**Módulo 16:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **2da Revisión de cursada** | | | | |
| **tipos de objetos** | **Funciones built in** | **operadores** | | |
| class booleans  class list, class set, class dict | Operaciones  ternarias con if-else por comprension.if-else-for | ternarios  one line comprehension | | |
|  | **Funciones aleatorias** |  | | |
| class float (entre 0 y 1)  class int  depende de la lista | random  import random | random.random ()  random.randrange ()  random.randint ()  random.choise () | | |
| **tipos de objetos** | **Funciones propias** | **operadores** | | |
| class functions | def  return  (retorno único y empaquetados y desempaquetados)  Parámetros y operadores  Argumentos posicionales  Argumentos por omisión  Recursividad  Anotaciones en funciones  Docstrings  lambda | def nombre (parametros)  return  return dato  return coleccion  = , \* , \*\*, /  param2=x, param1=y  x=”valor por defecto”  """ help """  anónimas pero con nombre | | |
| **tipos de objetos** | **Funciones de orden superior.** | **operadores** | | |
| class function  ------------------------------  class map  class filter  class reduce | def funcion():  def interna():  pass  return  interna()  return  zip ()  enumerate ()  ------------------------------  map ()  filter ()  reduce ()  \*import functoolse | ------------------------------  map (función, iterable)  filter (función, iterable)  reduce (función, iterable)  reduce (función, iterable, [initializer]) | | |
|  | **Manejo de excepciones** |  | | |
|  | try  except  else  finally  raiser | Exception  Excepciones propias | | |
|  | **Manejo de información en archivos externos** |  | | |
| txt  json  pickle | open  close  with | Serialización  Deserialización | | |
| read | rt o r | rb |
| write | wt o w | ab |
| append | at o a | ab |
| create empty | xt o x | xb |
| + | | |
|  | **Manejo de archivos y directorios (carpetas)** |  | | |
|  | os (operative system)  import os | os.path.isfile ()  os.path.isdir ()  os.mkdir ()  os.rename ()  os.rmdir ()  os.getcwd()  os.chdir()  os.remove ()  os.listdir() | | |
|  | **Ingreso de informacion de**  **web - apis** |  | | |
|  | request   * pip install requests   import request | url  apiKey  token  endpoint  headers | | |
|  | **Manejo de fechas y horas.**  **ISO** |  | | |
|  | datetime  import datetime  time  import time | now  strftime  strptime  deltatime | | |
|  | **Graphical User Interface** |  | | |
| (GUI) | tkinter  tk y ttk  widgets | windows,  buttons,  labels,  entry | | |

·

|  |
| --- |
| **Conceptos clave relacionados con los objetos en Python:**  **módulo:** Una Módulo es una plantilla o definición que describe las propiedades y comportamientos que tendrán los objetos creados a partir de ella. Define los atributos (variables) y los métodos (funciones) que estarán disponibles en los objetos de esa módulo.  **Instancia:** Una instancia es un objeto específico creado a partir de una módulo. Representa una ocurrencia única de esa Módulo y tiene sus propios valores para los atributos definidos en la módulo.  **Atributos:** Los atributos son las variables que pertenecen a un objeto. Pueden ser variables de instancia (definidas dentro de un método) o variables de Módulo (definidas en la Módulo pero fuera de los métodos). Cada instancia de una Módulo puede tener diferentes valores para los atributos.  **Métodos:** Los métodos son las funciones asociadas a una módulo. Pueden realizar operaciones en los datos del objeto y pueden acceder y modificar los atributos. Los métodos pueden ser llamados en las instancias de la Módulo y actúan en contexto a los datos específicos de la instancia.  **Encapsulación:** La encapsulación es un principio de la POO que se refiere a la agrupación de datos y métodos en un objeto y el ocultamiento de los detalles internos. Los objetos encapsulan sus datos y proporcionan interfaces (métodos) para interactuar con ellos, ocultando los detalles internos de implementación.  **Herencia:** La herencia es un concepto que permite crear nuevas módulos basadas en módulos existentes. Permite que una Módulo herede los atributos y métodos de otra módulo, extendiéndolos o modificándolos según sea necesario. La herencia permite reutilizar código y establecer relaciones jerárquicas entre las módulos.  **Polimorfismo:** El polimorfismo permite que los objetos de diferentes módulos respondan de manera diferente al mismo método. Esto significa que un mismo método puede tener diferentes implementaciones dependiendo del objeto en el que se llame. El polimorfismo promueve la flexibilidad y la reutilización de código. |

### ·