB

Jonatan Gómez Perdomo, Ph. D.

jgomezpe@unal.edu.co

Arles Rodríguez, Ph.D.

aerodriguezp@unal.edu.co

Camilo Cubides, Ph.D. (c)

eccubidesg@unal.edu.co

Carlos Andres Sierra, M.Sc.

casierrav@unal.edu.co

Research Group on Artificial Life – Grupo de investigación en vida artificial – (Alife)

Computer and System Department

Engineering School

Universidad Nacional de Colombia

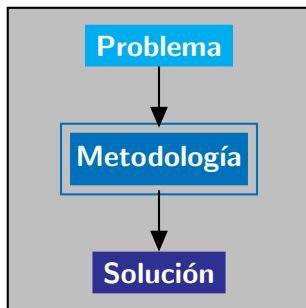
Agenda

- 1 Metodología de problemas algorítmicos
- 2 Ciclos de Vida
- 3 Modelos de Ciclo de Vida
- 4 Metodologías de Desarrollo
- 5 Programación en Python
 - Herramientas web para desarrollar
 - Colab
 - Manejo básico de Colab
 - Mi primer programa en Python
 - Repl.it



De problema a solución

Dado un problema, para encontrar la solución a dicho problema se puede pensar en utilizar una metodología general que se aplique sistemáticamente de tal manera que como resultado se obtenga la solución del problema, así como se muestra en el siguiente diagrama.



Metodología de programación I

Una metodología usualmente utilizada para resolver problemas algorítmicos está compuesta por la siguiente serie de etapas:

- 1 Análisis del problema
- 2 Especificación del problema
- 3 Diseño del algoritmo
- 4 Prueba del algoritmo y refinamiento
- 5 Codificación
- 6 Prueba y verificación



Metodología de programación II

Análisis del problema: Entender de manera clara el problema que se está resolviendo. Esto involucra el proceso de identificación de objetos conocidos, objetos desconocidos y condiciones del problema.

Especificación del problema: Descripción clara y precisa de

- Las entradas del problema.
- Las salidas del problema.
- Las condiciones, es decir la dependencia que mantendrán las salidas obtenidas con las entradas recibidas.



Metodología de programación III

Diseño del algoritmo: Es la fase en la que se construye el algoritmo que permitirá encontrar la solución al problema. Es una buena practica de programación subdividir el problema en problemas más sencillos de resolver.

Prueba del algoritmo y refinamiento: Proceso de seguimiento del algoritmo para verificar que cumple con la especificación. Si no se cumple con la especificación se va refinando hasta lograr el objetivo.



Metodología de programación IV

Codificación: Proceso en el cual se escribe el algoritmo en un lenguaje de programación, utilizando alguna herramienta de programación.

Prueba y verificación: Proceso en el cual se corrigen los errores de sintaxis y de lógica del programa, hasta lograr que el programa resuelva el problema de forma exitosa.



Metodología de programación V

Ejercicio

Utilizar la metodología de programación presentada previamente para resolver el siguiente problema bien condicionado ya conocido.

Ejemplo

Un granjero tiene cincuenta animales entre conejos y gallinas. Si la cantidad de patas de los animales es ciento cuarenta, ¿Cuántos conejos y cuántas gallinas tiene el granjero?.



Agenda

- 1 Metodología de problemas algorítmicos
- 2 Ciclos de Vida
- 3 Modelos de Ciclo de Vida
- 4 Metodologías de Desarrollo
- 5 Programación en Python
 - Herramientas web para desarrollar
 - Colab
 - Manejo básico de Colab
 - Mi primer programa en Python
 - Repl.it



Ciclo de Vida del Software

- 1 Es una manera de controlar el avance y maduración de un proyecto de software al dividirlo en etapas de acuerdo a la concentración de tareas y roles del equipo de trabajo.
- 2 Existen varios tipos de ciclo de vida, los cuales cambian la forma y momentos de abordar tareas. Sin embargo, la selección del ciclo de vida depende de las características del proyecto, tamaño del equipo, producto final a entregar, entre otros.
- 3 El ciclo de vida suele ajustarse a la metodología de desarrollo a utilizar, en donde siempre la finalidad es llegar a productos de alta calidad con los tiempos y costos estimados.



Fases del Ciclo de Vida del Software I

En general, los ciclos de vida tienen siete fases:

Planeación: En esta fase se pretende identificar claramente el producto que se quiere desarrollar. En este caso, se deben tener reuniones, consensos, levantamientos de información, con los clientes y todos los involucrados en el proyecto. Acá un hito importante es entender las expectativas del producto.

Análisis de Requerimientos: Una vez se entiende el producto a desarrollar, se debe construir una especificación de requerimientos. Esta especificación es donde se detallan las características del producto, y es prácticamente el contrato con el que se contrasta al final del proyecto para determinar que se cumplió con el producto esperado.



Fases del Ciclo de Vida del Software II

Diseño: En esta fase ya intervienen más miembros técnicos del equipo, en particular los arquitectos y los líderes técnicos. Se toman los requerimientos y se hacen especificaciones desde un punto de vista de la tecnología, medición de esfuerzos, herramientas a utilizar, entre otros.

Implementación: Es la etapa de desarrollo del producto, lo que también se conoce como programar o codificar. Es la fase que se considera más consume tiempo, debido al esfuerzo en términos de trabajo que implica. Aquí entra en juego el correcto uso de lenguajes, herramientas, procesos de trabajo en equipo, comunicación, ajustes de infraestructura, entre otras actividades que hacen de esta fase una de las más sensibles a fallas.



Fases del Ciclo de Vida del Software III

Pruebas: Se realizan sobre el producto cuando está terminado parcial o totalmente. Estas pruebas se enfocan en medir la calidad de producto, a nivel funcional y de desempeño, y la idea principal es encontrar fallas para retroalimentar rápidamente al equipo de implementación.

Despliegue: Cuando se considera el producto está listo para ser mostrado al cliente, se coloca en un medio de fácil acceso y manipulación para hacer la entrega de producto. También existen casos donde se trabaja con versiones denominadas como “Beta”.

Mantenimiento y Mejoras: Siempre luego de un despliegue se tienen retroalimentaciones tanto del cliente, como de fallas que suceden a nivel de producción. Todo eso implica nuevos desarrollos o ajustes.



Aspectos Importantes

Se deben controlar aspectos a lo largo del ciclo de vida con el fin de disminuir el fracaso del proyecto. Entre las causas comunes de fracaso se tienen las siguientes:

- Mala comunicación entre el equipo de trabajo.
- Mala estimación de costos y tiempos para el desarrollo del proyecto.
- Baja calidad en el producto desarrollado a nivel de código.
- No cumplir con los momentos claves del cronograma o con entregas parciales.
- Mal manejo de la documentación, esto debe ser equilibrado.



Agenda

- 1 Metodología de problemas algorítmicos
- 2 Ciclos de Vida
- 3 Modelos de Ciclo de Vida**
- 4 Metodologías de Desarrollo
- 5 Programación en Python
 - Herramientas web para desarrollar
 - Colab
 - Manejo básico de Colab
 - Mi primer programa en Python
 - Repl.it



Modelos de Ciclo de Vida - Cascada

El modelo en cascada es el modelo clásico de ciclo de vida de software. Creado en la década de 1960, asume todas las fases de manera lineal, es decir, cada fase inicia una vez se hayan terminado las anteriores.

Aunque es muy sencillo de comprender y de poner en práctica, este modelo no es flexible a ciertos cambios durante el desarrollo del proyecto. Esto implica que no siempre se logren cumplir las expectativas del cliente, a pesar de que el modelo asegura tener un producto completo al final.

Análisis

Diseño

Implementación

Pruebas

Despliegue

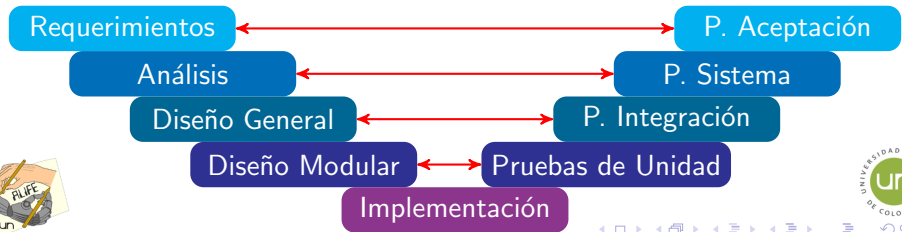
Mantenimiento



Modelos de Ciclo de Vida - Modelo V

El modelo en V es una versión mejorada del ciclo de vida en cascada, en donde para cada fase existe un equivalente de pruebas y validaciones que ejecutadas en dicho orden van dando puntos de chequeo que apuntan a alta calidad del producto.

Este modelo también es simple de comprender, pero no siempre de implementar debido a la alta concentración de pruebas que tiene, lo que también puede llevar a sobre documentación. Sin embargo, desde el punto de vista de pruebas es un modelo bastante utilizado.



Modelos de Ciclo de Vida - Ciclo Iterativo

En este modelo la idea es dividir el proyecto completo en pequeños entregables o grupos de funcionalidades completadas, lo que se conoce como iteraciones. Cada iteración suele tener las fases de planeación, diseño, implementación, pruebas y despliegue. Acá aparece con fuerza el concepto de versionamiento.

Este modelo es más flexible al cambio que sus predecesores, y genera entregables de software que permiten una rápida retroalimentación de los clientes. Sin embargo, no se tiene tanto control global sobre el desarrollo del proyecto, incluso llegando a no cumplir todos los requerimientos.



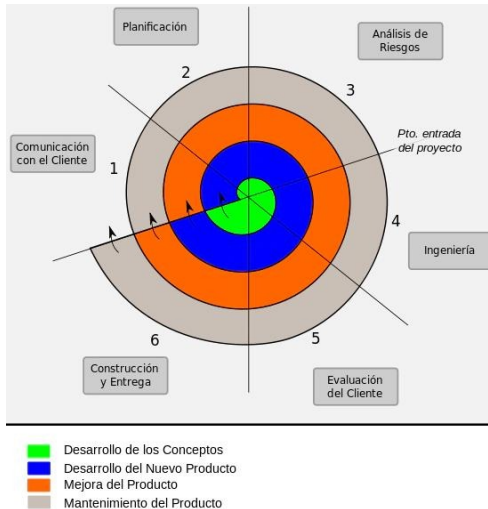
Modelos de Ciclo de Vida - Ciclo en Espiral I

La entrega de los productos de software se basa de nuevo en el concepto de iteraciones. Cada iteración de la espiral tiene las fases de identificación (similar a planeación), diseño, construcción y evaluación. En esencia busca que de manera incremental se construya el software basado en cambios detectados a tiempo.

Este modelo fue el primero en involucrar de manera sistemática la identificación y mitigación de riesgos. Al final de cada vuelta de la espiral la evaluación no solo busca medir la calidad, sino determinar potenciales fallas para tener en cuenta en la siguiente vuelta de la espiral.



Modelos de Ciclo de Vida - Ciclo en Espiral II



<https://sites.google.com/site/proyectoadpmodelosdedesarrollo/home/modelos-de-desarrollo>



Modelos de Ciclo de Vida - Ciclo Ágil

Es actualmente el ciclo de vida más popular, desarrollado al inicio del presente milenio. Inicialmente se consideró como una serie de principios, poco a poco se convirtió en un ciclo en donde lo importante es la satisfacción del cliente, la facilidad al cambio, retroalimentación constante, y versatilidad del equipo.

Este modelo es muy poderoso en equipos pequeños de trabajo, pero tiene desventajas en la medida que se genera poca documentación formal, y que al permitir los cambios en requerimientos, es difícil hacer una estimación real de costos y esfuerzos.



Agenda

- 1 Metodología de problemas algorítmicos
- 2 Ciclos de Vida
- 3 Modelos de Ciclo de Vida
- 4 Metodologías de Desarrollo
- 5 Programación en Python
 - Herramientas web para desarrollar
 - Colab
 - Manejo básico de Colab
 - Mi primer programa en Python
 - Repl.it



Metodologías de Desarrollo

- Acompañando a los ciclos de vida del software se encuentran las metodologías de desarrollo, las cuales no solo controlan aspectos técnicos del proyecto, sino también buscan manejar temas como comunicación, manejo de costos, manejo de recurso humano, entre otros aspectos que van más allá de la parte técnica y tecnológica del desarrollo de un producto de software.
- Existen metodologías tradicionales, en donde el manejo documental, de cronogramas, reclutamiento y construcción de equipos de trabajo, se rigen en procesos bastante estrictos. Por otro lado, existen las metodologías ágiles, basadas en el manifiesto ágil y ciclo de vida ágil, buscan ser flexibles a cambios por parte del cliente o modificaciones al modelo de negocio (*startup*).



Metodología SCRUM I

- Es la metodología de desarrollo ágil más utilizada actualmente. Basada en el ciclo de vida ágil, y por ende, los principios del manifiesto ágil.
- En esta metodología se trabaja con ventanas de tiempo llamadas "*sprint*". Cada *sprint* se realiza una iteración del ciclo de vida ágil, y normalmente la duración de un *sprint* es de dos semanas.
- La metodología está orientada a tener nuevas funcionalidades de código o correcciones al final de cada *sprint*, teniendo pequeños entregables de manera continua.
- La interacción con el cliente es vital en esta metodología, para recibir retroalimentación del entregable de cada *sprint*.



Metodología SCRUM II

- En la metodología *scrum* se suelen manejar tres roles esenciales dentro del equipo de trabajo: *product owner*, *scrum master* y *scrum team*.
- Suele basarse en el concepto de tablero *kanban*, uso de *backlog*, *tickets*, *daily meetings*, entre otros, para hacer la gestión del proyecto.
- El *product owner* es quien más interactúa con los clientes y externos al equipo de desarrollo. Suele ser quien crea nuevos requerimientos en el *backlog*.
- El *scrum master* toma la información consignada en el *backlog* y la divide en pequeñas tareas a través de *tickets* (*user stories*).



Agenda

- 1 Metodología de problemas algorítmicos
- 2 Ciclos de Vida
- 3 Modelos de Ciclo de Vida
- 4 Metodologías de Desarrollo
- 5 Programación en Python**
 - Herramientas web para desarrollar
 - Colab
 - Manejo básico de Colab
 - Mi primer programa en Python
 - Repl.it



Python

- Python es un lenguaje poderoso y fácil de aprender. Posee estructuras de datos de alto nivel y un enfoque simple de orientación a objetos.
- Python permite dividir los programas en módulos que pueden ser reutilizados por otros programas.
- Adicionalmente, viene con una gran colección de módulos propios para realizar tareas de entrada y salida, llamadas al sistema, *sockets*, interfaces gráficas, etc.
- Python es un lenguaje interpretado. Esto quiere decir que para ejecutar un programa en este lenguaje se requiere la instalación de un intérprete y se ejecuta línea a línea.
- El intérprete puede ser usado interactivamente, lo cual hace fácil experimentar con características del lenguaje.



Python como lenguaje

- Automatizar tareas es posible gracias a los lenguajes de programación.
- En ese sentido, entenderemos en este curso que programar es decirle al computador que haga lo que uno necesita que haga.
- Para comunicarnos con la máquina necesitaremos entender a Python como un lenguaje.



Algoritmos y resolución de Problemas

- Un algoritmo es una finita secuencia de pasos que resuelve un problema.
- A parte del dominio que se espera de un lenguaje, se pretende que el estudiante desarrolle una perspectiva algorítmica para solucionar problemas utilizando el computador.



Agenda

- 1 Metodología de problemas algorítmicos
- 2 Ciclos de Vida
- 3 Modelos de Ciclo de Vida
- 4 Metodologías de Desarrollo
- 5 Programación en Python**
 - Herramientas web para desarrollar
 - Colab
 - Manejo básico de Colab
 - Mi primer programa en Python
 - Repl.it



Herramientas web para desarrollar

En este curso se presentarán dos opciones para programar en Python:

Colab: Una versión en línea que provee Google.

<https://colab.research.google.com/>

repl.it: Una página web que permite desarrollar código en diferentes lenguajes de programación.

<https://replit.com/languages/python3>



Colaboratory (Colab)

Es una herramienta web de Google que permite ejecutar código en Python:

- Sin realizar configuración adicional.
- Con acceso gratis a GPU.
- Es fácil de compartir.

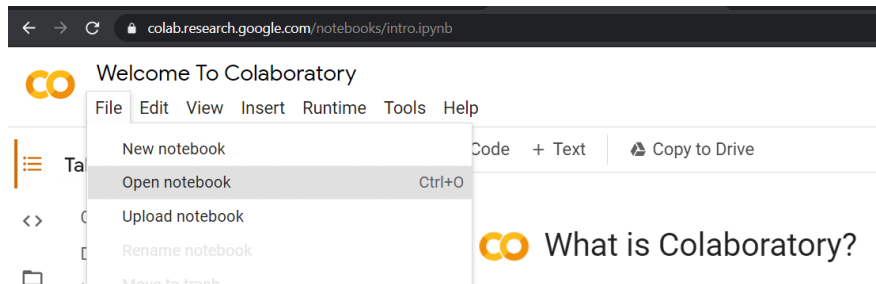
Los códigos en Colab se conocen como notebooks.



Cargar el notebook de esta sesión I

Para cargar este notebook se debe:

- 1 Ingresar a <https://colab.research.google.com/notebooks/intro.ipynb#recent=true>



Cargar el notebook de esta sesión II

Para cargar este notebook se debe:

- 2 Seleccionar Github e ingresar los datos como se muestra a continuación abriendo el notebook señalado como **1-Tareas Básicas en Python.ipynb**:

ExamplesRecentGoogle DriveGitHubUpload

Enter a GitHub URL or search by organization or user☐ Include private repos

arleserp/MinTIC2022

Repository: Branch:

arleserp/MinTIC2022 master

Path

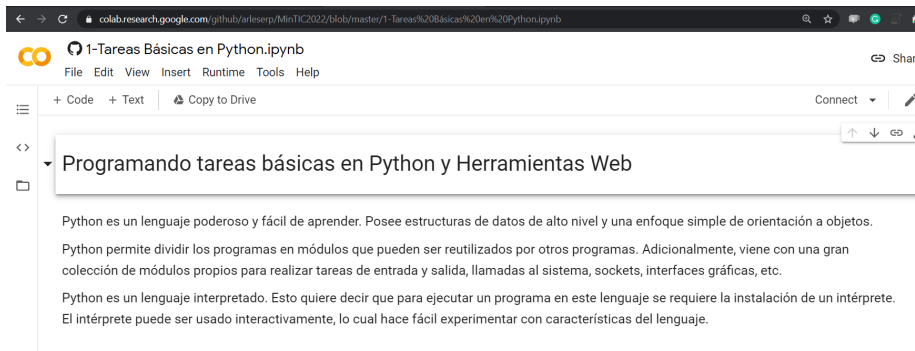
.ipynb_checkpoints/1-Tareas Básicas en Python-checkpoint.ipynb

1-Tareas Básicas en Python.ipynb

Open notebook

Cargar el notebook de esta sesión III

3 ¡Listo!:



The screenshot shows a Google Colab notebook interface. The browser address bar displays the URL: `colab.research.google.com/github/arleserp/MinTIC2022/blob/master/1-Tareas%20Básicas%20en%20Python.ipynb`. The notebook title is "1-Tareas Básicas en Python.ipynb". The menu bar includes File, Edit, View, Insert, Runtime, Tools, and Help. The toolbar shows options for adding code or text, copying to Drive, and connecting to Google Drive. The notebook content is titled "Programando tareas básicas en Python y Herramientas Web" and contains the following text:

Python es un lenguaje poderoso y fácil de aprender. Posee estructuras de datos de alto nivel y una enfoque simple de orientación a objetos.

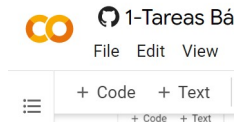
Python permite dividir los programas en módulos que pueden ser reutilizados por otros programas. Adicionalmente, viene con una gran colección de módulos propios para realizar tareas de entrada y salida, llamadas al sistema, sockets, interfaces gráficas, etc.

Python es un lenguaje interpretado. Esto quiere decir que para ejecutar un programa en este lenguaje se requiere la instalación de un intérprete. El intérprete puede ser usado interactivamente, lo cual hace fácil experimentar con características del lenguaje.



Introducción a Colab I

En Colab existen dos tipos de celda Texto y Código. Se pueden agregar utilizando los botones en la parte superior izquierda de la ventana de Colab.



Para ejecutar una celda se puede presionar el botón de *play* ó con el teclado escribir **Command/Ctrl+Enter**

En colab existen dos tipos de celda Texto y Código. Esta es una celda de texto.



Introducción a Colab II

El valor de una variable definida en una celda de código se puede usar en las celdas siguientes, por ejemplo, como `x` se había definido como `amigos`, se puede hacer lo siguiente:

```
[ ] print("Eso es todo " + x)
```

Eso es todo amigos



Celdas de texto

En los campos de texto se puede usar algo de HTML:

HTML

```
<table>
  <tr>
    <td>Nombre Estudiante</td><td>Nota</td>
  </tr>
  <tr>
    <td>Mike Myers</td><td>4.5</td>
  </tr>
  <tr>
    <td>Otm Shank</td><td>2.3</td>
  </tr>
</table>
```

Salida:

Nombre Estudiante	Nota
Mike Myers	4.5
Otm Shank	2.3



Celdas de texto

En las celdas de texto se puede usar también \LaTeX :

\LaTeX

```
 $\backslash\text{sum}_{\{i=1\}}^n i^2$ 
```

Salida:

$$\sum_{i=1}^n i^2$$



Mi primer programa en Python: componiendo reguetón

Vamos a mirar el primer minuto del siguiente video que *!está embebido en Colab!*:

```
[ ] from IPython.display import YouTubeVideo

YouTubeVideo("6i0lB0QLy84")
```



Algoritmo para generar reguetón

Pensemos en un minuto... Dado el video anterior: ¿Cuál es el algoritmo para generar una letra de reguetón?



Algoritmo para generar reguetón

Aquí vemos que el algoritmo para generar una frase de reguetón es explicado cuando don Melquiades muestra la tabla a continuación:

El abuelo Melquiades te enseña cómo componer reggaeton en tan solo 30 segundos - El Hormi...
#PaulaEchevarriaEH

CUADRO REGGAETÓN

MAMI	YO QUIERO	ENCENDERTE	SUAVE	HASTA QUE SALGA EL SOL	SIN ANESTESIA
BEBÉ	YO PUEDO	AMARTE	LENTO	TODA LA NOCHE	SIN COMPROMISO
PRINCESS	YO VENGO A	LIGAR	RÁPIDO	HASTA EL AMANECER	FEIS TO FEIS
MAMI	YO SOY A	JUGAR	FUERTE	TODO EL DÍA	SIN MIEDO

MORE VIDEOS

0:26 / 2:49

YouTube

UNIVERSIDAD NACIONAL DE COLOMBIA

Algoritmo para generar reguetón

El algoritmo que Don Melquiades usa consiste en:

- 1 Definir las categorías.
- 2 Seleccionar una palabra de cada categoría al azar.
- 3 Unir las palabras seleccionadas y mostrarlas al usuario.



Código para generar reguetón I

En Python la traducción del algoritmo que don Melquiades usa es:

① Definir las categorías:

```
import random # se importa la librería de python random

sujetos = ["mami", "bebé", "princess", "mami"] # se define una lista
intenciones = ["yo quiero", "yo puedo", "yo vengo a", "voy a"]
verbos = ["encendelte", "amalte", "ligal", "jugal"]
advs = ["suave", "lento", "lápid", "fuelle"]
complementos_uno = ["hasta que salga el sol", "toda la noche",
                    "hasta el amanecer", "todo el día"]
complementos_dos = ["sin anestesia", "sin compromiso", "feis to feis",
                    "sin miedo"]
```



Código para generar reguetón II

2 Seleccionar una palabra de cada categoría al azar:

```
sujeto_seleccionado = random.choice(sujetos)
                                # se utiliza la librería
intencion_seleccionada = random.choice(intenciones)
                                # para seleccionar un elemento
verbo_seleccionado = random.choice(verbos)
                                # al azar de la lista sujetos
adv_seleccionado = random.choice(advs)
compl1s_seleccionado = random.choice(complementos_uno)
compl2s_seleccionado = random.choice(complementos_dos)
```



Código para generar reguetón III

- ③ Unir las palabras seleccionadas y mostrarlas al usuario:

```
print("letra generada: " + sujeto_seleccionado + " "  
+ intencion_seleccionada + " " + verbo_seleccionado + " "  
+ adv_seleccionado + " " + compl1s_seleccionado + " "  
+ compl2s_seleccionado) # se imprime la canción
```

La salida sería como la siguiente:

```
letra generada: mami voy a encendelte fuelte todo el día feis to feis
```



Componiendo reagueton en Colab

El código en Colab se vería así:

```
import random #se importa la librería de python random

sujetos = ["mami", "bebé", "princess", "mami"] #se define una lista
intenciones = ["yo quiero", "yo puedo", "yo vengo a", "voy a"]
verbos = ["encendelte", "amalte", "ligal", "jugal"]
advs = ["suave", "lento", "lápido", "fuelle"]
complementos_uno = ["hasta que salga el sol", "toda la noche", "hasta el amanecer", "todo el día"]
complementos_dos = ["sin anestesia", "sin compromiso", "feis to feis", "sin miedo"]

sujeto_seleccionado = random.choice(sujetos) #se utiliza la librería para seleccionar un elemento al azar de la lista sujetos
intencion_seleccionada = random.choice(intenciones)
verbo_seleccionado = random.choice(verbos)
adv_seleccionado = random.choice(advs)
compl1s_seleccionado = random.choice(complementos_uno)
compl2s_seleccionado = random.choice(complementos_dos)

print("letra generada: " + sujeto_seleccionado + " " + intencion_seleccionada + " " + verbo_seleccionado + " "
      + adv_seleccionado + " " + compl1s_seleccionado + " " + compl2s_seleccionado) #se imprime la canción
```

letra generada: princess voy a jugal fuerte hasta el amanecer sin miedo



Repl.it I

Es una página web que permite ejecutar programas en varios lenguajes incluido Python. Para utilizarla se ingresará a <https://repl.it/> y se selecciona start coding. Posteriormente se crea una cuenta o se autentica con una cuenta de Gmail, Facebook o Github:

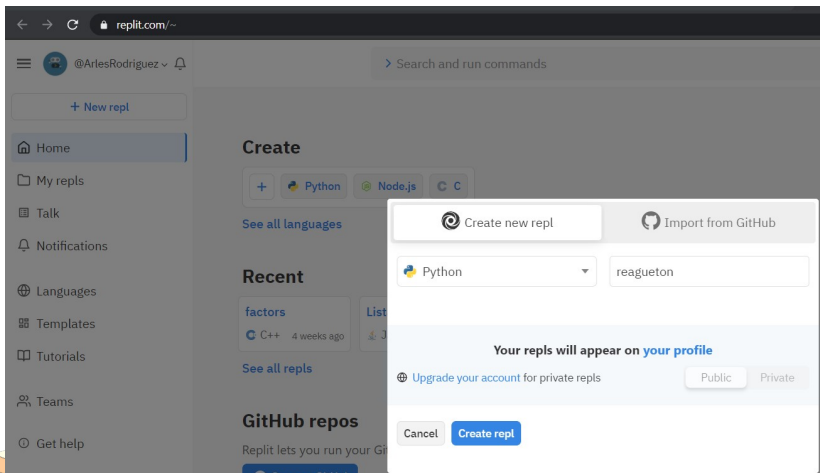
Code and collaborate, without friction.

Use our free, collaborative, in-browser IDE to code in 50+ languages — without spending a second on setup.

[<> Start coding](#)[Sign up](#)

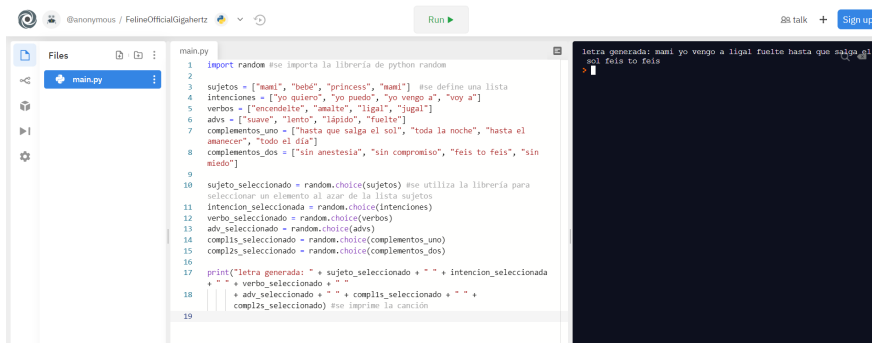
Repl.it II

Se selecciona +New repl y el lenguaje de programación Python:



Repl.it III

Se puede copiar y pegar el código anterior en el editor, se selecciona la opción run en la parte superior y se observa como se ejecuta en la consola de la parte derecha:



```
1 import random #se importa la librería de python random
2
3 sujetos = ["mami", "bebé", "princess", "mami"] #se define una lista
4 intenciones = ["yo quiero", "yo puedo", "yo vengo a", "voy a"]
5 verbos = ["encendele", "mañale", "ligal", "jugal"]
6 advs = ["suave", "lento", "lápido", "fuerte"]
7 complementos_uno = ["hasta que salga el sol", "toda la noche", "hasta el
8 amanecer", "todo el día"]
9 complementos_dos = ["sin anestesia", "sin compromiso", "feis to feis", "sin
10 miedo"]
11
12 sujeto_seleccionado = random.choice(sujetos) #se utiliza la librería para
13 seleccionar un elemento al azar de la lista sujetos
14 intencion_seleccionada = random.choice(intenciones)
15 verbo_seleccionado = random.choice(verbos)
16 adv_seleccionado = random.choice(advs)
17 comp1s_seleccionado = random.choice(complementos_uno)
18 comp2s_seleccionado = random.choice(complementos_dos)
19
20 print("letra generada: " + sujeto_seleccionado + " " + intencion_seleccionada
21 + " " + verbo_seleccionado + " "
22 + adv_seleccionado + " " + comp1s_seleccionado + " " +
23 comp2s_seleccionado) #se imprime la canción
```

letra generada: mami yo vengo a ligal fuerte hasta que salga el
sol feis to feis



Problema

Problema

Con base en el anterior programa elabore un programa que genere un discurso político al azar en una celda de Colab y en Repl:

Cómo hacer un discurso político

LAMBETAZO	POTENCIALES MARRANDOS	CONDICIÓN	COMPROMISO	ILUSIÓN GUERRERISTA	PROMESA	BENEFICIO POPULISTA	DEPENDIENDO DE LA CANTIDAD DE VOTOS
QUERIDOS	COMPAÑEROS	EN MI GOBIERNO	VOY A DERROTAR	LA VIOLENCIA Y	TRABAJARÉ POR	LA EDUCACIÓN	DEL PAÍS
APRECIADOS	CONCUDADANOS	CON SU APOYO	VENCERÉ	LA DELINCUENCIA Y	GARANTIZARÉ	EL EMPLEO	DE LA CIUDAD
DISTINGUIDOS	AMIGOS	SIENDO ELEGIDO	ELIMINARÉ	LA CORRUPCIÓN Y	PROTEGERÉ	LA SEGURIDAD	DE LA COMUNIDAD
HONORABLES	COTERRANEOS	CON SU AYUDA	ACABARÉ	LA INFLACIÓN Y	VELARÉ POR	LA PAZ	DE LA POBLACIÓN
ESTIMADOS	COPARTIDARIOS	SI ME SIGUEN	LUCHARÉ CONTRA	LA POBREZA Y	PROMOVERÉ	LA IGUALDAD	PARA TODA LA GENTE
RESPECTADOS	ELECTORES	DURANTE MI MANDATO	COMBATIRÉ	EL DESPLAZAMIENTO Y	DEFENDERÉ	LA SALUD	DE CADA COLOMBIANO

Referencias

- Armendáriz, D (2019). Introducción a Jupyter Notebook. Visitado el 2 de Junio de 2020. En:
<https://www.youtube.com/watch?v=orr063XGPPE>
- Barry, P. (2016). Head First Python: A Brain-Friendly Guide. "O'Reilly Media, Inc."
- Guttag, John. Introduction to Computation and Programming Using Python: With Application to Understanding Data Second Edition. MIT Press, 2016. ISBN: 9780262529624.
- Tutorial oficial de python 3, disponible en:
<https://docs.python.org/3/tutorial/interpreter.html>
- Rodríguez, A (2020). Curso de Programación en Python.
<https://github.com/arleserp/cursopython>

