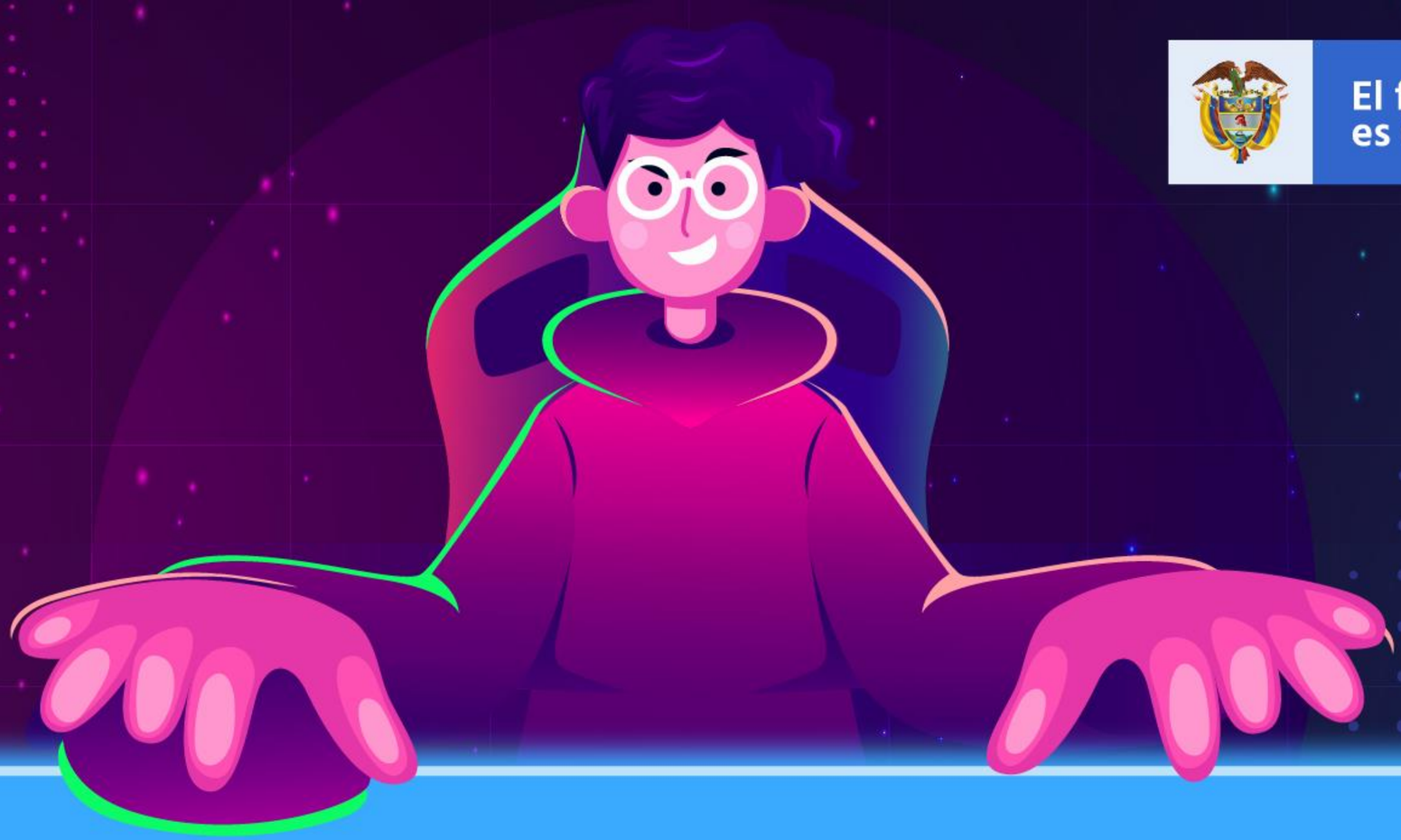


El futuro digital es de todos

MinTIC

Programación con Python

OPERADO POR:

Universidad Tecnológica de Pereira

Mision TIC 2022

RUTA DE APRENDIZAJE 1

Conceptos básicos en Phyton

En cualquier lenguaje de programación es necesario aprender a manejar los datos de nuestro problema, para usarlos de forma mas eficiente al comunicarnos con una computadora.

Python nos ayuda bastante a hacer este tipo de análisis, sin que la programación sea muy complicada para nosotros, especialmente por la forma en la que su lenguaje trabaja.

Python es un lenguaje que podemos considerar secuencial, un tipo de lenguaje que permite ser manejado de forma mas simple que otros lenguajes que son orientados a objetos.

En cases pasadas revisamos los conceptos de variables, parámetros o contantes y conjuntos, esto será especialmente importante para comenzar a trabajar con Python.

PYTHON

Variables de Python

Variable es un nombre que se usa en este caso para referirse a la ubicación de la memoria. La variable de Python también se conoce como un identificador y se usa para mantener el valor.

En Python, no necesitamos especificar el tipo de variable porque Python es un lenguaje de inferir y es lo suficientemente inteligente como para obtener el tipo de variable.

Los nombres de las variables pueden ser un grupo de letras y números

- Deben comenzar con una letra o un guion bajo.
- Si comienza con numero generara un error, sin embargo la variable si puede llevar números, solo no tiene permitido que se encuentre en la primera posición.
- Python distingue entre mayúsculas y miniscular, por lo tanto una variable de nombre: var1 es diferente a VAR1.

Declaración de variable y asignación de valores

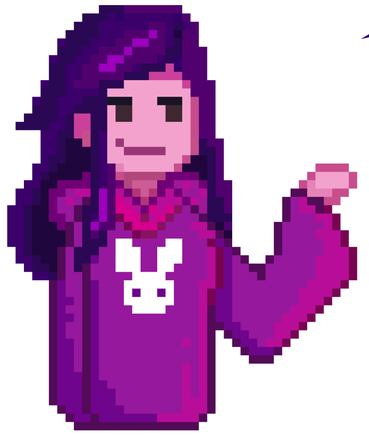
Python no nos obliga a declarar una variable antes de usarla en la aplicación. Nos permite crear una variable en el momento requerido.

No necesitamos declarar explícitamente variables en Python. Cuando asignamos cualquier valor a la variable, esa variable se declara automáticamente.

El operador igual (=) se utiliza para asignar valor a una variable.

En Python, las variables son un nombre simbólico que es una referencia o puntero a un objeto.

Las variables se utilizan para denotar objetos con ese nombre.



Vamos a programar usando un cuaderno de notas en **Jupiter** para tener todos nuestros apuntes en un solo lugar y entender mejor los conceptos.

En una celda de tipo "code" vamos a escribir el nombre de la variable, Var1 que queremos crear y con un = vamos a definir cuanto vale.

Vamos a usar un comando que ya conocemos, con print() vamos a pedirle al código que imprima el valor que le dimos a la variable

```
In [ ]:
var1 = 1.20
print(var1)
```

Si ejecutamos la celda, con la opción “Run” el código lee la variable var1, le da el valor de 1.20 y luego imprime el valor.

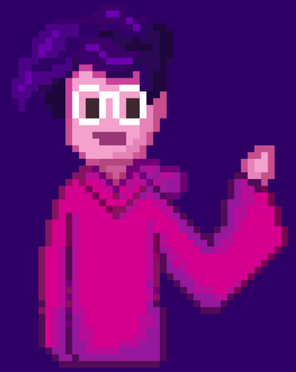
```
In [1]: var1 = 1.20
        print(var1)

        1.2
```

En Phyton podemos definir la variable según sus características, **si es entera, continua o binaria**. Para la **Var1** su valor es continuo, por lo tanto en programación vamos a ver esta caratersitica con el nombre **Float**.

```
In [ ]:  var1=int(var1)
         print(var1)
```

Ahora, vamos a tomar la variable **Var1** y vamos a cambiar su característica continua.

Para eso vamos a escribir **Var1 = int (Var1)**



Parece que estamos dando vueltas, pero realmente estamos **redefiniendo** la variable.

Var1 = int (Var1) -→ queremos decirle al computador:

Tome la variable **Var1** y conviértala en una variable entera y guarde ese resultado en el mismo nombre.

Así podemos cambiar su característica, sin necesidad de crear una variable nueva.

```
In [1]:
var1 = 1.20
print(var1)
```

```
1.2
```

El valor de la variable original, es 1.2, cuando cambiamos su característica a entero, el solo me imprime la parte **entera** del numero.

```
In [2]:
var1=int(var1)
print(var1)
```

```
1
```

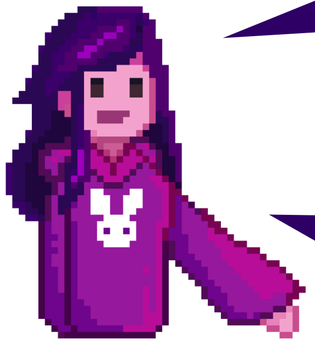
Como se puede ver, ahora aparece un [1] y un [2] en las dos celdas tipo **"Code"**. En Jupiter esto me indica cual celda se ejecuto primero.

Podemos entonces crear una variable con su valor en una celda, y usarla en otras celdas tipo **"code"** mas abajo. Lo importante es que ejecutemos la celda [1] para que **Jupiter** guarde el valor de la variable y podamos usarla después.

En Python podemos hacer que una variable nueva defina el valor de una variable que creamos antes, o podemos definir dos valores diferentes para cada variable. Eso depende del problema que estamos resolviendo.

```
In [3]: var1 = 120

        var2 = var1

        print(var2)
```

```
120
```



La variable var1, ya tenia un valor de 1.2, ahora vamos a cambiar ese valor a 120 y asignar ese mismo numero a una variable nueva var2.

En [3] podemos ver que el código hizo lo que queríamos, cuando imprimió el valor de la variable var2 = 120, que solo dependía del valor de var1.

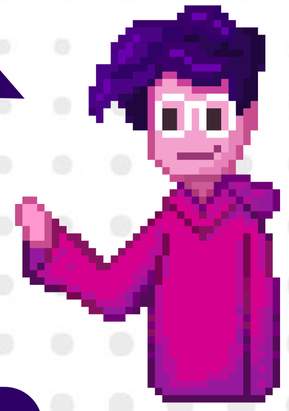
También podemos definir las variables que entregan cadenas de palabras algo que en programación llamamos una cadena de string.

En Python no es necesario definir una cadena de palabras de una forma complicada o extraña, solo es necesario que esta se encuentre entre comillas “”.

```
Variable = "Hola mundo"

print (Variable)
```

```
Hola mundo
```

Luego de definirla, podemos **imprimir** la variable de la misma forma en casos anteriores.



Esto es usado en muchos casos para verificar si un código esta funcionando correctamente, donde el código entregaría una frase especifica, si logra ejecutarse correctamente.

```
In [4]:
var1 = 120
var2 = 200

print(var1)
print(var2)
```

```
120
200
```

También podemos hacer una prueba, creando dos variables con valores diferentes. Si le pedimos al código que imprima el valor de cada una, nos entregara valores diferentes.

Como en cualquier lenguaje de programación no estamos exentos de cometer **errores de sintaxis**.

Miremos algunos de los errores mas comunes al definir variables:

```
In [ ]:
var 1 = 120
```

```
In [ ]:
@var1 = 120
```

```
In [ ]:
1var = 120
```

```
In [ ]:
var-1 = 120
```

En programación es importante saber como funcionan los comando del lenguaje que estamos utilizando. Sin embargo es igual de importante aprender a comprender que errores estamos cometiendo, y como Python nos lo comunica:

```
In [1]: var 1 = 120
  File "<ipython-input-1-e1cd4715cd7d>", line 1
    var 1 = 120
        ^
SyntaxError: invalid syntax
```

En este ejemplo, al definir la variable, dejamos un espacio en el nombre de la variable, eso no esta permitido. Por eso Python nos indica que tenemos un **error de sintaxis**, nos dice en cual **línea** de nuestro código y en que **posición**.

Python no nos dirá exactamente cual es el error, que escribimos mal, o que hicimos mal, solo nos dirá que tipo de error es, y donde esta ubicado.

En este caso, el nombre de la variable no puede tener espacios, por lo tanto muestra con una flecha indicando que el error se encuentra cerca a esa posición y genera un error de sintaxis: No se están siguiendo las reglas del juego.

En este caso, el error también es de sintaxis, no podemos usar símbolos especiales. Sin embargo aquí Python no nos muestra exactamente donde fue el error, solo nos dice que es en esa línea.

```
@var1 = 120
```

```
  File "<ipython-input-2-db7874e6de9e>", line 1
    @var1 = 120
          ^
SyntaxError: invalid syntax
```

- Para este caso, tenemos un **valor numérico** como símbolo inicial de la variable, lo cual **no esta permitido**.

- Otro error es usar **caracteres o símbolos que representen operaciones matemáticas**, el guion **–** representa la operación **resta**, por lo tanto no puede hacer parte del nombre de una variable. En este caso Python lo dice mas claramente en comparación de casos anteriores.

```
In [3]: 1var = 120
  File "<ipython-input-3-157c48da2623>", line 1
    1var = 120
     ^
SyntaxError: invalid syntax
```

```
In [4]: var-1 = 120
  File "<ipython-input-4-4e7cad843eaa>", line 1
    var-1 = 120
        ^
SyntaxError: cannot assign to operator
```

Traduciendo: No se puede asignar a un operador.

Asignación múltiple: Python nos permite asignar un valor a múltiples variables en una sola declaración, lo que también se conoce como asignaciones múltiples.

Podemos aplicar múltiples asignaciones de dos maneras, ya sea asignando un solo valor a múltiples variables o asignando múltiples valores a múltiples variables.

```
In [ ]: var1 = var2 = var3 = 200
```

```
In [ ]: var1, var2, var3 = 10 , 20 , 30
```

Podemos darles el mismo valor, o darles valores diferentes, solo separándolas con una coma.

Manipular números:

Operadores Aritméticos:

Tal como cualquier otro lenguaje de programación, los operadores de sumas, restas, multiplicaciones y divisiones pueden ser usadas con números.

Con los números se pueden realizar los siguientes tipos de operaciones:

Operación	Resultado
+	Suma
-	Resta
*	Multiplicación
/	División
//	División entera
**	Potencia

```
In [ ]:  2 + 3    # Suma
```

```
In [ ]:  #esta suma retornada un float

numero1 = 10
numero2 = 9.99
numero3 = numero1 + numero2
print(numero3)
type(numero3)
```

Podemos realizar una suma, directamente escribiendo los números, como en una calculadora.

También podemos declarar 2 variables con un valor numérico y una que sea igual a la operación, como se muestra en la imagen.

Si usamos el comando type, podemos pedirle al programa que nos diga si el resultado la la variable numero 3 es entera o continua.

Están son todas las operaciones que podemos realizar con Python y sus respectivos símbolos dentro del lenguaje.

Dependiendo del problema que estemos solucionando, vamos a necesitar hacer cálculos cada vez mas “complicados” por eso es tan importante recordar estas operaciones, para que nuestro trabajo sea mucho mas fácil.

```
In [ ]: 8 - 5   # Resta

In [ ]: 2 * 6   # Multiplicacion

In [ ]: 12 / 3  # Division

In [ ]: 12 // 3 #Division entera

In [ ]: 7 % 3   # Modulo (retorna el remanente de la división)

In [ ]: cuadrado = 7 ** 2 #potencia al cuadrado

In [ ]: cubico = 2 ** 3 #potencia al cubo

In [ ]: x = 15 + 18

In [ ]: sqrt = x**(1/2) # raiz cuadrada
```

Orden en las operaciones: Cuando en una expresión aparece más de un operador, el orden de evaluación depende de las reglas de precedencia. Para los operadores matemáticos, Python sigue la convención matemática. El acrónimo **PEMDAS** es una forma útil de recordar las reglas:

Paréntesis (P)

Los paréntesis tienen la mayor precedencia y pueden ser usados para forzar una expresión a evaluar en el orden que se desee. Como las expresiones entre paréntesis se evalúan primero:

Ejemplo: 2 * (3-1) es igual a 4 , Primero se resuelve la operación entre paréntesis y después se hace la multiplicación.

```
Los paréntesis nos pueden ayudar a leer las ecuaciones mas fácilmente,
así su uso no haga una diferencia en como se hace el calculo.
```

Exponenciación (E): La exponenciación tiene la siguiente precedencia más alta.

Ejemplo:

- 2**1+1 es 3, no 4.
- 3*1**3 es 3, no 27.

Multiplicación (M), División (D), Suma (A), Resta (S): En ese mismo orden se debe realizar el calculo en la ecuación, si no tenemos paréntesis o potencias.

Ejemplo:

2*3-1 es 5, no 4

Misma precedencia:

Los operadores con la misma precedencia se evalúan de izquierda a derecha.

Ejemplo:

Así que la expresión 5-3-1 es 1, no 3, porque el 5-3 ocurre primero y luego el 1 es restado de 2.

Si lo programamos en Python, nos entrega el mismo análisis, con el resultado igual a 1.

```
In [6]: 5-3-1

Out[6]: 1
```

Los operadores pueden ser utilizado con cadenas de palabras o frases, no es lo mismo si estamos trabajando con números, Python se comportara de forma diferente según el tipo de datos que estemos utilizando:

Ejemplo: El signo + se comporta de forma diferente o entrega un resultado diferente cuando estamos trabajando con cadenas de texto. En este caso, se encarga de conectar cadenas de texto.

H	O	L	A		M	U	N	D	O

Primero necesitamos comprender como se almacenan las palabras.

Cuando trabajamos con Python, las cadenas de palabras ocupan un espacio por cada letra, incluyendo los espacios, esto muy importante a la hora de programar y lo veremos con mas detalle mas adelante.

Si tenemos números el código nos entregara la suma de los valores. Si tenemos cadenas de texto nos entrega la unión de varias cadenas.

Debemos recordar como se guardan las palabras en la memoria, si queremos un espacio entre la palabra hola y la palabra de, debemos colocarlo.

```
primer_numero =7

segundo_numero = 9

suma = primer_numero + segundo_numero

print (suma)

primera_cadena = "Hola "

segunda_cadena = "de nuevo"

frase = primera_cadena + segunda_cadena

print (frase)
```

```
16
Hola de nuevo
```

El futuro digital es de todos

MinTIC

GRACIAS

OPERADO POR:

Universidad Tecnológica de Pereira

MisiónTIC2022