Recordatorio contenidos

En los ejemplos, lo marcado como `<texto>` se interpreta como lo que debes rellenar con tu código según cada caso.

1. Tipos de datos y operadores

Tipo de dato	Clase	Ejemplo
Números enteros	`int`	`2`
Números reales	`float`	`2.5`
Números complejos	`complex`	`2 + 3j`
Valores booleanos	`bool`	`True/False`
Cadenas de texto	`str`	`"hola"`

Operación	Descripción	Ejemplo
`+`	Suma	`2.3+5.4`
`-`	Resta	`45.45-10.02`
`-`	Negación	`-5.4`
`*`	Multiplicación	`(2.3+4.2j)*3`
`**`	Potenciación	`2**8`
`/`	División	`100/99`
`//`	División entera	`100//99`
`%`	Módulo	`10%3`

Prioridad (mayor a menor): `()`; `**`; `*`, `/`, `//` o `%`; `+` o `-` .

Operación	Descripción	Ejemplo
`== (!=)`	Igual (distinto) a	`2==2`
`<(<=)`	Menor (o igual)	`1<1.1`
`>(>=)`	Mayor (o igual)	`3>=1`
`and`	Ambos `True`	`2>1 and 2<3`
`or`	Algún `True`	`2!=2 or 2==2`
`not`	Negación	`not True`

Prioridad (mayor a menor): `()`; `or`; `and`; `not`; comparadores.

2. Funciones predefinidas

- `int(arg)` convierte `arg` a entero.
- `float(arg)` convierte `arg` a número real.
- `str(arg)` convierte `arg` a cadena de texto (string).
- `list(arg)` genera una lista con elementos según `arg`, que debe ser *iterable* (strings, listas, tuplas, `range`).

3. Función `print`

- Un argumento: `print(arg)`
- Dos o más argumentos:
  `print(arg1, arg2,arg3)`
- Uso de parámetros, por ejemplo, para eliminar salto de línea y separar con guión:
  `print(arg, sep='-', end='')`

4. Función `input`

- `ret = input(texto)` guarda en `ret` un `str` ingresado.
- `ret = int(input(texto))` guarda en `ret` un `int` ingresado.
- `ret = float(input(texto))` guarda en `ret` un `float` ingresado.

5. `if/elif/else`

```
if <cond 1> :
  <codigo si se cumple cond 1>
if <cond 1.1> :
  <codigo si se cumple 1.1>
else :
  <codigo si no se cumple 1.1>
elif <cond 2> :
  <codigo si se cumple cond 2 pero no cond 1>
else :
  <codigo si no se cumple cond 1 ni cond 2>
```

6. `while`

```
while <condicion> :
  <codigo que se ejecuta repetidas veces
mientras se cumpla condicion>
```

7. Funciones propias

```
def funcion(<argumentos>):
  <codigo de funcion>
  return <valor de retorno>
```

Variables y parámetros definidos dentro de funciones no son visibles fuera de la función (*scope* local).

8. Strings, clase `str`

Acceso a caracteres particulares con operador `[]`, partiendo con índice cero. Porción de string con *slice*, por ejemplo si `string='Hola'`, `string[1:3]` es `'ol'`. Algunos métodos y funciones de strings:

- Operador `+`: une (concatena) dos strings.
- Operador `in`: cuando `a in b` retorna `True`, entonces el string `a` está contenido en el string `b`.
- `string.find(a)`: determina si `a` está contenido en `string`. Retorna la posición (índice) dentro de `string` donde comienza la primera aparición del sub-string `a`. Si no está, retorna `-1`.
- `string.upper()`, `string.lower()`: retorna `string` convertido a mayúsculas y minúsculas, respectivamente.
- `string.strip()`: retorna un nuevo string en que se eliminan los espacios en blanco iniciales y finales de `string`.
- `string.split(a)`: retorna una lista con los elementos del `string` que están separados por el string `a`. Si se omite `a`, asume que el separador es uno o más espacios en blanco o el salto de línea.
- `p.join(lista)`: suponiendo que `p` es un string, retorna un nuevo string conteniendo los elementos de la lista "unidos" por el string `p`.
- Función `len(string)`: entrega el número de caracteres de `string`.

Una forma de iterar sobre los caracteres de `string`:

```
for char in string:
  <operaciones con el caracter char>
```

9. Listas

Secuencias de elementos-objetos. Se definen como `lista = [<elem1>,<elem2>,...,<elemN>]`. Los elementos pueden o no ser del mismo tipo. Para acceder al elemento `i`, se usa `lista[i]`. La sublista `lista[i:j]` incluye los elementos desde la posición `i` hasta `j-1`. Algunos métodos y funciones de listas:

- Operador `+`. concatena dos listas.
- Operador `in`: `a in b` retorna `True` cuando el elemento `a` está contenido en la lista `b`. Y `False` en otro caso.
- `lista.append(a)`: agrega `a` al final de la lista.
- `lista.insert(i,a)`: inserta el elemento `a` en la posición `i`, desplazando los elementos después de `i`.
- `lista.pop(i)`: retorna el elemento de la lista en la posición `i`, y lo elimina de `lista`.
- `lista.remove(elem)`: elimina la primera aparición de `elem` en la lista.
- Función `len(lista)`: entrega el número de elementos de `lista`.

Para iterar sobre los elementos de `lista`:

```
for elem in lista:
  <lo que quieran hacer con elem>
```

10. Programación oo

```
class <NombreClase>:
  def __init__(self, <parametros>):
    self.<atributos> = <algun parametro>
  def __str__(self):
    <codigo sobrecarga de funcion str()>
    return <string que representa
    el objeto>
  def <metodo propio>(self, <parametros>):
    <codigo de modulo propio>
```

11. Archivos

Abrir un archivo:
`archivo = open(<nombre archivo>,<modo>)`,
p. ej. `archivo = open('archivo.txt','r')`. `<modo>` puede ser `'w'` para escribir un archivo nuevo, `'r'` para leer un archivo (predeterminado), y `'a'` para escribir en un archivo ya existente, agregando datos al final del archivo.

Algunos métodos del objeto que retorna la función `open`:

- `archivo.readline()`: retorna un string con la línea siguiente del archivo, comenzando al inicio del archivo.
- `archivo.write(string)`: escribe en el archivo el string `string`.
- `archivo.close()`: cierra el archivo.

Para leer un archivo entero puedes usar `for`, que iterará línea por línea del archivo:

```
archivo = open('archivo.txt','r')
for linea in archivo:
  <lo que quieran hacer con linea>
```

12. Búsqueda y ordenamiento

Búsqueda secuencial o lineal Busca secuencialmente un elemento dentro de una lista de tamaño n hasta encontrarlo. Peor caso: elemento no está en lista, n comparaciones. Uso: lista desordenada.

Búsqueda binaria Supone lista ordenada. Divide la lista en sublistas: si el elemento es mayor que el elemento medio de la lista, se sigue por la sublista derecha; si no, por la lista izquierda. Peor caso: para una lista de n elementos, se realizan $log_2(n)$ comparaciones.

Ordenamiento por selección Busca el índice del elemento más pequeño de la lista, e intercambia los valores entre el índice encontrado y el primer elemento; luego, encuentra el segundo elemento más pequeño, y lo intercambia con el elemento de la segunda posición, realizando la misma operación para el tercero, cuarto, etc..

Ordenamiento por inserción Itera por los elementos de la lista, partiendo por el primer elemento, y generando una lista ordenada con los elementos mientras itera. Es similar a cómo una persona ordena una lista de cartas.

Métodos de Python para ordenamiento

- `lista.sort()`: ordena `lista` en forma ascendente.

13. Recursión y backtracking

Elementos de una función recursiva:

- Caso base: para terminar la recursión.
- Llamada recursiva: llamada a la misma función dentrl de la función, con parámetros distintos que hacen disminuir el problema original.

Para programar una función recursiva, descomponer la función en elementos que puedan ser llamados con subconjuntos de datos. Ejemplo: suma de los primeros N números

```
def suma(N):
  if N == 1: # caso base
    return 1:
  else: #descomponer en N y restantes N-1
    return N + suma(N-1)
```

Backtracking: cuando hay muchas formas de buscar una solución. Se puede formar un arbol con las soluciones, y el algoritmo comienza en el nodo raíz. Cada camino dentro del árbol es un código recursivo, y dentro del código se exploran las ramas del árbol. Ejemplo: encontrar la suma de todas las formas de combinar una lista de números:

```
def suma_conm_lista(lista, sublista=[], suma=0):
  # Caso base: sublista tiene N elementos,
  # o la lista tiene 0 elementos
  if len(lista) == 0:
    print sublista, suma
    return
  else:
    # Quito un elemento de la lista y lo
    # agrego a la sublista para llamar a la
    # funcion en forma recursiva, y aumento
    # la suma con el elemento que saque
    for i in range(len(lista)):
      suma_conm_lista( lista[:i]+lista[i+1:],\
          sublista+[lista[i]], suma+lista[i])
```