PYTHON PARA PRINCIPIANTES

CLASE 2

IGNACIO DEZA

MACHETE DE LA CLASE PASADA

- **print()**, **(+, -, *, /, %,**)**, **str()**, **float()**, **int()**, **round()**
- **len()**, **(\')**, **(\\)**, **(\n)**, **(\t)**
- {:**X <^> padding .trunc**} .format(**'string'**)
- {**:+X padding .decimal df**} .format(**número**)
- **tuples** (a,b,c,d) y **listas** [a,b,c,d]
- **min()**, **max()**, **sorted()**, **sum()**, **type()**
- Uso de **A[n]**, **A[n:m]**, **A[n:m:l]**
- Sólo listas: **A.append(n)** y **del(A[n])**
- El uso de **for()** ,y **if()**, **elif()** y **else()**. Los (**:**) y el **tab** (4 espacios)
- **range()** crea listas y **enumerate()** crea lista con índice.

CONTENIDOS DE LA CLASE DE HOY

- Para acceder a los PDF de las clases mandar mail a: **ignacio.deza@atlantida.edu.ar**
- Esta será la última clase **sin objetos**
- Hoy redondearemos las **bases** de python
- También comenzaremos con una intro a **módulos**
- Como sin práctica no se aprende un lenguaje, al final de la clase habrá un problema para entregar (la próxima clase).

QUE VAMOS A HACER HOY

- Convertir un numero de cifras a palabras. Ej: 1234 —> “mil doscientos treinta y cuatro”.
- Vamos a escribir un programa en un archivo y ejecutarlo
- Vamos a aprender las funciones de control de flujo que nos faltan **while**, **break**, **continue**, **pass**
- Vamos a aprender las estructuras de datos que nos faltan: sets, diccionarios y algunos métodos que nos faltaron
- Vamos a aprender a hacer **funciones** propias y **módulos** propios
- Y una pequeña introducción al manejo de **errores** y **excepciones**.

PROGRAMA1

- Cómo hacer un programa que escriba con palabras un número?
- Uso de indices
- Uso de tuples
- Uso the While y el loop infinito
- Uso de ctrl-C para cortar los loop infinitos
- ¿Qué se puede mejorar?
  - Hay un montón de cosas que hace **MAL**

PROGRAMA1_V2

- Atacamos los **casos particulares** con “if”
- No hagan TODO con “if” porque el programa será gigante (e incomprensible)
- Casos:
  - Numero no es de 4 cifras
  - Palabras que **cambian** (“quince”, “veintiseis”)
  - Palabras que usan “y” y otras que usan “i”
  - El numero cien (no ciento)
- Uso de **banderas**
- ¿Qué se puede mejorar? El programa se hace muy complejo

PROGRAMA1_V3

- Vamos a agarrar la el programa anterior y meterlo en una función:
- Para eso se usa **def** nombreDeFunción(Argumentos):
- **Ya no tenemos que usar banderas**
- Comando **return**
- Emparentado con los comandos **break** y **continue**
- ¿Qué se puede mejorar?
  - Una vez que la función anda bien no queremos tocarla más

PROGRAMA1_V4

- Vamos a poner la función anterior en un archivo aparte
- Para eso usamos **import** que es la forma de importar todos los modelos de python
- De repente el programa se vuelve muy **simple**
- Ya no nos tenemos que preocupar por lo que ya funciona bien. Eso se llama **reutilizar código**
- ¿Qué se puede mejorar?
  - Que el programa no se "cuelgue" cuando escribimos una palabra o algo, sino que sepa manejar ese tipo de errores.

PROGRAMA1_V5

```
Ingrese un número del 0 al 9999: d
Traceback (most recent call last):
  File "programa1_v3.py", line 76, in <module>
    respuesta = numALetras(int(numeroStr))
ValueError: invalid literal for int() with base 10: 'd'
```

- Vemos que el error es ValueError. Así que podemos capturar ese error con **try** y **except**
- Así los programas pueden decidir que pasa cuando pasa algo inesperado (una excepción)
- Lista de excepciones: https://docs.python.org/3/library/exceptions.html

EJERCICIO 1

- Escribí un programa que funciona igual al que estuvimos trabajando pero con la diferencia que no tiene limite de 4 cifras.
- EJERCICIO: traducir el programa **al inglés** de forma de que haga lo mismo pero en inglés.
- Cosas a tener en cuenta: “eleven”, “twelve”, Thir**teen**”
- Lo que nosotros llamamos mil millones ellos lo llaman **billon**, nuestro billón es un **trillion** etc..
- Cualquier cosa busquen la numeración en wikipedia
- Al final de la clase puedo responder algunas preguntas de esto.

SETS

- Un set es una lista de elementos no repetidos

```
>>> a = "murcielago"
>>> len(a)
10
>>> b = set(a)
>>> b
{'e', 'r', 'o', 'm', 'l', 'a', 'i', 'g', 'u', 'c'}
>>> len(b)
10
>>> c = "constantinopla"
>>> len(c)
14
>>> d = set(c)
>>> d
{'p', 's', 't', 'o', 'a', 'l', 'i', 'n', 'c'}
>>> len(d)
9
>>>
```

SETS

- Un set se puede ordenar:

```
>>> sorted(d)
['a', 'c', 'i', 'l', 'n', 'o', 'p', 's', 't']
>>> A= [x for x in "constantinopla" if x not in "murcielago"]
>>> A
['n', 's', 't', 'n', 't', 'n', 'p']
>>> B= {x for x in "constantinopla" if x not in "murcielago"}
>>> B
{'n', 's', 'p', 't'}
>>> sorted(B)
['n', 'p', 's', 't']
>>>
```

- También vemos una **notación** muy útil para crear **tuples** o **listas** o **sets** a partir de un for, con condiciones

DICCIONARIOS

- La última estructura de datos se llama **diccionario**

```
>>> tel = {"Juan":2234238794,
... "Pedro":1183961486,
... "Pablo":221345869,
... "Maria":294454675}
>>> tel["Juan"]
2234238794
>>> tel["juan"]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'juan'
>>> for nombre,telefono in tel.items():
...     print(nombre,telefono)
...
Juan 2234238794
Pedro 1183961486
Pablo 221345869
Maria 294454675
>>>
```

DICCIONARIOS

- Es lo que se conoce también como **mapa** o **hash table** y sirve para hacer búsquedas o acomodar variables en espacios donde de otra forma habría muchos espacios en blanco.
- También para asociar variable no numéricas
- len() funciona y otras propiedades más
-

```
>>> for nombre,telefono in tel.items():
...     if(nombre[0]=="P"):
...         print(nombre+": "+str(telefono))
...
Pedro: 1183961486
Pablo: 221345869
>>>
```

CONCLUSIONES

- Con lo aprendido se pueden hacer programas muy útiles e interesantes
- Faltan usar los módulos estándar de python
- Falta usar programación orientada a objetos
- ¡¡Faltan todas las aplicaciones!!
- Pero si entendieron más del 80% de la clase de hoy

- **¡FELICIDADES YA SABEN PYTHON!**

ESO ES TODO POR HOY

¡HASTA LA PRÓXIMA CLASE!

UNIVERSIDAD ATLÁNTIDA ARGENTINA | UAA INGENIERÍA

MACHETE

- **str()**, **float()**, **int()**, **round()**, **set()**, **tuple()**, **list()**
- **min()**, **max()**, **sorted()**, **sum()**, **type(), len()**
- Uso de **A[n]**, **A[n:m]**, **A[n:m:l], reverse**
- No tuples: **A.append(n)** y **del(A[n)**
- **tuples** (a,b,c,d), **listas** [a,b,c,d], **sets** {a,b,c,d}, **diccionarios** {a:b,c:d,e:f,g:h}
- **min()**, **max()**, **sorted()**, **sum()**, **type(),**
- No tuplas: **A.append(n)** y **del(A[n])**
- El uso de **for()** ,y **if()**, **elif()** y **else(), break**, **continue**, **pass**, **return**
- **range()** ,**enumerate()**, items()
- Cosas así. Si A=[1, 2, 3, 4] entonces **AA= [x**2 for x in A if x<3]**