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
Valores booleanos	`bool`	`True/False`
Cadenas de texto	`str`	`"hola"`

Operación	Descripción	Ejemplo
`+`	Suma	`2.3+5.4`
`-`	Resta	`45.45-10.02`
`*`	Multiplicación	`2.3*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): `()`; comparadores; `not`; `and`; `or`.

2. Conversiones

- `int(arg)` entrega `arg` como entero.
- `float(arg)` entrega `arg` como número real.
- `str(arg)` entrega `arg` como cadena de texto (string).

3. Input/Output

- Un argumento: `print(arg)`
- Dos o más argumentos:
  `print(arg1, arg2, arg3)`
- `ret = input(texto)` guarda en `ret` un `str` ingresado.

4. `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>
```

5. `while`

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

6. `for`

```
for x in range(a, b) :
  <codigo que se ejecuta b-a veces, con x
  tomando valores desde a hasta (b-1)>
```

7. Funciones

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

- `random.randint(a, b)` retorna un número entero aleatorio mayor o igual a `a` y menor o igual a `b`.
- `round(n, x)` retorna el número `n` redondeado en `x` decimales.

Para acceder a una función perteneciente a un módulo, antes de utilizarla debes importar dicho módulo poniendo `import <nombre modulo>`

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:

- Función `len(string)`: entrega el número de caracteres de `string`.
- Operador `+`: une (concatena) dos strings.
- Operador `in`: cuando `a in b` retorna `True`, entonces el string `a` está contenido en el string `b`.
- `string.upper()`, `string.lower()`: retorna `string` convertido a mayúsculas y minúsculas, respectivamente.
- `string.isupper()`: retorna `True` si todos los caracteres alfabéticos están en mayúscula, y `False` en caso contrario, o bien si el string es vacío.
- `string.islower()`: retorna `True` si todos los caracteres alfabéticos están en minúscula, y `False` en caso contrario o bien si el string es vacío.
- `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.

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 = [<elem0>,<elem1>,...,<elem(N-1)>]`. 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.pop(i)`: retorna el elemento de la lista en la posición `i`, y lo elimina de `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. 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 y `'r'` para leer un archivo. Si no se ingresa `<modo>`, se asume que que es en modo `'r'`.

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

- `archivo.readlines()`: retorna una lista, en donde cada uno de los elementos de la lista corresponde a una línea del archivo. `string`.
- `archivo.close()`: cierra el archivo.
- `print(string1, string2, sep=<sep>, file=<file>)`: Escribe en el archivo `<file>` los string `string1` y `string2`, separados por `<sep>`.

11. Programación OO

```
class <NombreClase>:
  def __init__(self, <parametros>):
    self.<atributos> = <valor inicial>
  def __str__(self):
    return <string que representa
    el objeto>
  def <metodo propio>(self, <parametros>):
    <codigo de metodo propio>
```

12. Ordenamiento

Métodos de Python para ordenamiento

- `lista.sort()`: ordena `lista` en forma ascendente.
- `lista.sort(key=function)`: ordena `lista` en forma ascendente basado en el valor retornado por `function`.