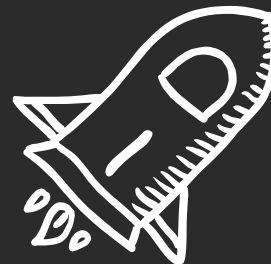




Semana 2

¡Bienvenidos!



Temas de hoy

1

Python: Primeros
pasos

2

PEP8

3

Integración de conceptos

1

Python: Primeros pasos

Luego de haber dado los primeros pasos en la programación, haber visto los conceptos básicos y haber trabajado sobre ellos, es momento de ¡jarrancar a programar con todo!

Desde hoy ya vamos a comenzar a trabajar con el lenguaje de programación elegido para este curso: PYTHON



¡Manos a la obra!



Python: el lenguaje más elegido 2024

Python a la vanguardia

Python se mantiene a la vanguardia en el panorama de la programación.

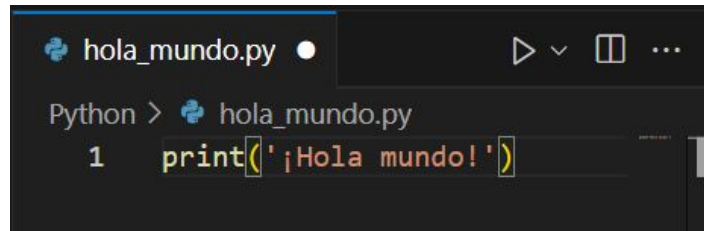
En el Ranking TIOBE de julio de 2024 (y en otros indicadores como PYPL Index o encuestas a desarrolladores), Python ocupa el puesto número 1 como el lenguaje de programación más popular del mundo. Esta posición la ha mantenido durante varios años, consolidando su liderazgo en la industria.

Python es de fácil uso, versátil, con una amplia comunidad, con una gran cantidad de librerías y frameworks y su documentación oficial siempre al día.

<https://docs.python.org/3/>

Primeros pasos

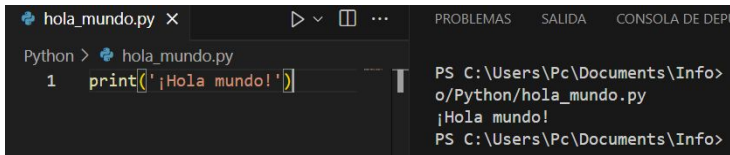
Así como en etapa 1, en esta etapa vamos a seguir trabajando con VSC como nuestro entorno de trabajo, por lo que (ya habiendo instalado Python, su extensión en VSC y teniendo nuestra carpeta donde trabajaremos) abriremos nuestro entorno y vamos a crear un archivo nuevo que extensión “py”, para hacer nuestro primer código en Python. Lo nombraremos “hola_mundo.py” y en él, escribiremos esta línea de código. `print('¡Hola mundo!')`



```
Python > hola_mundo.py
1 print('¡Hola mundo!')
```

Python: primeros pasos

Si ejecutamos esta línea de código (con el botón de ejecutar que es el ícono de “play”) en la consola de Python integrada en VSC, nos saldrá la leyenda “¡Hola mundo!” la cual hace referencia a la primer línea de código cuando se está comenzando a programar.



```
Python > hola_mundo.py
1 print('¡Hola mundo!')
```

```
PS C:\Users\Pc\Documents\Info>
o/Python/hola_mundo.py
¡Hola mundo!
PS C:\Users\Pc\Documents\Info>
```

Como te darás cuenta, la palabra reservada o comando que usamos para mostrar este texto por consola, es básicamente la traducción de “Imprimir” que usábamos en pseudocódigo con PSeint. Luego se siguen usando comillas para poner el texto que queremos. La única diferencia, es que el texto, aparte de usar comillas, está encerrado entre paréntesis.

Estos paréntesis son fundamentales ya que el comando “print” es una función, y las funciones siempre llevan

paréntesis abierto y cerrado. Al tema de *funciones* lo vamos a profundizar más adelante, pero te lo mencionamos ahora para que no se te olviden los paréntesis 😊.

Otra cosa para mencionar es que, las extensiones en Python siempre serán “.py” (que es propio de Python).

En Python, ¿es necesario definir las variables?

No, no es necesario. A diferencia de PSeint, donde teníamos que definir las variables en el pseudocódigo, en Python, al ser de tipado dinámico, no es necesario, simplemente se las declaran y se comienzan a usar.

La diferencia entre definir y declarar es que en la definición se debe escribir el nombre de la variable y el tipo de dato que almacenará, sin embargo, en Python, solo se debe poner el nombre de la variable y asignar el valor a usar, sin importar el tipo de dato. Python interpreta de manera automática el tipo de dato que recibe la variable (tipado dinámico).

Python: variables y tipos de datos

Si bien ya conociste los tipos de datos, y, en el material complementario este tema se profundiza, vamos a darte unos ejemplos para que comprendas mucho mejor.

```
variables_tipos_datos.py X
Python > variables_tipos_datos.py > ...
1 nombre = 'Informatario'
2 anio = 2024
3 etapa = 2
4 en_curso = True
5 variable_decimal = 2.15
6
7 print(nombre, anio, etapa, en_curso, variable_decimal)
```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL PUERTOS

```
PS C:\Users\Pc\Documents\Info> & C:/Users/Pc/AppData/Local/Programs/Python/Python311/Python.exe variables_tipos_datos.py
Informatario 2024 2 True 2.15
PS C:\Users\Pc\Documents\Info>
```

Para este ejemplo, usamos todos los tipos de datos más comunes (string, integer, bool, float), y luego, los mostramos por consola.

La forma de mostrar los valores de las variables por consola, es básicamente como lo hacíamos en pseudocódigo, donde escribíamos los nombres de las variables separados por coma, con la palabra reservada “Imprimir”, pero ahora usamos la palabra reservada “print” y sus paréntesis: **print()**.

Una función que nos brinda Python para ver el tipo de dato que tiene almacenado la variable, es la función **type()**, la cual, en conjunto con la función “print” nos muestra el tipo de dato.

```
variables_tipos_datos.py X
Python > variables_tipos_datos.py > ...
1 nombre = 'Informatario'
2 anio = 2024
3 etapa = 2
4 en_curso = True
5 variable_decimal = 2.15
6
7 print(type(nombre))
8 print(type(anio))
9 print(type(en_curso))
10 print(type(etapa))
11 print(type(variable_decimal))
```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN

```
PS C:\Users\Pc\Documents\Info> & C:/Users/Pc/AppData/Local/Programs/Python/Python311/Python.exe variables_tipos_datos.py
<class 'str'>
<class 'int'>
<class 'bool'>
<class 'int'>
<class 'float'>
PS C:\Users\Pc\Documents\Info>
```

Python: variables y tipos de datos

Como ves, cuando ejecutamos por consola, esta función “type()” en conjunto con “print()”, podemos ver cada tipo de dato que tiene almacenado la variable.

En este caso tenemos los tipos de datos fundamentales:

- **str** → String o Cadena de Caracteres
- **Int** → Integer o Entero (número entero)
- **bool** → Boolean o Booleano
- **float** → Float o Decimal (número decimal)

Otra función fundamental, es la que nos permite que el usuario ingrese un dato y lo podamos almacenar en una variable.

En PSeint, usábamos la palabra reservada “Leer” para almacenar un dato en la variable cuando el usuario ingresaba ese dato, pero en Python vamos a usar la palabra reservada “**input()**”. Esta función tiene la particularidad que permite mostrar por pantalla y a su

vez almacenar el dato que ingrese el usuario.

```
input.py > ...
1  nombre = input('Ingrese su nombre: ')
2
3  print(nombre)
4

PROBLEMAS  SALIDA  TERMINAL  ...

PS C:\Users\Pc\Documents\Info\Python> & C:/U
/Python312/python.exe c:/Users/Pc/Documents/
Ingrese su nombre: Informatorio
Informatorio
PS C:\Users\Pc\Documents\Info\Python> |
```

En el ejemplo, declaramos una variable y asignamos el valor usando la función “input()” para que de esta forma, el dato

Python: ingreso de datos y sus características

que se va a almacenar, sea ingresado por consola. La particularidad, como te mencionámos anteriormente, es que con “input” además, de poder leer y guardar el dato en la variable, también podemos mostrar por consola un texto. En la línea de código 3, te mostramos por medio de “print” que efectivamente, el valor ingresado, fue almacenado en la variable “nombre”.

Algo a tener muy en cuenta sobre la función “input()” es que los datos serán almacenados pero como tipo de dato “str” o string, por lo que, por más que yo ingrese un valor numérico o booleano, el dato almacenado, será tomado como “str”

```

input.py > ...
1  nombre = input('Ingrese su nombre: ')
2  numero = input('Ingrese un número: ')
3
4  print(type(nombre))
5  print(type(numero))
6
PS C:\Users\Pc\Documents> python s/Pc/AppData/Local/Programs/Python/Python.exe c:/Users/Pc/Documents/input.py
Ingrese su nombre: Info
Ingrese un número: 1234
<class 'str'>
<class 'str'>
```

Para solucionar esto, hay dos formas de poder hacerlo. Por un lado podemos tomar el valor de la variable y convertirlo al tipo de dato requerido:

```

input.py > ...
1  numero = input('Ingrese un número: ')
2
3  print(type(numero))
4
5  numero = int(numero)
6
7  print(type(numero))
PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAR

PS C:\Users\Pc\Documents\Info\Python> & C:/Users/Pc/
Ingrese un número: 1234
<class 'str'>
<class 'int'>
PS C:\Users\Pc\Documents\Info\Python>
```


Python: ingreso de datos y sus características

Como podés observar, Python nos permite tomar la variable con el valor tipo str y realizar una conversión a int, simplemente declarando una asignación a la misma variable, pero poniendo el tipo de dato al que requerimos se convierta.

Para este mismo caso, se podría utilizar otra variable y en esa variable asignar el valor pero convertido:

```
input.py > ...
1  numero = input('Ingrese un número: ')
2
3  conversion = int(numero)
4
5  print(type(conversion))

PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL

PS C:\Users\Pc\Documents\Info\Python> & C:/Users
Ingrese un número: 123
<class 'int'>
PS C:\Users\Pc\Documents\Info\Python> |
```

Otra cosa a tener en cuenta cuando usamos la función "input()", es la de manejar errores cuando el usuario ingresa mal un tipo de dato, ya que, si bien Python es de tipado dinámico, solo acepta de a un solo tipo de dato para las variables, es decir, no se puede poner letras y números y que reconozca que hay números y letras. En un caso así, Python va a reconocer todo como string:

```
input.py > ...
1  numero = input('Ingrese un número: ')
2  print(type(numero))
3

PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL

PS C:\Users\Pc\Documents\Info\Python> & C:/Us
Ingrese un número: Info 1234
<class 'str'>
```

Por eso, cuando hacemos una conversión a otro tipo de dato, tenemos que estar seguros que el tipo de dato al que vamos a convertir es el correcto. Por ejemplo:

Python: ingreso de datos y sus características

```
input.py > ...
1  numero = input('Ingrese un número: ')
2  print(type(numero))
3  numero = int(numero)

PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL  PUERTOS

PS C:\Users\Pc\Documents\Info\Python> & C:/Users/Pc/AppData/Local/Programs
Ingrese un número: Info 123
<class 'str'>
Traceback (most recent call last):
  File "c:\Users\Pc\Documents\Info\Python\input.py", line 3, in <module>
    numero = int(numero)
            ^^^^^^^^^^^
ValueError: invalid literal for int() with base 10: 'Info 123'
PS C:\Users\Pc\Documents\Info\Python> |
```

En este caso, estoy pidiendo a Python que me convierta el valor de la variable a entero, pero no puede hacerlo, ya que esta variable, tiene como valor letras y números. Si, solo hubiese tenido números (como en uno de los ejemplos anteriores), se le puede convertir a "int" sin problemas, pero como no puede coexistir más de un tipo de dato en una variable a mismo tiempo, nos da un error. Algo muy importante para aclarar, ya que seguramente

te estás preguntado, es ¿entonces el tipo de dato de una variable, no puede cambiar durante la ejecución del programa?, y la respuesta es que si puede cambiar de tipo de dato a lo largo del programa, pero más de un tipo de dato no puede existir como valor dentro de la variable al mismo tiempo. Por ejemplo:

```
input.py > ...
1  numero = input('Ingrese un número: ')
2  print(type(numero))
3  numero = int(numero)
4  print(type(numero))
5  numero = input('Ingrese una letra: ')
6  print(type(numero))

PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL

PS C:\Users\Pc\Documents\Info\Python> & C:/User
Ingrese un número: 098
<class 'str'>
<class 'int'>
Ingrese una letra: Info
<class 'str'>
PS C:\Users\Pc\Documents\Info\Python> |
```

Python: ingreso de datos y sus características

Entonces, podemos entender que los tipos de datos que se almacenen por defecto con la función “input()” van a ser de tipo “str”, sin embargo, este tipo de dato puede ser convertido y, si se vuelve a usar “input()” nuevamente, a lo largo de la ejecución del programa con la misma variable, esta va a volver a adoptar el tipo de dato “str”.

Para solucionar un posible error y que no se “rompa” el código (como vimos en un ejemplo anterior), en el caso que necesitemos que el usuario ingrese solo números, podemos usar los comandos “try - except”, y, en ese caso también nos vamos a ahorrar la línea de código de conversión, ya que no será necesario convertir al tipo de dato requerido luego de su ingreso, sino que, vamos a poder pedir el dato y hacer la conversión en un solo paso.

```
input.py > ...
1  try:
2      numero = int(input('Ingrese un número entero: '))
3      print('Muchas gracias por ingresar un número entero.')
4  except ValueError:
5      print('El dato ingresado no es un número entero. Intente nuevamente.')
6

PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL  PUERTOS

PS C:\Users\Pc\Documents\Info\Python> & C:/Users/Pc/AppData/Local/Programs/Python/Py
Ingrese un número entero: Info
El dato ingresado no es un número entero. Intente nuevamente.
PS C:\Users\Pc\Documents\Info\Python> & C:/Users/Pc/AppData/Local/Programs/Python/Py
Ingrese un número entero: 123
Muchas gracias por ingresar un número entero.
PS C:\Users\Pc\Documents\Info\Python> |
```

Como se puede ver, usamos el comando “try:”, luego el bloque de código donde pedimos el dato definiéndolo como entero y, con el comando “except ValueError:” mitigamos un posible error en el ingreso de un tipo de dato que no sea el solicitado.

Algo que también comenzamos a ver acá es el uso de la INDENTACIÓN, que es algo que ya viste en la etapa 1 con HTML, pero ahora vamos a profundizar en esto.

2

PEP8: Codea con estilo en Python

PEP8 es una herramienta valiosa para cualquier desarrollador de Python que quiera escribir código claro, consistente y fácil de mantener, por eso vamos a ver la importancia de mantener esta guía de estilos.

Python: Guía de estilos - PEP8

Entonces, teniendo en cuenta que ya vimos la sintaxis básica de Python con la declaración y asignación de variables, tipos de datos y su conversión, o el ingreso de datos y el manejo de errores, es momento de hablar del sistema de normas o estilos y convenciones que debemos manejar en Python.

PEP8 (Python Enhancement Proposal 8)

Es una guía de estilo que define las convenciones de escritura de código para el lenguaje de programación Python. Su objetivo es promover un código legible, consistente y fácil de mantener por parte de diferentes desarrolladores.

¿Por qué es importante usar PEP8?

Seguir las convenciones de PEP8 ofrece varias ventajas:

- **Legibilidad:** Un código bien formateado y con sangrías consistentes es más fácil de leer y comprender para otros desarrolladores, lo que facilita la colaboración y el mantenimiento del código.
- **Consistencia:** PEP8 ayuda a garantizar que el código de diferentes desarrolladores tenga un aspecto y una sensación similar, lo que mejora la coherencia y la profesionalidad del proyecto.
- **Facilidad de mantenimiento:** Un código que sigue las convenciones de PEP8 es más fácil de modificar y depurar, ya que es más predecible y consistente en su estructura.

Python: Guía de estilos - PEP8

- **Herramientas de análisis:** Existen herramientas que pueden analizar tu código Python y detectar violaciones de las convenciones de PEP8, lo que te ayuda a identificar y corregir problemas de estilo antes de que se conviertan en problemas mayores.

¿Qué cubre PEP8?

La guía PEP8 cubre una amplia gama de aspectos relacionados con el estilo de código Python, incluyendo:

- **Indentación:** Especifica cómo se deben realizar las sangrías en las líneas de código para mejorar la legibilidad y la estructura del código.
- **Nombres de variables:** Define convenciones para nombrar variables, funciones y clases de manera clara y descriptiva.
- **Espacios en blanco:** Regula el uso de espacios en blanco, tabuladores y otras formas de espacios en blanco para mantener una alineación y una estructura consistentes.
- **Importaciones:** Define cómo se deben organizar e importar los módulos en un proyecto Python.
- **Comentarios:** Proporciona pautas para escribir comentarios útiles y claros que documenten el código.
- **Declaraciones y expresiones:** Especifica cómo se deben formatear diferentes tipos de declaraciones y expresiones en Python.

Python: Guía de estilos - PEP8

¿Cómo puedo aprender PEP8?

Existen varios recursos disponibles para aprender PEP8:

- **Documentación oficial de PEP8:** <https://peps.python.org/pep-0008/>
- **Herramientas de análisis de código:** Existen herramientas como [pycodestyle](#), [flake8](#) y [black](#) que pueden ayudarte a analizar tu código Python y detectar violaciones de las convenciones de PEP8.
- **Libros:** Se han escrito varios libros sobre PEP8, como "[Expert Python Programming](#)" de [Michal Jaworski](#) y [Tarek Ziadé](#).

Beneficios de usar PEP8:

- **Mejora la calidad del código:** Un código que sigue las convenciones de PEP8 es más probable que esté bien escrito, libre de errores y fácil de mantener.
- **Aumenta la colaboración:** PEP8 facilita la colaboración entre desarrolladores, ya que todos comparten un entendimiento común de cómo se debe escribir el código Python.
- **Mejora la reputación profesional:** Escribir código que sigue las convenciones de PEP8 demuestra profesionalismo y atención al detalle, lo que puede mejorar tu reputación como desarrollador.

3

Integración de conceptos

Arrancamos esta etapa con los conceptos de lógica de programación, algoritmos, sintaxis básica, pseudocódigo y fuimos viendo de a poco la traducción al lenguaje de programación Python.

Para integrar todo, vamos a estar realizando un repaso de lo aprendido. 😊

PSeint y Python: Integración de conceptos

Pseudocódigo VS Código: Variable

Durante las clases anteriores pudimos ver pseudocódigo y código donde notamos sus similitudes y diferencias, por eso, ahora vamos a hacer un repaso general para consolidar lo aprendido.

Pseudocódigo	Código
<pre>1 Algoritmo variables 2 3 Definir saludo Como Caracter 4 5 saludo ← '¡Hola mundo!' 6 Imprimir saludo 7 8 FinAlgoritmo</pre>	<pre>1 saludo = '¡Hola mundo!' 2 print(saludo)</pre>

Al margen de la estructura del pseudocódigo donde se debe definir la variable, la parte de las acciones o parte lógica, no tiene mucha variación con Python.

En ambos casos se asigna un valor y se muestra por pantalla prácticamente de la misma forma.

PSeint y Python: Integración de conceptos

Pseudocódigo VS Código: Tipos de

Pseudocódigo	Código
<pre>3 Definir numero_entero Como Entero 4 Definir numero_decimal Como Real 5 Definir cadena_de_caracteres Como Caracter 6 Definir logico_booleano Como Logico</pre>	<pre>1 numero_entero = int 2 numero_decimal = float 3 cadena_de_caracteres = str 4 logico_booleano = bool</pre>

Con los tipos de datos también ocurre algo similar.

En pseudocódigo hay que definirlos, sin embargo en Python, basta con declarar la variable y asignar el valor que se quiera a la variable y Python lo reconocerá gracias al Tipado Dinámico de Python.

PSeint y Python: Integración de conceptos

Pseudocódigo VS Código: Estructuras de control: Condicionales

Pseudocódigo

```
variable_1 = 10
variable_2 = 20
Si variable_1 > variable_2 Entonces
    Imprimir 'La variable 1 es mayor.'
SiNo Si variable_1 < variable_2 Entonces
    Imprimir 'La variable 2 es mayor.'
SiNo
    Imprimir 'Las variables son iguales.'
FinSi
FinSi
```

Código

```
variable_1 = 10
variable_2 = 20
if variable_1 > variable_2:
    print('La variable 1 es mayor.')
elif variable_1 < variable_2:
    print('La variable 2 es mayor.')
else:
    print('Las variables son iguales.')
```

En los condicionales tampoco hay muchas diferencias. En el pseudocódigo, como siempre, hay que definir a las variables (solo que en esta imagen lo pasamos por alto), y luego asignar los valores, pero en Python, solo se declara y se asigna el valor.

La sintaxis como tal tiene pocas diferencias (las cuales ya vimos en la clase pasada).

PSeint y Python: Integración de conceptos

Pseudocódigo VS Código: Estructuras de control: Bucles

Pseudocódigo

```
i = 0
Mientras i < 5 Hacer
    i = i + 1
    Imprimir i
Fin Mientras
```

```
Para i Desde 1 Hasta 5 Hacer
    Imprimir i
Fin Para
```

Código

```
i = 0
while i < 5:
    i += 1
    print(i)
```

```
for i in range(1, 6):
    print(i)
```

Por último tenemos a los bucles, donde seguimos viendo similitudes muy grandes entre el pseudocódigo y el código.



Lo que vimos hoy 🤔

A lo largo de estas cuatro clases vimos varias cosas que te introdujeron al mundo de la programación mediante lógica de programación, conceptos básicos, prácticas y, paulatinamente el pase al lenguaje de programación que vamos a usar hasta el final de cursada, Python.

Con esto te quisimos mostrar que, por más que la programación como tal, sea en inglés, en un lenguaje de programación como Python, es muy fácil interpretar su sintaxis como así también, sus palabras reservadas o comandos, por lo que solo resta seguir practicando para seguir incorporando la lógica de programación que se necesita para ser un buen programador y tener el pensamiento crítico y eficiente para resolver problemas.



**¡Nos vemos
En la próxima
clase!**

