

Guía del Examen – Python

septiembre de 2019

by Felipe Ramírez, PhD.

PRELIMINAR

| | |
|---|----|
| GitHub | 2 |
| Python: Naturaleza del Lenguaje | 3 |
| Ambientación | 4 |
| Visual Studio Code..... | 5 |
| Lenguaje Python | 6 |
| Programas | 8 |
| HolaMundo.py: print, # | 8 |
| Conversiones.py: type(), int() y str.format()..... | 9 |
| Aleatorio.py: import, def, random.randrange() | 10 |
| Entrada.py: input(), str.isdigit(), str.find()..... | 11 |
| Nombre.py: Concatenación, str.upper()..... | 12 |
| Tabla.py: for, range()..... | 13 |
| Tablas.py: for else, nested for | 14 |
| Compara.py: nested if | 15 |
| Acumulado.py: while, break, +=..... | 16 |
| Multiplo.py: %, and, or..... | 17 |

GitHub

La exposición de todos los programas del curso deberá realizarse a través de la plataforma Git.

1. Genera una cuenta para plataforma Git.
2. Descarga e instala **Github Desktop**, en tu equipo.

<https://github.com/>

3. Toma el curso **Git y Github | Curso Práctico de Git y Github Desde Cero**, by Fazt

<https://www.youtube.com/watch?v=HiXLkL42tMU>

4. Saber qué es un **Repositorio** y qué es un **Branch**.
5. Usando tu cuenta, genera un repositorio llamado **AprendiendoPython**. Genera un **Readme** usando MD, donde expliques que es un repositorio de ejercicios para tu propio aprendizaje.
 - a. Cuando encargue programas en Python, solicitaré la liga para acceder a su repositorio, clonaré sus archivos, y revisaré a partir de ahí.
6. Conocer los comandos básicos de Git:
 - a. Init
 - b. Add
 - c. Status
 - d. Commit
 - e. Push
 - f. Pull
 - g. Clone

Python: Naturaleza del Lenguaje

En términos generales, esto es lo que puede decirse de Python.

1. **Propósito General.** Permite desarrollar aplicaciones de cualquier tipo de rama del conocimiento y cualquier tipo de aplicación.
2. **Tercera Generación.** Porque está basado en un conjunto de instrucciones que permiten detallar de forma procedural y secuencial la forma en cómo han de realizarse las tareas.
3. **Orientado a objetos.** Porque permite la creación e instanciación de clases, y permite implementar las tres características de POO: Encapsulamiento, Herencia y Polimorfismo.
4. **Interpretado.** Porque cada vez que se manda a ejecución un programa, se realiza la verificación sintáctica (*parsing*) y la generación de código binario ejecutable para la plataforma en que es ejecutado (compilación).
5. **Sensible a mayúsculas y minúsculas (*case sensitive*).** Porque el uso de mayúsculas y minúsculas sí importa, tanto para las palabras reservadas, como para el nombre de los elementos creados por el desarrollador.
6. **Posicional (*Indented*).** Porque la posición en la que inicia una línea de código influye en la manera en que el interprete de Python produce resultados.
7. **No requiere la especificación detallada de tipos (*Non Strong Type*).** Las variables no requieren declararse antes de utilizarse, ni deben utilizarse si son declaradas.
8. **Sin terminador de línea.** Python no requiere un terminador de línea en específico, basta pasar a la siguiente línea para dar por terminada la anterior.

Ambientación

Para desarrollar aplicaciones Python en ambiente Windows, se requiere instalar varios componentes, en la secuencia que se sugiere.

1. Instalar **Anaconda 3** (*Python engine & Data Science Libraries*). Incluye Python 3.7

https://repo.anaconda.com/archive/Anaconda3-2019.07-Windows-x86_64.exe

2. Instalado **Anaconda**, es importante saber dónde quedó instalado el interprete **Python.exe**, y comprobar que funciona.
 - a. Identifica la ruta completa (*path*) donde está el programa **python.exe**
 - b. Agrega la ruta que descubriste, a la variable de entorno **PATH**, del sistema operativo.
 - c. Desde línea de comandos de Windows, comprueba que Python funciona.
 - i. Presiona **Window+R**.
 - ii. Aparecerá la ventana **Ejecutar**.
 - iii. Escribe **cmd.exe** y presiona **Aceptar**.
 - iv. En la línea de comandos, escribe: **python --version**
 - v. Si aparece en la ventana la versión que tienes instalada de Python, todo va bien.
 - vi. Sal de la consola, con el comando **exit**.

3. Instalar **Visual Studio Code** (conocido simplemente como “**Code**”)

<https://code.visualstudio.com/download#>

- a. Agrega la ruta de **python.exe** a la declaración **python.pythonPath** en **settings.json**
 - i. En **Code**, selecciona en el menú **File – Preferences – Settings**.
 - ii. En la barra de búsqueda (**Search**), busca **settings.json**
 - iii. Haz clic en el vínculo **Edit in settings.json**
 - iv. Asegúrate que entre las llaves se encuentra la ruta del interprete.

“python.pythonPath”: “Colocar la ruta donde está python.exe”

Ejemplo.

“python.pythonPath”: “C:\\ProgramData\\Anaconda3”

Es importante que, en la ruta, los back slash sean dobles (\\)

4. Instala las siguientes extensiones de **Code**.
 - a. Extensiones: **Anaconda Extension Pack, by Microsoft**
 - b. Extensiones: **Code Runner, by Jun Han**
 - c. Extensiones: **Python, by Microsoft**
 - d. Extensiones: **Trailing Spaces, by Shardul Mahadik**
5. Salir de **Code** y entrar de nuevo.

Visual Studio Code



Entender el uso de **Extensions** (CTRL+MAYUSC+X)



Entender el uso de **Explorer** (CTRL+MAYUSC+E), y la jerarquía que hay entre Folder – Workspace – Files.

Seleccionar el motor de Python a utilizar.

Saber el uso de **Intellisense** (CTRL+Barra espaciadora)

Saber el uso de **Code Snippets**

Mandar a ejecutar código (CTRL+ALT+N)

Cómo se graba un archivo: **File – Save** (CTRL+S)

Habilita a **Code** para poder recibir datos desde la consola. En menú, selecciona **File – Preference – Settings**. En búsqueda, colocar **Run Code**, y en **Code-Runner: Run in terminal**, marca la casilla de verificación **whether to run code in integrated terminal** y reinicia **Code**.

Lenguaje Python

Todos los temas contenidos en esta guía pueden ser consultados en la que es, a mi juicio, la mejor referencia en línea del lenguaje Python, a nivel introductorio, que es

w3schools.com

Consulta: https://www.w3schools.com/python/python_intro.asp

6

- 1.Cuál es el terminador de línea en Python (**Enter**).
2. Cómo defino un bloque de código en Python (*Indentacion*).
3. Cómo se colocan comentarios en Python (**#**).
4. Cómo genero una salida de consola en Python (**print**).
5. Cómo se definen las variables en Python.
6. Cuáles son las reglas de nombrado de variable en Python.
7. Qué tipos de notación son los más utilizados en la actualidad para el nombrado de elementos en un programa:

NombreUsuario - Pascal Casing (Todos los inicios de palabra en mayúsculas)


nombreUsuario - Camel Casing (Primera minúscula, y luego mayúsculas los inicios de palabra.

objLista – Hungarian notation (Incluye referencia al tipo, como prefijo o como sufijo)

8. Qué notación se sugiere para el nombrado de variables en Python (*camelCase*).
9. Qué notación se sugiere para el nombrado de clases, métodos y funciones (*PascalCase*).
10. Cómo se declaran funciones simples, que no reciben parámetros, y no retornan valores (**def**).
11. Qué son las variables globales.
12. Que son las variables locales.
13. Para que sirve **global**
14. Cuáles son los *data types* admitidos por Python
15. Función para determinar el tipo de dato de una variable (**type()**)
16. Funciones para el establecimiento explícito de *data type*.
17. Funciones de conversión de tipos de datos.
18. Uso de **str.isdigit()**.
19. Cómo se importan librerías a un programa Python (**import**)
20. Función de generación de números aleatorios (**random.randrange()**). Requiere importar la librería (**random**).
21. Función para preguntar información desde la consola (**input()**)
22. Trabajo con strings.
 - a. Uso de literal.
 - b. Uso de multilínea.
 - c. Uso de **str.len**.
 - d. Métodos:
 - i. **str.strip**
 - ii. **str.lower**

- iii. `str.upper`
 - iv. `str.replace`
 - v. `str.split`
 - vi. `str.partition`
 - e. `not in`
 - f. Concatenación
23. Dando formato a strings usando placeholders (`str.format`)
 24. Manejo de datos booleanos.
 25. Manejo de operadores (aritméticos, asignación, lógicos, comparación)
 - a. Sólo los más usuales.
 - b. Especial atención en el uso de `is` e `is not`, con `type()`.
 26. Manejo de `if`
 27. Manejo de `if` / `else`
 28. Manejo de `while`
 29. Manejo de `while` / `else`
 30. Uso de `break` (para ciclos infinitos)
 31. Manejo de `for`
 32. Uso de `range()` con `for`.

Programas

| | |
|--|---|
|  python™ | <h2>HolaMundo.py: print, #</h2> <p>Elabore un programa llamado HolaMundo.py, que muestre en la consola el mensaje “Hola Mundo, ¡ahora en Python!”. Al principio del programa, agregar comentarios que indiquen quién es el autor del programa, y la fecha de elaboración del programa.</p> <p>Entradas y salidas: Hola Mundo, ¡ahora en Python!</p> |
|--|---|

8

```
HolaMundo.py
1  # HolaMundo.py
2  # Autor: Felipe Ramírez
3  # Fecha de creación: 12/09/2019
4
5  # Los comentarios se realizan con #. El resto de
6  # la línea no tiene significado para el interprete.
7
8  # print genera una salida a consola. El elemento
9  # a imprimir debe delimitarse entre paréntesis.
10 print("Hola Mundo, ahora en Python!")
```




Conversiones.py: type(), int() y str.format()

Elabore un programa llamado **Conversiones.py**, que declare una variable de tipo **str** con un valor de **"1234"**, y que muestre el data type de la variable: realizar la conversión del dato a **int**, y mostrar el nueva *data type*. Mostrar también el número que se convirtió. Usar **type()**, **int()** y **str.format()**.

Entradas y salidas:

```
<class 'str'>
```

```
<class 'int'>
```

```
El número utilizado es 1234
```

9

```
Conversiones.py > ...
1  # Se declara una variable str, con una cadena de dígitos
2  numero="1234"
3  # Se muestra el tipo que tiene la variable. La salida de type()
4  # no es un str, es un dato type.
5  print(type(numero))
6  # Se convierte la cadena a su equivalenci int.
7  numero=int(numero)
8  # Se muestra cómo el tipo ha cambiado, aunque se usa la misma
9  # variable.
10 print(type(numero))
11 # Se declara un str con meta sustitución (posiciones donde irán
12 # valores proporcionados usando format.
13 salida="El número utilizado es {}"
14 # Se muestra el resultado. La meta sustitución hará que donde está
15 # {} se coloque el valor de numero.
16 print(salida.format(numero))
```



Aleatorio.py: import, def, random.randrange()

Elabore un programa llamado **Aleatorio.py**, que declare una variable global de tipo **float**, asignándole un valor cualquiera, explícitamente **float**; elabore una función que declare una variable local de tipo **float**, que adquiera un valor aleatorio entre 1 y 10, y que muestre en consola el resultado de la suma de las dos variables, usando el mensaje “La suma de x y y es z”. Usar **import**, **float()**, **def** y **str.format()**.

Entradas y salidas:

La suma de 10.5 y 5.4 es 15.9

10

```
Aleatorio.py > ...
1  # python posee muchas librerías listas para utilizarse.
2  # A dichas librerías les da el nombre de módulos (module)
3  # Para utilizar in módulo en un programa, primero debe
4  # importarse, usando la instrucción import
5  import random
6
7  # Se define una variable float, y se le asigna un valor.
8  numero1=float(10.5)
9
10 # En python, una función es un conjunto de instrucciones que
11 # procesan una tarea específica, como una unidad de ejecución.
12 # Se declaran con def. Todo el código posicionado a la derecha
13 # de la definición, forma parte de la función.
14 def miFuncion():
15     # Se convierte a float el número aleatorio generado por
16     # random.randrange() (sólo está disponible si se importa
17     # el módulo random)
18     numero2=float(random.randrange(1,10))
19     # Se utilizan meta sustituciones para mostrar resultados.
20     mensaje="La suma de {} y {} es {}"
21     print(mensaje.format(numero1,numero2,numero1+numero2))
22
23 # Se ejecuta la función definida en el código.
24 miFuncion()
```



Entrada.py: input(), str.isdigit(), str.find()

Elabore un programa llamado **Entrada.py**, que declare una variable que reciba un **valor**; muestre el *data type* del valor capturado. Si el valor capturado es **integer** (es dígito y no contiene punto), mostrar la leyenda **“Dato entero. ¡Muy bien!”** o de lo contrario, mostrar **“Dato no es entero. Intentar nuevamente.”** Ejecutar proporcionando un entero, un flotante, y una cadena. Usar **input()**, **str.isdigit()**, **str.find()**.

Entradas y salidas:

a)

12

<class 'str'>

Dato entero. ¡Muy bien!

b)

Hola

<class 'str'>

Dato no es entero. Intentar nuevamente.

c)

12.5

<class 'str'>

Dato no es entero. Intentar nuevamente.

11

Entrada.py > ...

```
1  entrada=input()
2  print(type(entrada))
3  # La variable booleana contiene el resultado de verificar
4  # si lo capturado es dígito, y si no se encuentra un punto
5  # en lo capturado, lo que indicaría se que trata de un
6  # número con decimales, es decir, float. Si find retorna -1
7  # quiere decir que lo buscado, en este caso un punto decimal
8  # no se encontró. Si esEntero es True, lo capturado es entero.
9  esEntero=(entrada.isdigit() and entrada.find(".")!=-1)
10 if (esEntero):
11     # Líneas que se ejecutarán si la condición es verdadera (True)
12     print("Dato entero. ¡Muy bien!")
13 else:
14     # Líneas que se ejecutarán si la condición es falsa (False)
15     print("Dato no es entero. Intentar nuevamente.")
```



Nombre.py: Concatenación, str.upper()

Elabore un programa llamado **Nombre.py**, que pregunte dos datos: nombre, y apellido. Los debe transformar a mayúsculas, y mostrar en forma de nombre completo (concatenación). Usar **input()** y **str.upper()**.

Entradas y salidas:

Nombre:Felipe
Apellidos:Ramírez
FELIPE RAMÍREZ

12

```
Nombre.py > ...
1  nombre=input("Nombre:")
2  apellidos=input("Apellidos:")
3  # Se concatenan los valores str, junto con la literal " "
4  nombreCompleto=nombre+" "+apellidos
5  # Se asigna a la variable la representación en mayúsculas
6  # de lo que contenía.
7  nombreCompleto=nombreCompleto.upper()
8  print(nombreCompleto)
```



Tabla.py: for, range()

Elabore un programa llamado **Tabla.py**, que pregunte un número entero del 1 al 9, y muestre la tabla de multiplicar del número proporcionado. Utilizar **for**, **int**, **range()** y **str.format()**.

Entradas y salidas:

Dame un número del 1 al 9: 4

4 x 1 = 4

4 x 2 = 8

4 x 3 = 12

4 x 4 = 16

4 x 5 = 20

4 x 6 = 24

... así hasta terminar

```
Tabla.py > ...
1  numero=input("Dame un número del 1 al 9: ")
2  numero=int(numero)
3  # for ejecuta un bloque de código un número determinado
4  # de veces, cuando se pide que recorra un rango de
5  # valores. El segundo parámetro de range no se incluye
6  # en la serie de valores. Aquí sería del 1 al 10.
7  for i in range(1,11):
8      # i va variando a cada iteración.
9      salida("{} x {} = {}".format(numero,i,i*numero))
10     print(salida.format(numero,i,i*numero))
```



Tablas.py: for else, nested for

Elabore un programa llamado **Tablas.py**, que elabore las tablas de multiplicar del 1 al 10. Cada tabla deberá tener un encabezado "Tabla del x". Entre una tabla y otra, debe haber un salto de línea. Usar **for** y **for else** anidados. Usar **range()** y **str.format()**.

Entradas y salidas:

Tabla del 1

1 x 1 = 1
1 x 2 = 2
1 x 3 = 3
1 x 4 = 4
1 x 5 = 5
1 x 6 = 6
1 x 7 = 7
1 x 8 = 8
1 x 9 = 9
1 x 10 = 10

Tabla del 2

2 x 1 = 2
2 x 2 = 4
2 x 3 = 6
2 x 4 = 8
2 x 5 = 10

... así hasta terminar

Tablas.py > ...

```
1  for i in range(1,11):
2      encabezado="Tabla del {}"
3      print(encabezado.format(i))
4      # print sin argumentos es un salto de línea.
5      print()
6      # Dentro de un for, se pone otro for.
7      for j in range(1,11):
8          # Aquí, i contiene el número base de la tabla
9          # y j el elemento de la tabla.
10         salida("{} x {} = {}".format(i,j,i*j))
11         print(salida.format(i,j,i*j))
12     else:
13         # Al concluir con las iteraciones indicadas
14         # se ejecuta este código, que es un salto de
15         # línea.
16         print()
```



Compara.py: nested if

Elabore un programa llamado **Compara.py**, que pregunte dos números, y que muestre cuál de los dos es mayor, el primero o el segundo. También debe reportar si son iguales. El mensaje debe decir:

“Números proporcionados: x y y. El mayor es primero.” (o el segundo, o son iguales, según sea el caso).

Use **str.format** e **if** anidados. Pruebe usando dos entradas: una donde el primero sea mayor, y otra donde el segundo sea mayor.

Entradas y salidas:

a)

Número 1:10

Número 2:10

Números proporcionados: 10 y 10. Los números son iguales.

b)

Número 1:10

Número 2:20

Números proporcionados: 10 y 20. El mayor es el segundo.

b)

Número 1:20

Número 2:10

Números proporcionados: 20 y 10. El mayor es el primero.

15

```
Compara.py > ...
1  numero1=input("Número 1:")
2  numero2=input("Número 2:")
3  salida="Números proporcionados: {} y {}. {}."
4  if (numero1==numero2):
5      # Entra aquí si los números capturados son iguales.
6      print(salida.format(numero1, numero2,"Los números son iguales"))
7  else:
8      # Condicionales anidadas, if dentro de otro if.
9      # Si los números no son iguales.
10     if numero1>numero2:
11         # Reporta si el primer número es mayor al segundo.
12         print(salida.format(numero1, numero2,"El mayor es el primero"))
13     else:
14         # O de lo contrario, si el primero no es mayor al segundo.
15         print(salida.format(numero1, numero2,"El mayor es el segundo"))
```



Acumulado.py: while, break, +=

Elabore un programa llamado **Acumulado.py**, que pregunte números enteros indefinidamente. Cada número que pregunte, deberá acumularlo, mostrando **"Acumulado hasta el momento: x"**. El programa no deja de preguntar números y acumularlos, hasta que se deje vacía la entrada. Use un ciclo infinito usando **while**, y **break**. Usar suma incluyente (**+=**).

Entradas y salidas:

Dame un número entero: 10
Monto acumulado: 10
Dame un número entero: 20
Monto acumulado: 30
Dame un número entero: 25
Monto acumulado: 55
Dame un número entero:
Vacío. Salida del programa.

16

```
Acumulado.py > ...
1  # Se declaran las variables de trabajo, con tipo explícito.
2  acumulado=int(0)
3  numero=str("")
4
5  # Al colocar True como condición de while, se forma un
6  # ciclo infinito que no se romperá hasta que de forma
7  # explícita se utilice la instrucción break.
8  while True:
9      numero=input("Dame un número entero: ")
10     if numero=="":
11         # Si el número es vacío, reporta la situación y sale.
12         print("Vacío. Salida del programa.")
13         break
14     else:
15         # Si se proporcionó valor, acumulado = acumulado + numero
16         # Se realiza el cálculo usndo suma incluyente (+=)
17         acumulado+=int(numero)
18         salida="Monto acumulado: {}"
19         print(salida.format(acumulado))
```




Multiplo.py: %, and, or

Elabore un programa llamado **Multiplo.py**, que pregunte un número entero. Si el número es múltiplo de 3 y múltiplo de 5, o múltiplo de 7, muestra el mensaje **“Correcto”**, de lo contrario, **“Incorrecto”**. Tip: Si un número es múltiplo de otro, módulo es cero. Usar **%**, **and** y **or**, y variables booleanas de apoyo.

Entradas y salidas:

a)

Dame un número entero: 15
Correcto.

b)

Dame un número entero: 14
Correcto.

b)

Dame un número entero: 10
Incorrecto.

✚ Multiplo.py > ...

```
1  # Se captura un número y se almacena una vez que es
2  # convertido a int
3  numero=int(input("Dame un número entero: "))
4  # Se almacenan en valores booleanos la evaluación
5  # de residuales. Si el residual es cero, quiere decir
6  # que el número es múltiplo.
7  esMultiplo3=((numero%3)==0)
8  esMultiplo5=((numero%5)==0)
9  esMultiplo7=((numero%7)==0)
10 # Cuando se usa and, se resuelve por verdadero si todas
11 # las condiciones son verdaderas. Cuando se usa or, se
12 # resuelve por verdadero si al menos una condición es
13 # verdadera. Los paréntesis le dicen a python que
14 # las primeras dos condiciones son juntas, y la tercera
15 # es independiente.
16 if ((esMultiplo3 and esMultiplo5) or esMultiplo7):
17     print("Correcto.")
18 else:
19     print("Incorrecto.")
```

Fin de la guía.

Los programas contenidos en esta guía deberán estar elaborados y en su repositorio **Git**, donde serán revisados.

En el examen es teórico / práctico, y podrán utilizarse los siguientes recursos:

- Podrá utilizarse la computadora portátil, para desarrollar y ejecutar los programas solicitados.
- Podrán consultarse hasta 3 *cheat sheet* de Python, impresas.
- Podrán consultarse los programas que se hayan cargado en su repositorio Git. Sólo se puede consultar el repositorio personal.

REVISAR CONSTANTEMENTE ESTE ARCHIVO, PORQUE PUEDE CONTENER NUEVAS INDICACIONES. LAS MODIFICACIONES SE DETENDRÁN CUANDO SEA SEÑALADO COMO **“DEFINITIVO”**.