TEXT CLASS REVIEW

TEMAS A TRATAR EN EL CUE

- Instrucción Print.
- Operaciones aritméticas básicas.
- Sintaxis básica: nombre de variables, métodos, indentación y comentarios.
- Manejo de módulos: importación, módulos como scripts, ruta de búsqueda, archivos compilados, la función dir().
- Manejo de paquetes: importación, referencias internas y paquetes en múltiples directorios.

INSTRUCCIÓN PRINT

Desde tiempos inmemorables, la primera lección de código en cualquier lenguaje de programación es la impresión de texto en la pantalla, el famoso "hola mundo", y esta vez no será la excepción.

Para imprimir texto en pantalla, Python cuenta con la función `print()`, y su sintaxis es muy sencilla. Dentro de los paréntesis pondremos lo que deseemos imprimir; si se encuentra dentro de comillas, imprimirá textualmente lo que se encuentre dentro de ellas, y de lo contrario, imprimirá el valor de la variable que coloquemos dentro del paréntesis.

En caso de que el texto se escriba directamente en la función, la sintaxis es:

```
1 print('Texto a imprimir')
```

Por otro lado, en caso de que imprimamos el valor de una variable, la sintaxis es:

```
1 print(variable_a_imprimir)
```

OPERACIONES ARITMÉTICAS BÁSICAS

Python nos permite realizar las siguientes operaciones aritméticas básicas:

- Suma (+)
- Resta (-)

- Multiplicación (*)
- División (/)
- División entera (//) – devuelve solo la parte entera del resultado de la división.
- Módulo o resto (%) – devuelve el resto de la división de 2 números.
- Potencia (**) – devuelve el resultado en elevar un número a otro.

Python automáticamente imprimirá en pantalla el resultado de una operación aritmética.

```
>>>3+2
5

>>>5*4
20

>>>7 % 2
1
```

SINTAXIS BÁSICA

La sintaxis de Python es relativamente sencilla, pues omite gran parte de los elementos que usualmente son usados en otros lenguajes de programación, y está diseñada para que sea lo más parecida posible a como escribiríamos en nuestro idioma, es decir, que es similar al seudocódigo.

Los **nombres de variables** no ameritan símbolos adicionales como el **$** para designarlos, basta con escribir el nombre de la variable que se desea, y asignarle su valor. Es importante tomar en cuenta que existe diferenciación entre mayúsculas y minúsculas; el nombre de la variable no puede empezar con un número, y tampoco está permitido el uso de guiones (-), o de espacios en blanco Ejemplo:

```
>>>cadena = "Esto es un segmento de texto"
```

La **indentación** es un elemento de gran importancia en Python. A diferencia de otros lenguajes de programación, las llaves no se utilizan para crear bloques de código, sino que en su lugar se utiliza la indentación. Es posible usar tabulación o espacios, la regla general es que se utilicen 4 espacios en caso de usar estos últimos.

```
1 if True:
2     print("True")
```

En caso de no usar la indentación, el código podría dar un resultado completamente inesperado, o simplemente arrojar un mensaje de error.

El uso de punto y coma (;) al final de una línea de código no es necesario en Python.

Cuando necesitamos crear **comentarios** en nuestro código con el fin de brindar información sobre lo que se está haciendo en un lugar específico de nuestra programación, debemos preceder a crear el comentario con el símbolo numeral (#). De esta forma le estamos indicando a Python que lo que venga después en esta línea es un comentario.

```
1 # Esto es un comentario
```

También es posible comentar grandes segmentos de texto colocando 3 comillas simples (‘’’), o dobles (“””), al principio y al final del bloque de texto que se desee comentar.

```
1 """Todo esto es un comentario"""
```

MANEJO DE MÓDULOS

Los módulos son archivos con extensión **.py**, los cuales permiten agrupar código relacionado. De esta manera podemos organizar eficientemente nuestro código de programación. Por ejemplo, podríamos tener un módulo de funciones, de clases o de variables, así como de código ejecutable, el cual podríamos llamar **utilidades.py**.

Para hacer uso de un módulo, primero debemos importarlo con la sentencia **import**. Esta sentencia la podemos utilizar en cualquier lugar del código de programación y, una vez ejecutada, tendremos acceso a todo el código contenido dentro del archivo importado.

Con **import** podemos importar uno o varios módulos en la misma sentencia. Para ello solo debemos separar el nombre de los módulos con una coma. No es necesario agregar la extensión del archivo.

```
1 import utilidades, funciones
```

Una vez que salimos del intérprete de Python, todo el código que hayamos escrito allí se pierde, razón por la cual crear módulos es de gran utilidad cuando se requiera utilizar el código una y otra vez. Cuando creamos un módulo con código ejecutable básicamente estamos creando un **script.**

Debemos tener en cuenta que Python solo ejecutará un módulo en la primera llamada que se haga a él dentro del código de programación; por lo que si dentro del código se hace otra llamada al mismo modulo, se omitirá su ejecución.

Cuando importamos un módulo, Python primero buscará en el directorio actual el archivo importado, y en caso de no encontrarse allí, buscará en **ruta de búsqueda** en la variable de entorno **PYTHONPATH** del sistema operativo.

PYTHONPATH puede ser configurada en la ventana de consola haciendo uso del siguiente código:

```
set PYTHONPATH = C:\python20\lib;
```

Para el caso del Sistema Operativo Windows o Archivos compilados.
Con la función **dir()** podemos consultar todas las propiedades y métodos de un objeto, sin tomar en cuenta sus valores. Incluso las propiedades incorporadas que están por defecto en todos los objetos son devueltas por la función.

Su sintaxis es:

```
dir(objeto)
```

Un ejemplo de uso sería:

```
>>>number = [12]
>>>print(dir(number))

# Output: ['__add__', '__class__', '__contains__', '__delattr__',
'__delitem__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__',
'__getattribute__', '__getitem__', '__gt__', '__hash__', '__iadd__',
'__imul__', '__init__', '__init_subclass__', '__iter__', '__le__',
'__len__', '__lt__', '__mul__', '__ne__', '__new__', '__reduce__',
'__reduce_ex__', '__repr__', '__reversed__', '__rmul__', '__setattr__',
'__setitem__', '__sizeof__', '__str__', '__subclasshook__', 'append',
'clear', 'copy', 'count', 'extend', 'index', 'insert', 'pop', 'remove',
'reverse', 'sort']
```

MANEJO DE PAQUETES

Un paquete es una carpeta que contiene varios módulos. Esta carpeta debe contener un archivo llamado **__init__.py** para que Python entienda que se trata de un paquete.

Existen varias formas para hacer la importación de un paquete, todas ellas haciendo uso de la sentencia **import**.

Supongamos que tenemos un paquete llamado **matemática**. La carpeta debe tener el nombre **matemática**, y también que dentro tenemos 2 módulos: **aritmetica.py** y **geometria.py**; así como nuestro archivo **__init__.py**. De la misma manera, que el módulo **matemática** contiene una función

llamada `sumar`. Entonces tenemos las siguientes formas de escribir la sentencia `import` para usar la función sumar:

a)

```
>>>import matematica.aritmetica
>>>print(matemática.aritmetica.sumar(7, 5))
```

En esta forma del `import`, al hacer uso del módulo es necesario escribir la dirección completa de la función, es decir: `paquete.modulo.funcion()`.

b)

```
>>>from matematica import aritmetica
>>>print(aritmetica.sumar(7, 5))
```

En esta otra forma del `import`, ya no es necesario escribir el nombre del paquete al usar la función correspondiente, es decir, solo basta con escribir en la forma: `modulo.funcion()`.

c)

```
>>>from matematica.aritmetica import sumar
>>>print(sumar(7, 5))
```

También es posible que un paquete se divida en subpaquetes. Es decir, que una carpeta tendrá subcarpetas. Para que una subcarpeta sea considerada un subpaquete debe contener el archivo **__init__.py**. Por ejemplo, supongamos que tenemos un paquete llamado *sound*, y tres subpaquetes llamados: *formats*, *effects* y *filters*.

La forma de usar un subpaquete seria idéntica a las anteriores. Sin embargo, debemos agregar un punto adicional para cada subnivel, es decir:

```
1 import paquete.subpaquete.modulo
```

También es posible hacer referencias internas, es decir, importar desde un módulo a otro módulo que se encuentra en un subpaquete diferente. Para ello lo hacemos de la siguiente manera:

```
1 from paquete.subpaquete import modulo
```