Trabajo grupal o individual

Identificación del trabajo

a. **Módulo:** 3

b. **Asignatura:** Scripting

c. **RA:** Aplica estructuras de datos complejos, utilizando listas, tuplas y diccionarios, en la manipulación de datos

d. **Docente Online:** Gonzalo Cárdenas

e. **Fecha de entrega**: 24-07-2023

Identificación del/los estudiante/s

Nombre y apellido	**Carrera**
William Huera	Técnico en Data Science

Introducción

En la empresa de desarrollo de software donde trabajas, tienes un rol importante: recibir y preparar a alumnos en práctica para mejorar sus habilidades de scripting. Estos alumnos ya poseen conocimientos básicos de scripting, y tu tarea consiste en llevarlos al siguiente nivel, enseñándoles el uso de estructuras complejas de datos.
Para cumplir con esta meta, te han solicitado crear un programa de diagnóstico del estado del conocimiento de los alumnos. El objetivo es que a través de este programa, los alumnos puedan demostrar sus habilidades y comprensión de los conceptos clave relacionados con las estructuras complejas de datos.

Problemática

Desarrollo

Item1:
Analiza el uso de tuplas y diccionarios en términos de las funciones y variables definidas, en lo referente a la llamada y el resultado esperado, a partir de un script, preparando un informe que dé cuenta del análisis realizado para las estructuras. Para ello

1. Identifica el tipo de estructura de dato usado en la función suma, indicando la manera en que se usa en un mínimo de 3 líneas y un máximo de 5.
2. Identifica el tipo de estructura de dato usado en la función saluda, indicando la manera en que se usa en un mínimo de 3 líneas y un máximo de 5.
3. Identifica el tipo de estructura de dato usado en la función invierte, indicando la manera en que se usa en un mínimo de 3 líneas y un máximo de 5.
4. Identifica el tipo de estructura de dato usado en la función completa, indicando la manera en que se usa en un mínimo de 3 líneas y un máximo de 5.

Código:

```
def suma(a, b, c):
    res = a + b + c
    print('la suma es:', res)


tp = 55, 55.22, 33.33
suma(*tp)
```

```
la suma es: 143.55
```

la función recibe una tupla (estructura de datos inmutable) como paramétro, recorre cada valor con el operador de desempaquetado, que descompone la tupla en tres argumentos individuales y los suma, lo almacena en la variable “res” y finalmente imprime la respuesta.

```python
def saluda(nombre, edad, sexo, nacionalidad):
    print('Hola', nombre)
    print('Tienes', edad)
    print('eres del sexo', sexo)
    print('eres de nacionalidad', nacionalidad)

dicc = {'nombre':'Miguel C.', 'edad':22, 'sexo':'Masculino','nacionalidad':'Mexicana'}
saluda(**dicc)
```

```
Hola Miguel C.
Tienes 22
eres del sexo Masculino
eres de nacionalidad Mexicana
```

la función saluda, recibe un diccionario como parámetro, de igual forma con el operador desempaquetador, envia cada uno de las variables y finalmente las imprime.

```python
def invierte(lst):
    print(lst)
    lst_rev = lst[::-1]
    print(lst_rev)

lst = ["IPP", "Python", "Scripting", "Curso", "Material"]
invierte(lst)
```

```
['IPP', 'Python', 'Scripting', 'Curso', 'Material']
['Material', 'Curso', 'Scripting', 'Python', 'IPP']
 ~/Doc/IPP-Chile/scripting  on master +6 !1 ?1
```

la función invierte, recibe un objeto tipo lista como parámetro, y con el operador [::-1], se invierte el orden de los elementos de la lista, finalmente imprime la lista

```python
def completa(dct1, dct2):
    dct2.update(dct1)
    print(dct2)

dict1 = {'bookA':1,'bookB':2,'bookC':3}
```

```
dict2 = {'bookC':2,'bookD':4,'bookE':5}
completa(dict1,dict2)
```

```
{'bookC': 3, 'bookD': 4, 'bookE': 5, 'bookA': 1, 'bookB': 2}
~/Doc/IPP-Chile/scripting on master +6 !1 ?1
```

La función completa, recibe dos objetos de tipo diccionario como parámetros, con el operador "update" actualiza los valores del diccionario dict2 con los valores de dict1, para el caso de bookC:3 y agrega los elementos bookD:4, bookE:5 y finalmente imprime el diccionario

Git: https://github.com/whuera/scripting/blob/master/modulo3/m3-t1.py

Item2:
Determina las variables de entrada y salida, las transformaciones o conversiones solicitadas en cada punto para los scripts descritos a continuación, de manera de explicar a los practicantes lo que realiza cada uno de los scripts descritos:

```
37  mylist = list((1,2,3,4))
38  print(mylist)
39  print(type(mylist))
40
41  mytuple = tuple(mylist)
42  print(mytuple)
43  print(type(mytuple))
```

```
46  lista1 = list(('nombre', 'edad', 'direccion'))
47  lista2 = list(('Juan', '30', 'Valparaiso'))
48  clave_tupla = tuple(lista1)
49  valor_tupla = tuple(lista2)
50
51  dicc = dict(zip(clave_tupla, valor_tupla))
52  print(dicc)
53  print(type(dicc))
```

1. Determina, para el primer script, la identificación y descripción de los valores de las variables iniciales y de las salidas.
2. Determina, para el primer script, las transformaciones o conversiones entre los distintos tipos de datos.
3. Determina, para el segundo script, la identificación y descripción de los valores de las variables iniciales y de las salidas.

4. Determina, para el segundo script, las transformaciones o conversiones entre los distintos tipos de datos

Código:

```
## Script 1
mylist = list((1,2,3,4))
print(mylist)
print(type(mylist))

mytuple = tuple(mylist)
print(mytuple)
print(type(mytuple))

## Script 2

lista1 = list(('nombre', 'edad', 'direccion'))
lista2 = list(('Juan','30','Valparaiso'))

clave_tupla = tuple(lista1)
valor_tupla = tuple(lista2)

dicc = dict(zip(clave_tupla, valor_tupla))
print(dicc)
print(type(dicc))
```

```
(1, 2, 3, 4)
<class 'tuple'>
{'nombre': 'Juan', 'edad': '30', 'direccion': 'Valparaiso'}
<class 'dict'>
~/Doc/IPP-Chile/scripting/modulo3 on master !3
```

Script 1:

Parámetros de entrada:	parámetro objeto tupla (1,2,3,4)
Transformaciones:	el objeto tupla (1,2,3,4) se convierte en una lista usando la función list() y se asigna a la variable mylist
Salida:	impresión de las variables mylist (<class 'list'>), mytuple (<class 'tuple'>)

Objetivo:	el código muestra como convertir una tupla y esa nueva lista en una nueva tupla

GitHub: https://github.com/whuera/scripting/blob/master/modulo3/m3-t1.py

Item3:
Aplica los conceptos aprendidos, desarrollando un ejercicio mediante la creación de un script que identifique la estructura de datos idónea para el ingreso de notas de 10 alumnos, generando 2 fórmulas: una que calcule el promedio de notas y otra que calcule la desviación estándar.
Se evaluará lo siguiente:

1. Determinar la estructura de datos adecuada, con la justificación del caso, en un mínimo de 3 líneas y un máximo de 8 líneas como comentario dentro del código fuente.
2. Generar la función solicitada de cálculo del promedio.
3. Generar la función solicitada de cálculo de desviación estándar.
4. Crear el script con la sección principal de ingreso de datos, y despliegue por pantalla de los resultados

Código:

```
# script ingreso de notas de alumnos
# descripcion: script para ingreso de notas de alumnos con estructura clave-valor: {nombre, notas}
import math


# función para calcular el promedio
def calcularPromedio(notas):
    suma = sum(notas)
    return suma / len(notas)


# funcion para desviacion standar
def calcularDesviacionStandar(notas):
    # calculo del promedio
    promedio = calcularPromedio(notas)
    varianza = sum((nota - promedio)**2 for nota in notas) /len(notas)
    # calculo de raiz cuadrada de la varianza para obtener desviacion standar
    return math.sqrt(varianza)
```

```python
# funcion main
def main():
    # input datos de alumnos
    alumnos = {}
    for i in range(1,11):
        nombre = input(f"Ingrese el nombre del alumno{i}: ")
        notas = list(map(float, input(f"Ingrese las notas del alumno {nombre}, separadas por espacio: ").split()))
        # asignar notas al key del alumno
        alumnos[nombre] = notas

    for nombre, notas in alumnos.items():
        promedio = calcularPromedio(notas)
        desviacion_standar = calcularDesviacionStandar(notas)
        print(f"Alumno: {nombre}")
        print(f"Promedio: {promedio}")
        print(f"Desviacion standar: {desviacion_standar}")

if __name__=="__main__":
    main()
```

```
Alumno: e
Promedio: 83.0
Desviacion standar: 16.0
Alumno: f
Promedio: 78.0
Desviacion standar: 11.0
Alumno: g
Promedio: 88.0
Desviacion standar: 11.0
Alumno: h
Promedio: 76.5
Desviacion standar: 10.5
Alumno: i
Promedio: 89.5
Desviacion standar: 0.5
Alumno: j
Promedio: 89.5
Desviacion standar: 10.5
~/Doc/I/scripting/modulo3  on master !3 ?1
```

Para este script utilizamos lo siguiente:

estructura de datos:	diccionario donde para ingresar nombre alumno y notas {Nombre, notas}, dado que al utilizar el diccionario nos facilita el acceso y la manipulación de datos.
Función: `def calcularPromedio(notas):`	Utilizamos la función para calcular el promedio, donde le pasamos como parámetro el objeto notas
Función: `def calcularDesviacionStandar(notas):`	Utilizamos la función para calcular el desviación standar, donde le pasamos como parámetro el objeto notas y reutilizamos la primera función calcularPromedio y calculamos la varianza

Script main:
`def main():`
`  # input datos de alumnos`
`  alumnos = {}`
`  for i in range(1,11):`
`  nombre = input(f"Ingrese el nombre del alumno{i}: ")`
`  notas = list(map(float, input(f"Ingrese las notas del alumno {nombre}, separadas por espacio: ").split()))`
`  # asignar notas al key del alumno`
`  alumnos[nombre] = notas`
`  for nombre, notas in alumnos.items():`
`  promedio = calcularPromedio(notas)`
`  desviacion_standar = calcularDesviacionStandar(notas)`
`  print(f"Alumno: {nombre}")`
`  print(f"Promedio: {promedio}")`
`  print(f"Desviacion standar: {desviacion_standar}")`
	En esta sección del código, realizamos la lectura desde consola de los nombres de los alumnos y las notas, separadas por espacio, luego llamamos a las funcionesde calcularPromedio y calcularDesviacionStandar

Utilizamos la función calc

GitHub: https://github.com/whuera/scripting/blob/master/modulo3/m3-t1-alumnos.py

Conclusión

Utilizar funciones y diccionarios en Python ofrece numerosas ventajas que facilitan el desarrollo y mantenimiento de programas.

Ventajas de utilizar funciones:

1. Reutilización de código: Las funciones te permiten encapsular bloques de código en tareas específicas y reutilizarlos en diferentes partes del programa. Esto fomenta la modularidad y evita la duplicación innecesaria de código.

2. Mantenibilidad: Al dividir el código en funciones más pequeñas y con propósitos específicos, el código general se vuelve más fácil de entender y mantener. Las actualizaciones y correcciones pueden aplicarse de manera más localizada.

3. Abstracción: Las funciones ocultan los detalles de implementación al usuario, lo que permite trabajar a un nivel más alto de abstracción. Esto hace que el código sea más fácil de usar y comprender.

4. Legibilidad: Al utilizar funciones con nombres descriptivos, el código se vuelve más legible y autodocumentado. Esto es especialmente útil cuando otros desarrolladores revisan o colaboran en el proyecto.

5. Depuración y pruebas: Al dividir el código en funciones, es más fácil aislar problemas y realizar pruebas unitarias, lo que simplifica la tarea de encontrar y corregir errores.

Ventajas de utilizar diccionarios:

1. Estructura clave-valor: Los diccionarios son ideales para almacenar y acceder a datos en función de claves únicas. Esta estructura es muy útil para organizar información relacionada.

2. Búsqueda eficiente: Los diccionarios en Python utilizan tablas hash, lo que permite búsquedas rápidas y eficientes por clave. Esto hace que sea muy útil para trabajar con grandes conjuntos de datos.

3. Flexibilidad: Los diccionarios pueden contener cualquier tipo de datos, incluso otros diccionarios, listas o funciones, lo que brinda una gran flexibilidad en la representación de datos complejos.

4. No requiere índices numéricos: A diferencia de las listas que se indexan mediante números, los diccionarios utilizan claves, lo que los hace más intuitivos y menos propensos a errores de índice.

5. Combinación con iteradores: Los diccionarios en Python son compatibles con iteradores, lo que permite recorrer fácilmente sus elementos para realizar operaciones en ellos.

En conclusión, el uso adecuado de funciones y diccionarios en Python contribuye a un código más estructurado, modular, reutilizable y fácil de mantener

Bibliografía

(w3schools, 2020)
Documentación guía, material de apoyo del módulo 3