TEMA 1.

Python (1991) es de tipo script, interpretado, no necesario compilar, no necesita declaración de variables y tiene multitud de herramientas y bibliotecas. Sensible a mayúsculas y minúsculas, con recolector de basura basado en referencias.
Los tipos de datos son dinámicos, se pueden conocer con type(). Se pueden reutilizar resultados con _.
Hay que tener cuidado con las operaciones básicas, el redondeo de números a potencias de 2 hace que los resultados puedan no ser tan exactos como se esperaría:

Ej: 0.3-(0.1+0.1+0.1)!=0

Las variables se crean cuando se asignan, una asignación entre variables hace que referencien al mismo sitio, no realizan copias. (Con id(var) podemos ver su ID única).
Podemos hacer multiasignación e intercambio de manera sencilla

x,y=2,3 x=y=2 x,y=y,x

Convención de nombres:

- Funciones, métodos y atributos, letras minúsculas y _
- Constantes con mayúsculas
- Clases comienzan por mayúscula

Las funciones se declaran con def, pueden usarse con distintos tipos de datos sin modificarlas, si no hay return devuelve none por defecto. Se permite hacer return multiples return a,b. Ademas de esto se permite el uso de alias a=función. El uso de parámetros por defecto se hace usando par=x en la declaración de la función.

LAMBDA:
Funciones especiales lambda, funciones en línea básicas ejemplo:

suma=lambda x,y:x+y

Lambda if/else

ejm=lambda n:1 if n==1 of n==0 else -1

OPERADOR []:
Cad= “Hola mundo”

Cad[0:3] >> “Hol”
Cad[3:-1] >> “a mund”
Cad[-1:0:-1] >> “odmun alo”
Cad[:] >> “Hola mundo”
Cad[::-1] >> “odmun aloH”

Bucles:

for x in obj
for x in (1,2,3,4)
for x,y in zip(range(10),range(100,120) >> 0,100 1,101…
for x,y,z in zip (range(20),range(12),(21,45,16)) >> 0,0,21 1,1,45 2,2,16 end.

TEMA 1.

ESTRUCTURAS BASICAS:

Tuplas: () -> Parantesis y comas, acceso con corchetes, inmutables, mas rapidas.

- Operadore de creación para un elemento es (x,) con coma
- Tuplas vacías ()
- Menos características que las listas, no modificables pero más rápidas.

Listas: [] -> Corchetes y comas, acceso con corchetes, mutable

- Append para añadir al final
- Insert(pos,val) añade en pos el valor val
- Extend(list) añade los elementos de esa lista a la anterior
- Index(value) índice count(value) apariciones
- Remove(val) quita la primera aparición del[index] elimina el valor de un índice
- Reverse,sort(key=función)

- Conversión lista=list(tupla) tupla=tuple(lista)

Cadenas “”-> Comillas

* Operador “in”, comprueba si un valor aparece en un contenedor o subcadena en cadena.
* Operador “+”, produce una nueva tupla, lista o cadena concatenando sus contenidos.
* Operador “*”, produce una nueva t,l o c repitiendo el contenedor original.

COMPRENSIÓN DE LISTAS

- Lista=[2*i+5 for i in range (n+1)
- #L=[[0 for col in range(5)]for fil in range(6)]
  - 6 Filas de 5 columnas cada una
- n=[1,2,3,4] frutas=["m","ma","pe","pla"] print([(i,f)for i in n for f in frutas])
  - 1 con todos, 2 con todos, ….

OPERACIONES ADICIONALES:

- Enumarete(LISTA) : par índice/elemento
- Sorted(lista): devuelve lista ordenada (no modifica la original, eso es sort())
- Reversed(lista): devuelve lista invertida.
- Len(),min(),max(),sum()
- Isinstance(lista,list): True si es lista
- Lista.copy(): Devuelve copia de la lista

SET:

Colección no ordenada de datos (pueden ser de distinto tipo), set es mutable, frozenset e inmutableset no.

Add(x) permite añadir elementos, remove(excepción si no existe) y discard para borrar.

Pop devuelve y elimina un elemento aleatorio, con copy() copia todos los elementos y con clear() se vacía.

Hay operaciones s.union(s2) s.intersection(s2) s.difference(s2) s.symmetric_difference(s2)

TEMA 1.

DICCIONARIO:
Permite guardar pares clave/valor, claves inmutables, los valores si pueden modificarse. Denominados tablas hash o array asociativos, permite buscar, borrar, modificar o definir pares. Se usa{}, separando clave y valor con : y cada par con ,

EJ: datos("nombre":"pepe","edad":"25")

Para borrar podemos usar del dic("clave) o dic.pop("clave"), podemos eliminar solo valores poniendo a None o la lista entera con dic.clear().

Iterable con for normal for i in datos o se puede usar for clave,valoerin dactos.items()

ARRAY:
Se usa el módulo array, se crea indicando el tipo de dato, con operaciones de acceso similares a las listas[]

Arr=array("l",[1,2,2,3,4,5])

FICHEROS:
Las operaciones sobre ficheros son:

- Abrir f=open(nombre,modo(r,w,a)
- Cerrar f.close()
- Comprobar si existe os.path.isfile(nombre)
- Comprobar permisos os.stat(nombre)
- Leer un numero n de bytes f.read(n)
- Leer línea o todas las líneas f.readline() f.readlines()
- Consultas: name, closed, mode.
- Escribir: f.write("datos")

El modulo sys permite acceso a stdin, stdout y stdeer como entradas y salida estándar.

- Sys.path devuelve una lista de path del sistema

El modulo csv permite trabajar con este formato.
El modulo urlib.request permite leer datos de la web como si se tratara de un archivo.

Se pueden guardar estructuras pasándolas a string, leerlas de la misma manera sin problema (función útil __repr), guardar datos en binario es posible usando cadenas o el modulo struct.

El guardado de datos completos se puede hacer con el modulo cPickle y la función dump y load para guardar y cargar respectivamente.

Trabajar con datos en discto sin cargarlos es posible haciendo uso de shelve, que permite "abrirlos" para trabajar con ellos.

TEMA 1.

PROGRAMACIÓN FUNCIONAL

MAP

Con el formato map(función,objetoIterable) se aplica la función a cada uno de los elementos del objeto y devuelve un iterador con los resultados (casteo a list para acceder)

```
Ej:     def cuadrado(x): return x*x
        list(map(cuadrado,range(10,20)))
```

Suele hacer uso de funciones lambda.

FILTER

Con el formato filter(función,objetoIterable) se aplica la función a cada uno de los elementos, si la función devuelve True ese elemento se acepta, al igual que map hace falta casteo a list.

```
Ej:     list(filter (lambda x: x%2==0,range(10,20)))
```

REDUCE

Con el formato reduce(función,objetoIterable,inicializador]), se aplica la función a cada elemento del objeto iterable además de la suma hasta ese momento y realiza la suma acumulada.

La función debe tener dos argumentos, si existe inicializador se usará como primer argumento de la suma (valor inicial). Reduce se encuentra en un modulo de functools.

MANEJO DE ERRORES

La mejor soluciones es evitar que se realicen operaciones, o manejar el error cuando se produce, con try/except.

Las excepciones son eventos que pueden modificar el fliujo de un programa, se lanzan con los errores:

- Try/except: capturan y recuperan una excepción
  - Pueden existir varios except cada uno con un error() distinto
  - Con except vacio podemos recuperarlas todas.
- Rty/finnally: realiza acciones se produczca o no excepción
- Raise generar excepción manualmente
  - Raise nombre, datos
- Assert generar excepción de forma condicional.
  - assert x==0, “x es negativo”

Se suelen usar estas sentencias en operaciones que pueden fallar como apertura de ficheros, preferible un solo try y varios except que varios try/except. La excepción muere al ser capturada.

GENERADORES, ANOTACIONES y CÁLCULO SIMBÓLICO.

Los generadores permiten iterar sobre un conjunto de elemento una única vez sin guardar elementos

```
Def pares(lista):
    For i in lista:
        If i%2==0
            Yield i
```

```
nums=pares(lista)
for i in nums: print(i)
```

Los generadores deben instanciarse y cada vez que se llaman cogen uno de los valores denotado por yield, si se tienen varios yield llamará a uno cada vez.