Conociendo Python (Parte II)

Clase sincrónica

{desafío} latam_

Utiliza herramientas, comandos y estructuras básicas de Python para la creación de programas sencillos.

- Unidad 1: Conociendo Python
  (Parte I)

  (Parte II)
- Unidad 2: Tipos y estructura de datos
  (Parte I)

  (Parte II)
- Unidad 3: Manipulación y transformación de datos
  (Parte I)

  (Parte II)

  (Parte III)

Te encuentras aquí

{desafío} latam_

¿Qué aprenderás en esta sesión?

- *Crear programas utilizando estructuras de control de flujo.*

{desafío}
latam_

¿Qué tipos de datos podemos utilizar en Python?

¿Cuáles son las principales funciones que hemos visto?

{desafío} latam_

/*Algoritmos*/

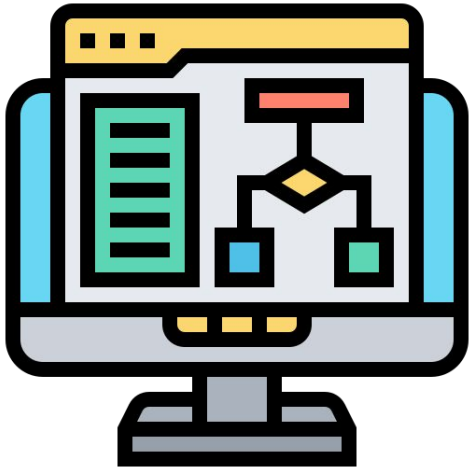
Algoritmo

¿Qué es un algoritmo?

Se llama **algoritmo** a una serie de pasos **ordenados** y **bien definidos** que se utilizan para resolver un problema específico.

Es como una receta que sigue un cocinero para preparar un plato de comida, pero en lugar de cocina, se utiliza en programación para resolver problemas.

- Debe entregar, efectivamente, una solución o respuesta.
- Debe ser finito, es decir, terminar (esto no es trivial).
- Debe ser preciso, es decir, no prestarse a interpretaciones.



Ejercicio - ¿Cómo se hace un huevo frito?

{desafío}
latam_

Ejercicio

¿Cómo se hace un huevo frito?

Para preparar un huevo frito, podemos aplicar los siguientes pasos.

- Encender un quemador de la cocina
- Poner una cucharadita de aceite en un sartén
- Poner el sartén sobre el quemador
- Luego de 30 segundos, quebrar un huevo sin romper la yema, ponerlo en el sartén sin cáscara y agregar una punta de cuchillo de sal.
- Cuando toda la clara del huevo esté blanca, apagar el quemador.
- Retirar el sartén y servir.

/*Diagramas de flujo*/

Diagramas de flujo

Representando un algoritmo

Un **diagrama de flujo** es una representación visual de un algoritmo, incluyendo diferentes casos y situaciones. Por ejemplo:

- Para elegir un día aleatorio de la semana, primero necesitamos importar el módulo '**random**'.
- Luego, creamos una lista con los días de la semana y usamos el método '**choice**' para seleccionar uno al azar.
- Finalmente, lo mostramos por pantalla.

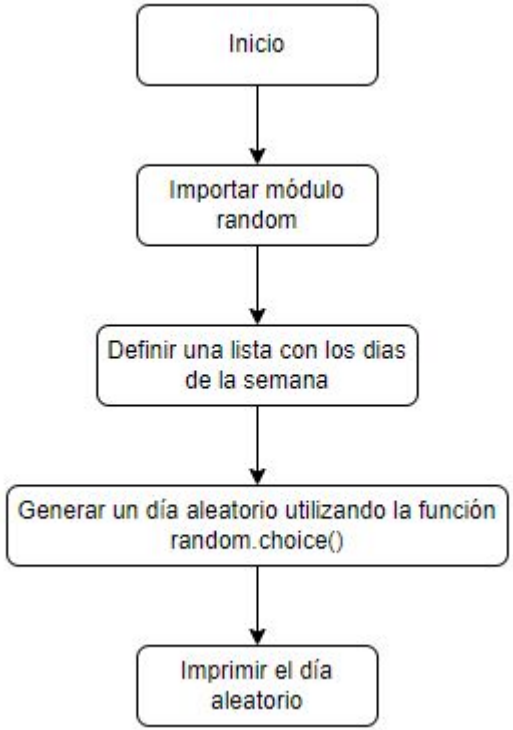


Diagrama de flujo

Representando un algoritmo

```
import random

dias = ["Lunes", "Martes", "Miércoles", "Jueves", "Viernes", "Sábado", "Domingo"]
dia_aleatorio = random.choice(dias)
print(dia_aleatorio)
```

```
Martes
```

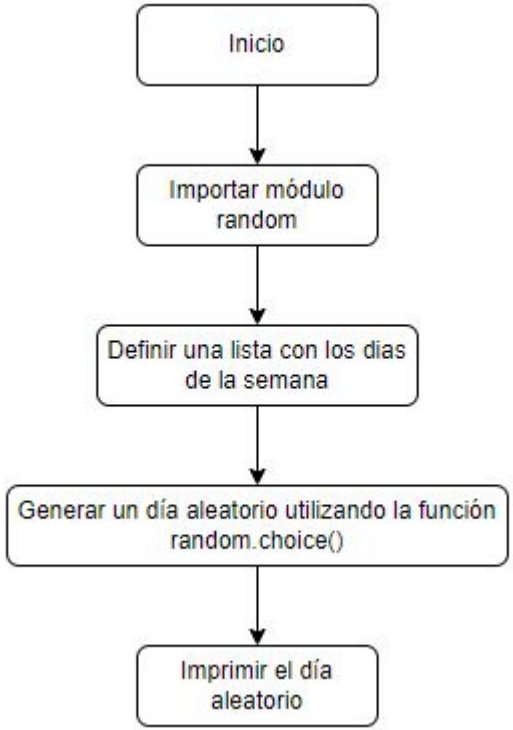


Diagrama de flujo

Tomando decisiones

Si es lunes, sacar la basura

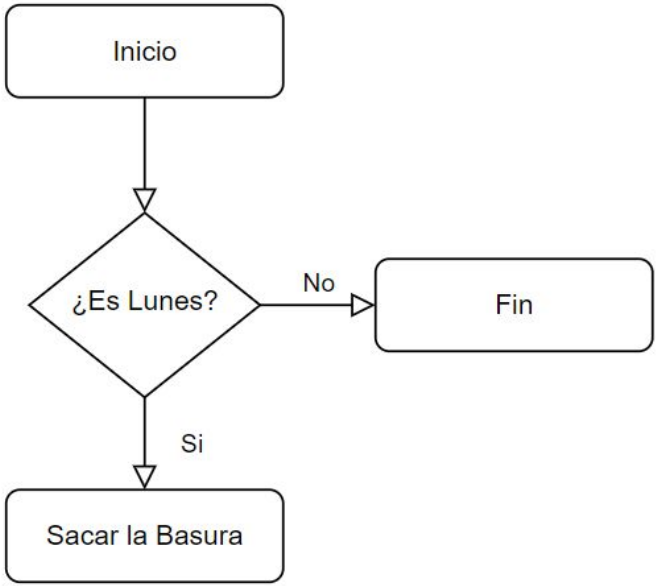
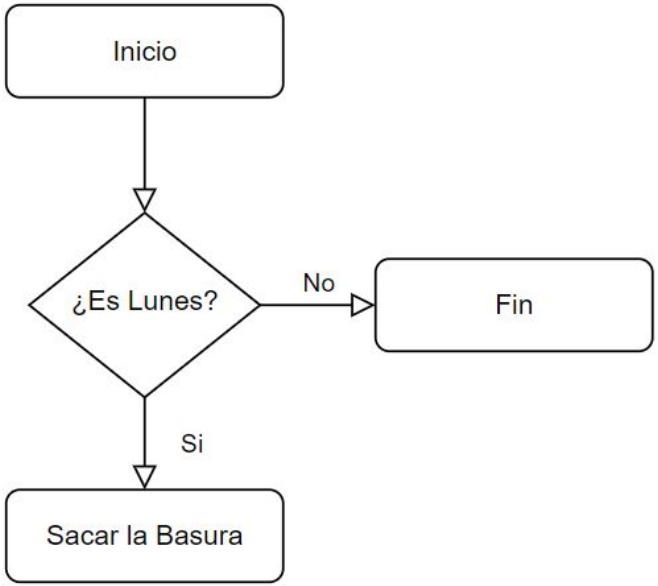


Diagrama de flujo

Tomando decisiones



- **Rectángulos:** se utilizan para representar procesos o actividades.
- **Rombos:** se utilizan para representar decisiones, es decir, puntos en el proceso en los que se debe tomar una decisión basada en cierta condición.
- **Flechas:** se utilizan para conectar las diferentes figuras geométricas y mostrar la dirección del flujo del proceso.

Diagramas de flujo ¡utiliza Draw.io!

{desafío} latam_

Diagramas de flujo

Utilizando Draw.io

Crea un diagrama de flujo que te ayude a elegir qué ropa usarás mañana dependiendo de las condiciones climáticas. Utiliza decisiones booleanas y los símbolos necesarios para representar el proceso de toma de decisión. Una vez que hayas terminado el diagrama, guárdalo como imagen o PDF para compartirlo o imprimirlo.

1. Abrir **Draw.io** desde un navegador web.
2. Seleccionar el tipo de diagrama "Flujo de proceso" desde el menú desplegable.
3. En la barra de herramientas lateral, seleccionar los símbolos necesarios para crear el diagrama de flujo.

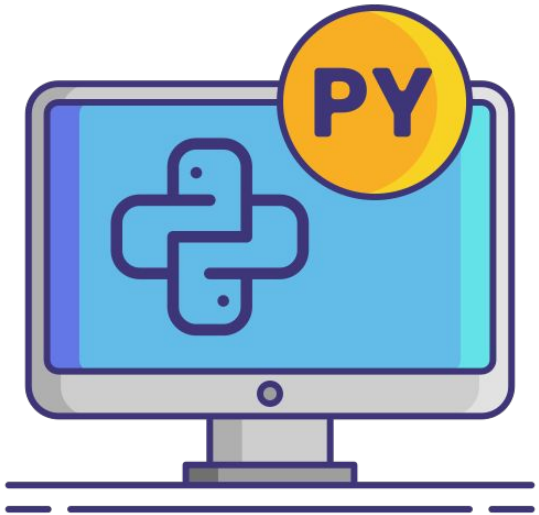
/*Operadores*/

Operadores en Python

Los **operadores** en Python son símbolos especiales que se utilizan para realizar operaciones matemáticas, comparaciones, asignaciones, y otras operaciones en el código.

Podemos distinguir:

- Operadores de Asignación y Comparación
- Operadores lógicos



Operadores de asignación y comparación

```
a=5
b=10

print(a==b)
print(a!=b)
print(a>b)
print(b>=a)
print(b>=7>a)
```

```
False
True
False
True
True
```

Operadores lógicos

and

```
print("True and True:",True and True)
print("True and False:",True and False)
print("False and False:",False and False)
```

```
True and True: True
True and False: False
False and False: False
```

Operadores lógicos

or

```
print("True or True:",True or True)
print("True or False:",True or False)
print("False or False:",False or False)
```

```
True or True: True
True or False: True
False or False: False
```

Operadores lógicos

not

```
print("not True:", not True)
print("not False:", not False)
```

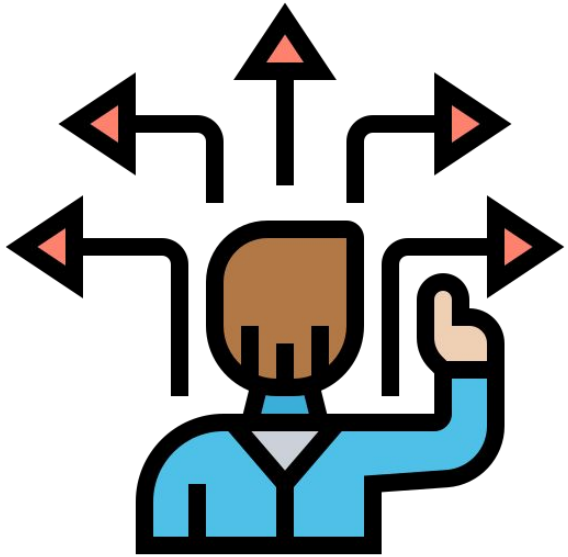
```
not True: False
not False: True
```

/*Estructuras de control*/

Estructuras de control

if - else - elif

Los **comandos** "if - else - elif" en Python permiten tomar decisiones basadas en una o varias condiciones, lo que permite que el programa pueda actuar de manera diferente según el caso que se presente.



{desafío}
latam_

¿Qué significan estas palabras, en inglés?

Solución

- if: si (condicional)
- else: si no...
- elif (else -if): si no, si...

Estructuras de control

Indentación

```
p = 7
if p > 5:
    print("azulejo")
    print(p+1)
```

```
azulejo
8
```

Estructuras de control ¡practiquemos!

{desafío} latam_

Estructuras de control

¡Practiquemos!

Veremos directamente, en Jupyter Notebook, el uso de las estructuras de control. Para ello, descarga y ejecuta el archivo 01. Ejemplo en clase 1 - estructuras.ipynb. En él, veremos el uso de las estructuras:

1. if
2. else
3. elif

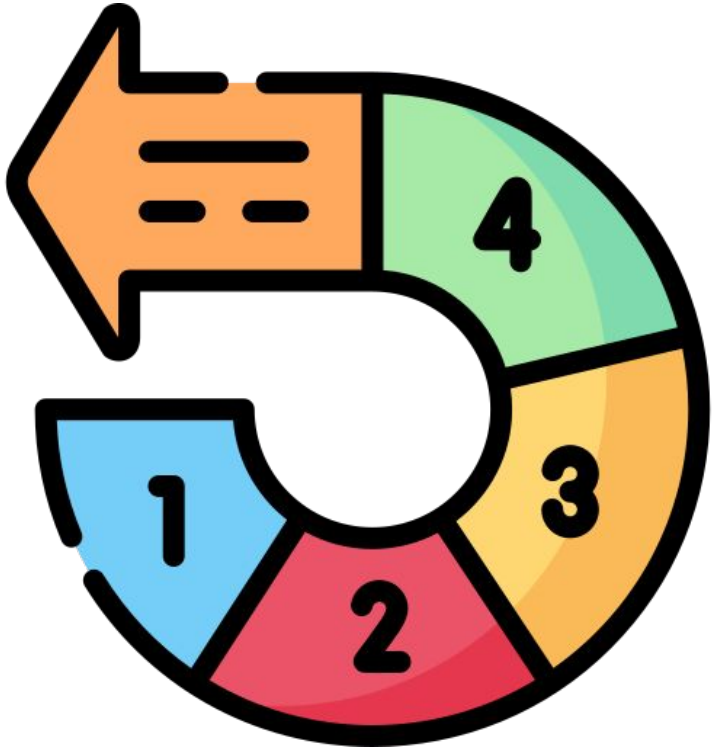
/*Ciclos*/

Ciclos

Repetición de procesos

Los **ciclos** en programación son formas de repetir una tarea varias veces, sin hacerlo manualmente.

- for
- while



¿Qué significan estas palabras, en inglés?

Solución

- for: para...
- while: mientras...

Ciclos

Estructura

```
lista = ["a",3,"t","perro"]
for i in lista:
    print(i)
```

```
a
3
t
perro
```

```
p = 0

while p <= 3:
    print(p)
    p = p + 1
```

```
0
1
2
3
```

Ciclos
¡practiquemos!

{desafío}
latam_

Ciclos

¡Practiquemos!

Veremos directamente, en Jupyter Notebook, el uso de los ciclos. Para ello, descarga y ejecuta el archivo 01. Ejemplo en clase 1 - ciclos.ipynb Abordaremos:

1. ciclo for
2. ciclo while
3. ciclos anidados

{desafío} latam_

Desafío -
Conociendo Python (Parte II)

{desafío}
latam_

Desafío

"Conociendo Python (Parte II)"

- Descarga el archivo "Desafío".
- Tiempo de desarrollo asincrónico: desde 4 horas.
- Tipo de desafío: individual.

¡AHORA TE TOCA A TI! 💪

{desafío} latam_

Ideas fuerza

Un algoritmo *es una* ***secuencia ordenada y finita de pasos*** *que entregan una* ***respuesta o solución.*** *Se puede representar mediante un* ***diagrama de flujo.***

Utilizamos ***operadores*** *de diferente tipo para asignar, realizar cálculos, verificar y/o comparar variables.*

Los ***ciclos*** *nos permiten realizar procesos repetitivos. En Python utilizamos* ***for*** *y* ***while****.*

¿En qué tareas puede ser importante considerar condicionales y ciclos?

Recursos asincrónicos

¡No olvides revisarlos!

Para esta semana deberás revisar:

- Guía de estudio
- Desafío “Conociendo Python (Parte II)”

Próxima sesión...

- *Utilizar y operar datos con diferentes estructuras.*
- *Cargar diferentes tipos de datos.*

{desafío}
latam_

{desafío}
latam_

Academia de
talentos digitales