Fundamentos de informática

Atajos	Comandos
ctrl + s ⟶ guardar	ls ⟶ lista de archivo de directorio
ctrl + j ⟶ terminal	pwd ⟶ path del dir en que estoy
ctrl + d ⟶ selección múltiple	cd “path”⟶ cambio de dir
ctrl + a ⟶ seleccionar todo	cd .. ⟶ un directorio para atrás
	cd ⟶ root
	os ⟶ biblioteca que permite mover / ejecutar terminales
	sys ⟶ biblioteca que toma parámetros entre terminales
	read (lines) ⟶ devuelve lista
	git add .⟶ preparar para sacar foto git commit -m“mensaje” ⟶ sacar foto git push ⟶ lo sube a la web *git pull ⟶ bajar los cambios*
	git clone ⟶ bajarme un proyecto de cero
	os.mdkir(ruta(ej; “../fundamentos/practica1”))
	os-chdir(ruta) ⟶ me muevo ahí
	EN BASH: ls -l ⟶ muestra lista
	os.listdir() ⟶ nombre de arch en listas

Teórica
Clase 2

```
mi_script.py
print ("mora")
"""comentario"""
""".py SIEMPRE QUE HAGA UN ARCHIVO PHYTON"""
```

Descargar:

- ~~Visual Code~~
- ~~Gitbash~~
- ~~Anaconda (Python)~~
- ~~Python~~

(Github.com/AJVelezRueda)

Fundamentos_de_informatica/Práctica_de_introducción_a_Python_Parte1.md at master · AJVelezRueda/Fundamentos_de_informatica · GitHub → práctica

Script

- Es un archivo
- Para ejecutarlo tengo que estar en su carpeta
- Se ejecuta llamando a un intérprete en la terminal
- Si están los >>>, ya estoy dentro de python y no puedo ejecutar.
- **Comando shebang**
  - tienen que tener: **#!/bin/python3**
  - Estructura correcta:
    - ```
      def main():
          print("hola mundo")


      if __name__ == "__main__":
          main()
      ```
  - Si este archivo es el main, ejecuta esto.

ctrl+j → terminal

terminal → donde estoy parada
      ```

Comandos terminal

- **ls** → lista archivos de un directorio

  Armar el código:

- Visual code
- armo archivo .py
- agregar código
- terminal
- En la terminal en donde esta el script
  - ls → debería estar el nombre del script en “Length Name”
- PS C:\Users\4e45> **python tab**
  - PS C:\Users\4e45> **python** .\mi_script.py

Si estás en el lugar equivocado:

Comando cd → change directory

- Archivo
  - Copy
  - cd
    - copio y prgp
- Comando **Pwd** → en donde estas parado

Biblioteca

- Directorio/carpeta con scripts con funciones de python que puedo usar.
- Como instalarla en Python: gestor de paquetes → **pip**
- Para instalar un paquete: pip → instalar
  - SIEMPRE chequear que no esté instalada antes
    - >>> import **nombre de la biblioteca**

Teórica
Clase 2

Workshops/[ES]Scripting.md at master · WomenBioinfoDataScLA/Workshops · GitHub

os ⟶ operative system

Biblioteca que permite mover/ejecutar cosas

sys ⟶ biblioteca que toma parámetros entre terminales

Tarea:

¡Desafío final! Creá un script `swap.py` que tome dos nombres de archivo y renombre al primero con el nombre del segundo, y al segundo lo renombre con el nombre del primero. Ejemplo:

```
$ cat hola.txt
hola
$ cat chau.txt
chau
$ ./swap.py hola.txt chau.txt
$ cat hola.txt
chau
$ cat chau.txt
hola
```

Manipulación de archivos

Fundamentos_de_informatica/Manipulación_de_archivos.md at master · AJVelezRueda/Fundamentos_de_informatica · GitHub

Archivos de lectura:

- Se puede leer todo en una línea o línea por línea.
- **open(path**_al_articulo, **modo)** ⟶ abre archivo. **close** ⟶ cierro el archivo
    - Tiene que ser un **STR** con la dirección a mi archivo
    - modo ⟶ la forma en que quiero ejecutarlo
        - r ⟶ de lectura
        - w ⟶ escritura. Si el archivo no existe crea uno, si es que si, sobreescribe sobre eso borrando lo anterior.
        - r+ ⟶ lectura y escritura

        - a → agrega al final del archivo
        - **STR " "**
- Otra forma para abrir archivo de forma segura:
    - with open(path_al_archivo, modo) as miarch:

```
#Aquí van las líneas de procesamiento del archivo
```

Como crear un script

1. Genero un archivo ".py"
2. Agrego el codigo a mi archivo (del paso 1)
3. Guardar los cambios
4. Abro la terminal
5. (ls) Verifico que mi script está en la carpeta
6. Escribo en la terminal de bash: python [nombre del archivo]

Teórica
Clase 3

- [ ] ¿Como se usa os.path.exists?

Cerrando la clase pasada

- readlines() → lee línea a línea

```
1   arch = open ("Mi_nombre.txt", "r")
2   print(arch)
3   print(arch.readlines())
```

Mi_nombre.txt

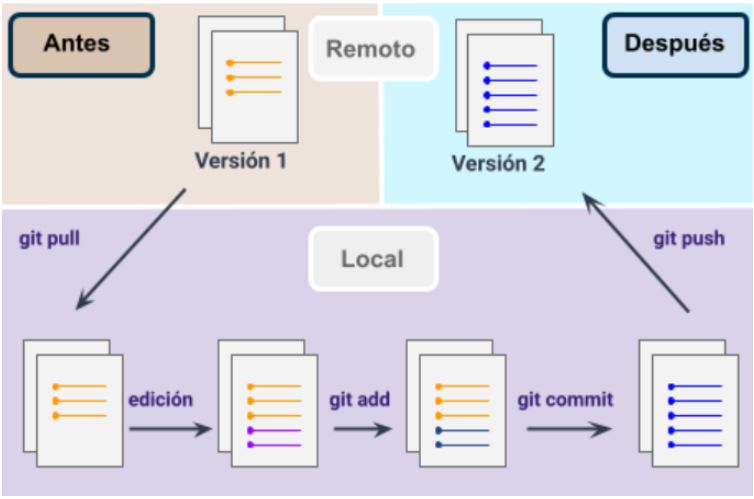
```
1   Mora Hambo
2   Tuca
```

```
PS D:\Mora\Informatica 2023\clases> python .\3_clase_teorica.py
<_io.TextIOWrapper name='Mi_nombre.txt' mode='r' encoding='cp1252'>
['Mora Hambo\n', 'Tuca']
```

- Ruta absoluta: home/usuario/doc/Mi_nombre.txt → esto lo hago si el archivo que llama está en otro directorio.
    - Asumo un usuario, cuando lo prueba otro rompe.
- Ruta relativa: "donde estoy parada"./doc/Mi_nombre.txt
    - No asume un usuario → mejor para los parciales.

- [ ] Antes de comenzar → Control de versiones

Sistema de Control de Versiones



- Git → software para compartir códigos.
- Se hace desde la terminal.
- git add → preparase para "sacar la foto"
- git commit → "saca la foto"
    - Anota metadata y cambios.
        - metadata; info asociada.
- git commit -m "add script clase_3"

Para subirlo

1. Tener un usuario de github
2. Vincular con la compu
3. Subirlo a la web a través de: git push
    - Hacerlo SIEMPRE antes de cerrar.
4. Bajar los cambios que hice a otra compu: git pull
    - Hacerlo SIEMPRE antes de arrancar.

1. Perfil
2. Repositorios
3. New
4.

Práctica
Clase 2

Fundamentos_de_informatica/Teoría_suplementaria.md at master · AJVelezRueda/Fundamentos_de_informatica · GitHub

Práctica
Clase 3

Fundamentos_de_informatica/Expresiones_regulares.md at master · AJVelezRueda/Fundamentos_de_informatica · GitHub

Teórica
Clase 4

10/04

Fundamentos_de_informatica/Expresiones_regulares.md at master · AJVelezRueda/Fundamentos_de_informatica · GitHub

Secuencia de escape	representa
\n	salto de línea
\t	Tab o cambio de
\s	espacio
'	Comillas simple
"	Comillas dobles

Metacaracter	Significado
^	Inicio de línea
$	Fin de linea
\A	Inicio de texto
\Z	Fin de texto
.	Coincide con cualquier caracter en una línea dada

Metacaracter	Significado
\w	Caracter alfanumércio
\W	Caracter NO alfanumércio
\d	Caracter numércio
\D	Caracter NO numércio
\s	Un espacio, de cualquier tipo (\t\n\r\f)
\S	Cualquier caracter que no sea un espacio

Metacaracter	Significado
*	Cero o más: todas las ocurrencias de un dado substring
+	Una o más ocurrencias del patrón
?	Cero o una
{n}	Exactamente n veces
{n,m}	Por lo menos n pero no más de m veces.

Práctica
Clase 5

12/04

Fundamentos_de_informatica/Anexo_Expresiones_Regulares.md at master · AJVelezRueda/Fundamentos_de_informatica · GitHub

Expresiones regulares

Patrones

- Lo que voy a querer buscar.
- Se puede escribir como patron= “ ” → y después ponerlo en el () del re.
- o directo en el re. como; re.función (r “ ”,)

Evaluar resultados de búsqueda

- Funciones; search, findall, sub

- re.search → método de biblioteca re.
    - Obtiene string buscado y posición donde se encuentra.
    - Ej;

```
import re
if re.search(patron, texto) is not None:
        bloque de código
else:
        bloque de código
```

- re.findall → se obtienen todas las apariciones del patrón.
- re.sub → reemplazas todas las ocurrencias del patrón, por otro en un string.

```
def reemplazar_por_barra (string1):
    string_nuevo = re.sub(r"[ _:]", "|", string1)
    return string_nuevo
```

    - o si escribo el patrón primero:

```
def reemplazar_por_barra (string1):
    patron = "[ _:]"
    string_nuevo = re.sub(patron, "|", string1)
    return string_nuevo
```

- .group() → mostrar el rtado de una busqueda.
- None → no es nada
    - is not None → si es algo

Unión de rangos

- Poner el inicio y el fin del rango entre corchetes y separados por un guión ([a-z] para todas las letras en minúscula)
- ^ ([^a-z] → ignorar rango

Palabras

- \b el cual delimita caracteres alfanuméricos de otros que no lo son.
- Separa caracteres
- Ej;

```
>>> import re
>>> texto = "Lorem ipsum dolor sit amet, consectetur ipsum elit. Amet sit
amet."
```

```
>>> patron = r"\bipsum\b"
>>> re.sub(patron, "lapsus", texto)
"Lorem lapsus dolor sit amet, consectetur lapsus elit. Amet sit amet."
```

Apuntes:

- Todo lo que no sea 0 es True → (booleano)
- No encuentra rtado: bool(None) → Falso
- “he{}” → solo busca esa palabra entre esos valores
- $ → donde termina la línea
- find all → devuelve lista cada vez que encuentra un patron
- . → cualquier cosa
- * → cero o una más veces
- ? →
    - ? después de un caracter → cero o una vez
    - ? después de un modificador → todos los matches
- def get_substr(string):

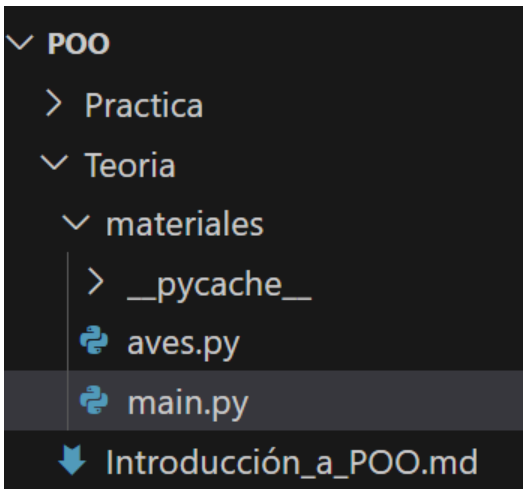
        return re.findal1("[@|&](.*?)[@|&]", string)
-

Teórica
Clase 5

14/04

Programación orientada a objetos

- Programación Orientada a Objetos
- (Abrir repositorio)



- Def: Entidad computacional que me permite interactuar.

Características

- Me puedo comunicar, entiende mensajes, tiene referencias internas y es consciente de eso.
- Pueden tener distintos estados basales.
  - Aunque haya distintos estados basales, los objetos pueden entender los mismos **métodos** (métodos == mensaje)
- Pueden entender los mismos mensajes aunque sean de otra clase.
  - Objeto dentro de una clase → **instancia.**
- Conjunto de mensajes/métodos que entiende la clase → **interfaz**
  - Las distintas clases tienen interfaces parecidas
- Objetos que comparten interfaz y hay una tercer clase que los usa→ **polimórficos**
  - Puede ser parcial o total.
  - Ellos no saben que son polimórficos. Se necesita de un observador/otro actor para verlo.
- Estado → se da por el conjunto de atributos
  - Puede cambiar o modificarse con el tiempo. Por ejemplo luego de dar órdenes (comer, volar, etc.)

"Diccionario"

- Clase: class **Clase**
- Atributo: def __init__ (self, parametro, parametro)
  - **el atributo es: self.atributo**

- Método: def **metodo** (self, parámetros(o no)): → sería la función
  return o self. o self
- Estado: conjunto de atributos.
- Metodos == mensajes
- Instancia: objeto dentro de clase.
- Interfaz: Conjunto de mensajes/métodos que entiende la clase
- Polimorfismo: Objetos que comparten interfaz y hay una tercer clase que los usa

Programarlo

Parte 1: defino clase

- Se define con la palabra reservada **class**
  - Los nombres se ponen en mayúscula

```
class Golondrina:
  def __init__(self, energia):
    self.energia = energia

  def comer_alpiste(self, gramos):
    self.energia += 4 * gramos

  def volar_en_circulos(self):
    self.volar(0)

  def volar(self, kms):
    self.energia -= 10 + kms
```

- __init__ → para crear el objeto
- self → el objeto mismo → se usa siempre
  - En este caso requiere monto inicial de energía para crearse

Mumuki

- Podemos usar funciones que ya conocemos.
  - Ej; en vez de usar str.upper;
    - “objeto”.upper()
  - En vez de list.append(numeros, 32);
    - numeros.append(32)
    - Donde numeros es el objeto, append la función y 32 el nro que quiero agregar.

- str.startswith, str.endswith, str.strip, str.lower, list.remove

- Otro ejemplo de que como programar una clase:

```
class Pintura:
 def __init__(self, un_artista, una_tecnica, un_ancho, un_alto):
     self.artista = un_artista
     self.tecnica = una_tecnica
     self.ancho = un_ancho
     self.alto = un_alto
```

- **Raise Exception(“”):**

Herencia

- = ⟶ no hay return ⟶ modifica el valor
- == ⟶ hay return ⟶ T o F
- Ejemplo de heredar:

```
class Ave:
 def __init__(self, energia):
  self.energia = energia
 def volar(self):
  self.energia -= 20

class Condor(Ave):
 def dormir(self,minutos):
  self.energia += minutos * 3
```

  - Condor hereda de ave que self.energia=energia y lo de volar. Condor es un ave.
- **Clase abstracta**
  - Clase Madre.
  - Proveen comportamiento a sus subclases.
- **Clase concreta**
  - Crean instancias
- **Super** ⟶ se utiliza en las subclases.
  - Evalúa el método con el mismo nombre de su superclase y además lo que se está redefiniendo. ⟶

```
class Perro:
  def cruzarse_con(self, un_perro):
    self.mover_la_cola()
    self.oler(un_perro)

class PerroCascarrabias(Perro):
  def cruzarse_con(self, un_perro):
    super().cruzarse_con(un_perro)
    self.ladrar()
```

- Si en la clase abstracta puse algo, <u>se puede modificar</u> en la subclase. Redefinir el método. Ejemplo:

```
class MedioDeTransporte():
  def __init__(self, un_combustible):
    self.combustible = un_combustible

  def cargar_combustible(self, cantidad_combustible):
    self.combustible += cantidad_combustible

  def entran_personas (self, cantidad_personas):
    return cantidad_personas <= self.maximo_personas()
```

vs

```
class Colectivo(MedioDeTransporte):
  def __init__(self):
    self.pasajeros = 0
    self.combustible = 100
  def entran_personas(self, pasajeros):
    return self
```

- En colectivo se eliminó el “maximo_de_personas”
- Se inicializa el atributo “self.pasajeros”
- Se inicializa self.combustible con 100.

Colecciones

- Se puede usar self y append juntos. Ej:

```
class Biblioteca:
  def __init__(self):
    self.libros = []
  def agregar_libro(self, libro):
    self.libros.append(libro)
```

- Listas por comprensión → permite hacer mapeos y filtrados de la lista.
- **Sets:**
  - Parecidos a las listas.
  - No tienen elementos repetidos.
  - Sus elementos no están ordenados.
- Tupla:
  - Retornar más de una cosa y sabemos exactamente cuantas.
  - Ej: que me devuelva las primeras letras del nombre, segundo y apellido.

```
class Persona:
  def __init__(self, un_nombre, un_segundo_nombre, un_apellido):
    self.nombre = un_nombre
    self.segundo_nombre = un_segundo_nombre
    self.apellido = un_apellido

  def iniciales(self):
    return (self.nombre[0], self.segundo_nombre[0], self.apellido[0])
```

    - Estoy pidiendo la posición 0.
  - También se puede usar + - * /
    - def precios_con_iva (self):

```
        return self.precio_minorista * 1.21, self.precio_mayorista *1.21
```