**GUÍA DE APRENDIZAJE Nº 2**

1. **IDENTIFICACIÓN DE LA GUIA DE APRENDIZAJE**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Programa de Formación:  ANALISIS Y DESARROLLO DE SISTEMAS DE INFORMACION | Código:  Versión: | 228106  102 | | |
| Nombre del Proyecto:  IMPLEMENTACION DE UN APLICATIVO WEB PARA EL SEGUIMIENTO Y EVALUACION DE LA ETAPA PRODUCTIVA DE LOS APRENDICES SENA A TRAVES DE UNA BITACORA VIRTUAL | Código: | 1107755 | | |
| Fase del proyecto: | | ANALISIS | | |
| Actividad (es) del Proyecto:  Analizar el sistema | Actividad (es) de Aprendizaje:   * Implementar programas con la instrucción condicional if/then/else | Ambiente de formación ESCENARIO Aula 2030A | MATERIALES DE FORMACIÓN | |
| DEVOLUTIVO Televisor, tablero | CONSUMIBLE (unidades empleadas durante el programa) |
| **Resultados de Aprendizaje:**  Construir el modelo conceptual del macrosistema frente a los requerimientos del cliente, | Competencia:  Analizar los requerimientos del cliente para construir el sistema de información. | | | |
| Duración de la guía ( en horas): | 24 | | | |

1. **INTRODUCCIÓN**

|  |
| --- |
| Las funciones en python, también llamadas subrutinas, son instrucciones agrupadas bajo un mismo nombre, lo que permite modularizar el código y ordenarlo, de manera tal que una determinada tarea se puede “empaquetar” en una función para posteriormente ser llamada cada vez que sea necesario. |

1. **ESTRUCTURACION DIDACTICA DE LAS ACTIVIDADES DE APRENDIZAJE**

|  |
| --- |
| * 1. **Actividades de contextualización e identificación de conocimientos necesarios para el aprendizaje.)** * Las funciones se pueden crear en cualquier punto de un programa, escribiendo su definición.   La primera línea de la definición de una función contiene: la palabra reservada **def,** el nombre de la función (la guía de estilo de Python recomienda escribir todos los caracteres en minúsculas separando las plabras por guiones bajos) paréntesis (que pueden incluir los argumentos de la función, como se explica más adelante)   * Las instrucciones que forman la función se escriben con sangría con respecto a la primera línea. * Por comodidad, se puede indicar el final de la función con la palabra reservada return (más adelante se explica el uso de esta palabra reservada), aunque no es obligatorio. * Para poder utilizar una función en un programa se tiene que haber definido antes. Por ello, normalmente las definiciones de las funciones se suelen escribir al principio de los programas   A continuación se muestran dos ejemplos de funciones. Digítelos, compílelos y córralos  Ejemplo1:  def adsi():  print ("SENA - REGIONAL TOLIMA")  print ("CENTRO DE COMERCIO Y SERVICIOS")  print ("ADSI 1823621")  print ("ACERCA DE...")  adsi()  Ejemplo2:  def bienvenida():  print("Bienvenido al SENA\n")  nombre = input("Cuál es su nombre ? ")  edad = int(input("Cuantos años tiene ? "))  if(edad >= 18):  print(nombre, " usted es mayor de edad")  else:  print(nombre, " usted es menor de edad")  bienvenida() |
| * 1. **Actividades de transferencia del conocimiento (Conceptualización y Teorización).**   **Funciones y argumentos**  Los argumentos o parámetros generan mayor flexibilidad a las funciones, permitiendo la reutilización de código.  def factorial(numero):  facto = 1  for i in (2,numero+1):  facto = facto \* i  return facto  numero = int(input("Numero: "))  print(f"El factorial de {numero} es {factorial(numero    **Alcance o ámbito de las variables**  El uso de variables en subrutinas puede ocasionar conflictos entre los nombres de estas. Los lenguajes de programación limitan lo que se llama alcance o ámbito de las variables; esto quiere decir que los lenguajes de programación permiten que una variable exista solamente dentro de una función y no afecte el nombre a otras variables situadas fuera de dicha función. Como las subrutinas pueden contener otras subrutinas en su interior, podemos decir que existe una jerarquía; donde el programa principal tiene el nivel más alto, el siguiente nivel serían las subrutinas (procedimientos, funciones y/o métodos; dependiendo del lenguaje de programación) y se irá descendiendo para las subrutinas contenidas en otras – practica no recomendada -.  Pero un problema surge cuando una función necesita tener acceso a una variable que este en otro punto del programa. De manera que los lenguajes de programación establecen mecanismos para aislar las variables y evitar conflictos de nombres, pero también para accesarlas desde cualquier punto del programa en el caso de ser necesario.    **Características del comportamiento de las variables:**   * Cada variable pertenece a un ámbito determinado * Las variables definidas dentro de una función no son visibles fuera de esta. * Las variables son visibles en ámbitos inferiores al que pertenecen (el lenguaje permite elegir esa opción al programador. * En cada función las variables que se utilizan pueden ser:   + Locales: las que pertenecen a la función y son visibles en ámbitos inferiores.   + Libres: las que pertenece a niveles superiores pero no son visibles en las subrutinas.   **Características del comportamiento de las variables en Python**   * Python distingue tres tipos de variables:   + **variables locales**: Cuando creamos una variable dentro de una función, es local por   defecto. Pertenecen al ámbito de la subrutina (y que pueden ser accesibles a niveles inferiores)   * + **variables globales**: Cuando definimos una variable fuera de una función, es global por   defecto. Pertenecen al ámbito del programa principal. Puede ser accesada dentro de la función, pero no puede ser modificada; a menos que se adicione el modificador de acceso “global”   * + **variables no locales:** las que pertenecen a un ámbito superior al de la subrutina. * Si el programa contiene solamente funciones que no contienen a su vez funciones, todas las variables libres son variables globales. Pero si el programa contiene una función que a su vez contiene una función, las variables libres de esas "subfunciones" pueden ser globales (si pertenecen al programa principal) o no locales (si pertenecen a la función). * Para identificar explícitamente las variables globales y no locales se utilizan las palabras reservadas global y nonlocal. Las variables locales no necesitan identificación. La palabra reservada nonlocal se introdujo en Python 3 ([PEP 3104](https://www.python.org/dev/peps/pep-3104/)).   **Variables locales**  **Ejemplo 1:**  def variables\_locales\_ej1():  a = 5  print(a)  a = 15  variables\_locales\_ej1()  print(a)  Si no se han declarado como globales o no locales, las variables a las que se les asigna un valor en una función se consideran variables locales; es decir solo existen en la propia función.  **Ejemplo 2:**  def variables\_locales\_ej2():  print(a)  a = 5  print(a)  a = 100  variables\_locales\_ej2()  print(a)  El anterior programa genera siguiente error:  Traceback (most recent call last):  File "C:/Users/DELL/AppData/Local/Programs/Python/Python37/variables\_locales\_ej2.py", line 7, in <module>  variables\_locales\_ej2()  File "C:/Users/DELL/AppData/Local/Programs/Python/Python37/variables\_locales\_ej2.py", line 2, in variables\_locales\_ej2  print(a)  UnboundLocalError: local variable 'a' referenced before assignment  Lo anterior debido a que la variable “a” de la función variables\_locales\_ej2() es local, debido a que se le asigna un valor dentro de la misma, y está siendo referenciada antes de la asignación.  **Variables libres globales y no locales**  Si a una variable no se le asigna un valor en una función, Python la considera libre y busca su definición en los niveles superiores, hasta el programa principal. Si a la variable se le asigna un valor en un nivel intermedio, se considera no local y si se le asigna un valor en el programa principal, se le considera global.  **Ejemplo 3 (variable global)**  def variables\_globales():  print(a)  a = 1500 **#variable global**  variables\_globales()  print(a)  En el anterior programa la variable a se le considera global porque toma su valor del programa principal.  **Ejemplo 4 (variable global)**   1. a = 1500 **#variable global** 2. def variables\_globales(): 3. a = a + 1 4. print(a) 5. variables\_globales() 6. print(a)   El anterior programa, genera un error en la línea 3, debido a que aunque se ha definido la variable **a** fuera de la función, al interior de ella se le está haciendo una asignación; motivo por el cual el la enriende como si fuera local; pero no se ha definido. Para que la variable global “a” pueda ser modificada, se necesita hacer uso de la palabra **global**  **Ejemplo 5 (Uso de la palabra global)**  a = 0 **# global variable**  def fun\_var\_global():  global a  a = a + 2  print("Dentro de fun\_var\_global:", a)  print("Programa principal", a)  fun\_var\_global()  Nota: el uso de la palabra global fuera de la función no tiene ningún efecto  **Ejemplo 5 (variable nonlocal)**  def funcion\_a():  x = 10  def funcion\_sub\_a():  nonlocal x  x = 15  print(f"funcion\_sub\_a() id={id(x)} valor de x= {x}")  funcion\_sub\_a()  print(f"funcion\_sub\_a() id={id(x)} valor de x= {x}")  funcion\_a()    En el anterior ejemplo se define la variable **x** en la **función\_a()** y se la asigna 10. Dentro de esta misma función se define la subfuncion **función\_sub\_a()** y se define la variable **x** como **nonlocal,** lo cual causa que se genere una nueva dirección para la variable x, la cual también es asignada a la variable x de a función **función\_a()**  **Parámetros por valor y/o por referencia**  En algunos lenguajes, en donde las variables almacenan valores, cuando estas se envían como argumentos, se puede hacer de la siguiente manera:   * + - Por valor: se envía el valor, de manera tal que la función no puede modificar la variable     - Por referencia: se envía la dirección de la variable en la memoria, de manera que la función puede modificar su contenido.   En python, cuando se envía una variable como argumento, siempre se envía la referencia al objeto; el que se pueda modificar o no, depende de si el objeto es mutable o inmutable.  En el siguiente código se envían como parámetros dos objetos inmutables, por lo cual los parámetros formales (numero1 y numero2) no pueden ser alterados en la función intercambio(parametro1, parametro2)  **Ejemplo 6**  def intercambio(a,b):  aux = a  a = b  b = aux  numero1 = int(input("Numero1: "))  numero2 = int(input("Numero2: "))  intercambio(numero1,numero2)  print(f"Numero1 = {numero1}")  print(f"Numero2 = {numero2}")  El siguiente código hace uso funciones con paso de parámetros, usando como argumentos una lista, la cual es un objetos mutable; lo cual causa que la función pueda cambiar el valor de los parámetros actuales.  **Ejemplo 7**  def intercambio(lista):  aux = lista[0]  lista[0] = lista[1]  lista[1]= aux  lista += [7]  print('-------------------- Intercambio -------------------')  print(lista)  numeros = [2,3]  print('---------------------------------------------------------')  print(numeros)  intercambio(numeros)  print('---------------------------------------------------------')  print(numeros)  **Retornando múltiples valores**  También es posible que una función retorne múltiples valores  **Ejemplo 8**  def suma\_y\_media(n1,n2,n3):  suma = n1 + n2 + n3  media = suma / 3  return suma,media  suma, media = suma\_y\_media(5,10,15)  print('Suma = ',suma)  print('Media = ',media)  **Parámetros arbitrarios**  A continuación, mostraremos, como una función puede admitir una cantidad indeterminada de valores.  Para definir argumentos arbitrarios en una función, se antecede al parámetro un \*.  def media(\*args):  total = 0  for i in args:  total += i  return total / len(args)  print (f"La media de 5,7,14 es {media(5,7,14)}")  Nota: si la función espera recibir parámetros fijos y arbitrarios, los atributos fijos, siempre deben preceder a los arbitrarios.  **Parámetros clave-valor**  Python también permite definir y pasar parámetros del tipo clave-valor; para lo cual se debe anteceder el nombre del parámetro por dos asteriscos  def contactos(\*\*lista):  for clave in lista:  print(clave,' - ',lista[clave])    contactos(amor=3111111,camila=355555,karol=3211544)  **Desempaquetando parámetros**  También se puede presentar la situación inversa; es decir que la función espere una lista fija parámetros, pero estos al momento de enviarlos, se encuentran en una lista o tupla. En este caso el signo asterisco debe preceder el nombre de la lista que es pasada como parámetro durante la llamada a la función. Ejemplo:  def suma(n1,n2,n3):  return n1+n2+n3  numeros = [101,102,103]  print (f"{numeros} = {suma(\*numeros)}")  **Retornando múltiples valores y parámetros arbitrarios**  También es posible que una función retorne múltiples valores  def media\_y\_desviacion(\*args):  total = 0  for i in args:  total += i  media = total / len(args)  total = 0  for i in args:  total += (i - media) \*\* 2  desviacion = (total / len(args)) \*\* 0.5  return media,desviacion  a,b,c,d = 10,20,25,35  media,desviacion = media\_y\_desviacion(a,b,c,d)  print(f"Datos {a} {b} {c} {d}")  print(f"Media = {media}")  print(f"Desviacion = {desviacion}")  **Parámetros por defecto**  Los parámetros por defecto permiten que la función sea llamada con menos argumentos de los que se esperan. Ejemplo1:  def saludo(nombre,mensaje = "Bienvenido"):  print (mensaje, nombre)    saludo("Fernando")  La salida de este programa es: **Bienvenido Fernando**  Ejemplo2:  def saludo(nombre,mensaje="Bienvenido"):  print (mensaje, nombre)    saludo("Fernando","Hola")  La salida de este programa es: **Hola Fernando**  **Funciones anónimas**  Las funciones anónimas están pensadas para implementar cálculos sencillos. Para crear una función anónima se utiliza la instrucción **lambda.**   |  |  | | --- | --- | | **Función tradicional** | **Función anónima** | | def potencia(base,exponente):  return base \*\* exponente | pot = lambda base,exp: base \*\* exp |   Cómo en la definición de funciones anidadas, las formas con lambda pueden hacer referencia a variables del ámbito en el que son contenidas.  A continuación un ejemplo con el código completo haciendo uso de las anteriores funciones  **#Estilo tradicional**  def potencia(base,exponente):  return base \*\* exponente  **#Funcion anónima**  pot = lambda base,exp: base \*\* exp  base = int(input("Base: "))  exp = int(input("Exponente: "))  **#Llamado a funciones**  print(f"{base} elevado a la {exp} = {potencia(base,exp)}")  print(f"{base} elevado a la {exp} = {pot(base,exp)}") |
| * 1. **Actividades de apropiacion del conocimiento (Conceptualización y Teorización).**       + Implemente un programa que solicite dos números y almacene sus valores en dos variables, y a través de una función con parámetros se intercambien sus valores.      + Implemente un programa usando funciones con parámetros arbitrarios, para calcular la suma y el promedio de una lista de números.      + Implemente una función que reciba parámetros arbitrarios y retorne la suma y promedio.      + Implemente una función que retorne una lista con la serie de Fibonacci, la cantidad de números de la serie debe ser solicitada por consola. |

1. **RECURSOS PARA EL APRENDIZAJE**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **ACTIVIDADES DEL PROYECTO** | **DURACIÓN  (Horas)** | **Materiales de formación devolutivos: (Equipos/Herramientas)** | | **Materiales de formación (consumibles)** | | **Talento Humano (Instructores)** | | **AMBIENTES DE  APRENDIZAJE TIPIFICADOS** |
| **Descripción** | **Cantidad** | **Descripción** | **Cantidad** | **Especialidad** | **Cantidad** | **ESCENARIO (Aula, Laboratorio, taller, unidad productiva)** y elementos y condiciones de seguridad industrial, salud ocupacional y medio ambiente |
|  |  |  |  |  |  |  |  |  |

|  |
| --- |
| **JDK**  **J2SE**  **Java** |

1. **GLOSARIO DE TERMINOS**
2. **REFERENTES BIBLIOGRÁFICOS**

|  |
| --- |
| Harvey M. Deitel, Paul J. Deitel (2004). Como Programar en Java, Pearson Education.  [Froufe Quintas, Agustín](http://biblioteca.sena.edu.co/F/8IVEUS4EGGBXJ5SGUXR2QRQLL3TFQ63B6GFACX6M9J3XX3KCS5-24793?func=full-set-set&set_number=009595&set_entry=000006&format=999) (2009). [Java 2 : manual de usuario y tutorial](http://biblioteca.sena.edu.co/F/8IVEUS4EGGBXJ5SGUXR2QRQLL3TFQ63B6GFACX6M9J3XX3KCS5-24794?func=full-set-set&set_number=009595&set_entry=000006&format=999), Alfaomega Ra-Ma  Flórez Fernández, Héctor Arturo (2012). Programación Orientada a Objetos Usando Java,  [Ecoe Ediciones](http://www.digitaliapublishing.com.bdigital.sena.edu.co/s/e/179).  <http://www.digitaliapublishing.com.bdigital.sena.edu.co/a/30030/programacion-orientada-a-objetos-con-java> |

1. **CONTROL DEL DOCUMENTO (ELABORADA POR)**

|  |
| --- |
| **Ing. Luis Fernando Corredor Mora** |