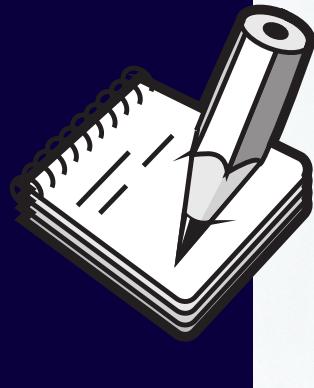


ANOTACIONES

PYTHON



La sintaxis de Python es sencilla

```
variable = 1
nombre = "nombre"
dicc = {}

if nombre != "Jhon":
    print("No eres Jhon")

while variable <= 10:
    print(variable)
    variable += 1
```

- Es fácil definir variables que tomen un tipo de dato
- la escritura es clara ya que no necesita de paréntesis ni llaves para definir bloques de código, ni son necesario los punto y coma (;) al final. Solamente escribimos dos puntos (:) después de una instrucción del lenguaje
- los nuevos bloques de código son reconocibles, pues llevan una sangría de cuatro espacios

Algunos tipos de datos en python son:



Mutables

list: [int, bool, str, list] **matriz:** [[], []]

set: {str, int, bool, tuple} *

Elementos únicos y sin orden

dict: {'key', 'value'}, * Elementos indexados por claves que siempre deben de ser un tipo de dato inmutable. Generalmente se usan int y str para claves

Inmutables

str: "Cadenas de texto"

float: 34.5232323

int: 23

boolean: True o False

tuple: 1, 2, True (1, 2, True) * No se pueden modificar después de ser creadas

Podemos convertir de un tipo de dato a otro usando su constructor, siempre y cuando el tipo a convertir sea compatible

tupla = 'a', 'b', 'c'

lista = list(tupla)

cadena = "1234"

numeros = int(cadena)

```
edad = "25" # cadena

if int(edad) >= 18:
    print("Es mayor de edad")

elif int(edad) > 100:
    print("Nooo..., te pasaste :"))

else:
    print("Es menor de edad")
```

condicionales

Estructuras que nos ayudan a evaluar objetos dando como resultado verdadero o falso (True or False)



también operadores lógicos que nos ayudan a tomar esas decisiones

or		<		and
not		==		in
is		!=		>
<=		>=		

```
if <condicion>:  
    "hará algo si  
    condicion es True"  
elif <condicion>:  
    "Si la anterior no  
    es True y esta si  
    lo es, ingresará aquí"  
else:  
    "Se ejecuta en caso  
    de que ninguna  
    condición se cumpla"
```

```
while <condicion>  
    "Mientras la condición  
    sea verdadera, no se  
    detendrá"  
    !! Cuidado con bucles  
    infinitos !!
```

Algunos métodos de cadenas y Secuencias....

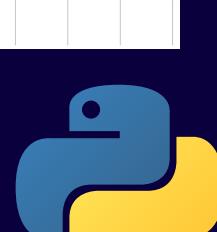
String (Cadena)

Algunos métodos

```
a = "Hola mundo"  
b = a.replace('la', 'ja') - Reemplaza el primer parámetro pasado por  
el segundo, devuelve una cadena nueva.  
a.find("a")] - Primera aparición en la cadena, o -1 si no lo  
encuentra.  
a.upper() - Convertir a mayúscula.  
a.lower() - Convertir a minúscula.  
a.strip() - Eliminar espacios al inicio y al final.  
a.split(" ") - Separar una cadena en base a un carácter que  
tengamos dentro de la cadena.  
a.isalpha() - Revisa si los caracteres son alfabéticos.  
a.isdigit() - Revisa si los caracteres son numéricos.
```

También podemos ingresar a los elementos de una cadena por su índice:

a[1]
a[:5] - Slice
a[:] - Vacío es ir desde el inicio al final



Secuenciales

Tipos de datos que nos permiten guardar distintos datos en sucesión.

Pueden ser recorridas por medio del ciclo **for**

```
for var in tuple:  
    pass
```

```
for var in lista:  
    pass
```

Tuplas

Algunos métodos

```
objeto_tupla = "monitor", "teclado", "torre"  
var_1, var_2, var_3 = objeto_tupla - Desempaquetar una tupla, la cantidad de variables debe ser idéntico a la cantidad de datos
```

```
objeto_tupla [:3] - Slice
```

```
objeto_tupla [::2] - Slice por pasos
```

```
objeto_tupla [::-1] - Le damos vuelta
```

```
objeto_tupla .count("monitor") - Contamos cantidad de veces que aparece el valor que le pasamos
```

listas

Algunos métodos

```
computador = "monitor,teclado,raton"  
armar_computador = computador.split(",") - Crear lista con método.
```

```
armar_computador.append('torre') - Agregar a la lista
```

```
armar_sala_pc = ["silla", "escritorio"]
```

```
armar_sala_pc += armar_computador - Unimos listas
```

```
armar_sala_pc * 2 - Replicamos la misma lista
```

```
armar_sala_pc[-1] - Indexar listas
```

```
armar_sala_pc[1:4]
```

```
armar_sala_pc[0] = "Silla escritorio" - Cambiar un elemento
```

```
armar_sala_pc.insert(0, "Silla gamer") - Agregar elemento en un índice, el elemento que estaba antes no se borra si no que hace espacio
```

```
armar_sala_pc.remove("Silla escritorio") - Eliminamos primer encuentro de un elemento
```

```
abcdario = ["w","e","r","a","d","t","u"]
```

```
abcdario.sort() - Ordenamos lista
```

```
numeros = [4,5,2,9,0,34,5,7,21,54]
```

```
numeros_ordenados = sorted(numeros) - Ordenamos los elementos pero mantenemos intacta la lista original donde están
```

```
numeros.pop(3) - Elimina un elemento en un índice u lo retorna como valor
```



Diccionarios

Algunos métodos

diccionario = {} - Creamos diccionario vacío

```
persona = {  
    "nombre": "Eren",  
    "apellido": "Kruger",  
    "ciudad": "Paradise",  
    "lenguajes_programacion": ["Python", "Java", "C#", "PHP"]  
}
```

persona["nombre"] Indexamos por medio de las claves del diccionario

persona['apellido'] = "Jaeger" – Cambiamos un valor

persona['lenguajes_programacion'].append("JavaScript") – Agregamos un valor, lo hacemos así aquí porque el valor que guarda esta llave es de tipo lista

del persona["fecha_nacimiento"] – Eliminamos una llave

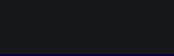
a = persona.pop("nombre") – Eliminamos un elemento, pero nos quedamos con su valor

t = persona.popitem() – Elimina último elemento par clave-valor y lo devuelve como tupla

persona.clear() Eliminar todos los elementos



```
for llave, valor in dicc.items():  
    print(llave, valor)
```



```
for llave in dicc.keys():  
    print(llave)
```



```
for valor in dicc.values():  
    print(valor)
```

Métodos para
acceder a
propiedades del
diccionario

Podemos recorrer con un **for**
tradicional y esto nos dará solo
llaves

