Referencia Rápida de Python

FPI/FCyP

FACULTAD DE INGENIERÍA
UNIVERSIDAD DE SANTIAGO DE CHILE

Tipos Básicos

`int`	Números enteros	`42`
`float`	Números flotantes	`1.618`
`bool`	Valores lógicos	`True`
`str`	Strings	`"Monty"`
`list`	Listas	`[1, 2, 3]`

Operaciones Básicas

`input("Mensaje")`	Muestra `Mensaje` y recibe entrada por teclado del usuario
`print(valor,...)`	Muestra por pantalla los `valores` separados por coma
`x = y`	Asignación, `x` toma el valor de la expresión `y`

Operadores Aritméticos

A igual nivel de precedencia (P) en las operaciones, se agrupan de a pares de izquierda a derecha.

Ejemplo	Operación	P	T[a]
`x ** y`	Exponenciación[b]	1	B
`+ x`	Identidad	2	U
`- x`	Cambio de signo	2	U
`x * y`	Multiplicación	3	B
`x / y`	División	3	B
`x // y`	División entera	3	B
`x % y`	Módulo (resto[c])	3	B
`x + y`	Suma	4	B
`x - y`	Resta	4	B

Los operadores unarios (U) toman solo un operando.
Los operadores binarios (B) toman dos.
Todos los operadores aritméticos binarios tienen una versión de asignación como `+=`, que se usa como en el ejemplo:

`x += y` se interpreta como `x = x + y`

[a] Tipo
[b] Se agrupan de derecha a izquierda.
[c] De la división entera.

Operadores lógicos

A igual nivel de precedencia en las operaciones, se agrupan de a pares de izquierda a derecha.

Ejemplo	Operación	P	T
`x > y`	Mayor que	5	B
`x >= y`	Mayor o igual que	5	B
`x < y`	Menor que	5	B
`x <= y`	Menor o igual que	5	B
`x == y`	Igual que	5	B
`x != y`	Distinto que	5	B
`x in l`	Pertenece a	5	B
`x not in l`	No pertenece a	5	B
`not p`	Negación	6	U
`p and q`	Y lógico	7	B
`p or q`	O lógico	8	B

Decisiones con `if`

Ejecutar sentencias que solo deben ocurrir al cumplirse la `<condición>` (expresión booleana):

```
if <condición>:
    <sentencias condicionadas>
```

Ejecutar sentencias que solo deben ocurrir al cumplirse una condición, y otras en caso que no:

```
if <condición>:
    <sentencias condicionadas>
else:
    <sentencias alternativas>
```

Ejecutar sentencias que solo deben ocurrir al cumplirse una condición, otras en caso de que no se cumpla la primera condición, pero si una segunda condición[a], y otras en caso de que no se cumpla la primera ni la segunda:

```
if <condición 1>:
    <sentencias condicionadas 1>
elif <condición 2>:
    <sentencias condicionadas 2>
else:
    <sentencias alternativas>
```

[a] Tantas condiciones secundarias (`elif`) como se necesite.

Bloques de decisiones

Cada secuencia `if`, `if-else` o `if-elif-else` es un bloque independiente.
Dentro de un mismo bloque se ejecutarán las sentencias condicionadas a la primera condición válida comprobada.
Si existieran 2 bloques de decisión consecutivos se ejecutarán las sentencias condicionadas a la primera condición válida para cada bloque de manera independiente.

Strings

`s = "Monty"` `s = 'Python'`	Define un objeto de tipo string.
`s.find(sub)`	Retorna el índice donde empieza `sub`.
`s.rfind(sub)`	Como `find`, pero desde la derecha.
`s.index(sub)`	Como `find`, pero arroja error si no encuentra.
`s.lower()`	Retorna el string en minúscula.
`s.upper()`	Retorna el string en mayúscula.
`s.strip(t)`	Retorna el string eliminando los caracteres en `t` de los extremos del string.
`s.strip()`	Como `strip`, pero eliminando espacios en blanco.
`s.capitalize()`	Retorna el string convertido con el primer caracter a mayúscula, si es una letra, y el resto a minúscula.
`s.title()`	Retorna el string en "Formato De Título".
`s.count(sub)`	Retorna la cantidad de apariciones no superpuestas de `sub`.

Consultas sobre Strings

Todos estos métodos devuelven `True` o `False`.

`s.isupper()`	Todas las letras son mayúsculas y hay al menos una.
`s.islower()`	Todas las letras son minúsculas y hay al menos una.
`s.isalpha()`	Todos los caracteres son alfabéticos.
`s.isdigit()`	Todos los caracteres son dígitos.
`s.isalnum()`	Todos los caracteres son alfabéticos o dígitos.
`s.endswith(t)`	El string termina con `t`.
`s.startswith(t)`	El string empieza con `t`.

Ciclos con `while`

Realizar una repetición condicionada de sentencias:

```
while <condición>:
    <sentencias_a_repetir>
```

Donde `<condición>` es una expresión booleana y `<sentencias_a_repetir>` (última instrucción indentada con respecto a `while`) es la secuencia de instrucciones a repetir.
A través de este bloque, se asegura la ejecución de las `<sentencias_a_repetir>` mientras se cumpla la `condición` señalada.
Se puede repetir cualquier sentencia, incluyendo otros ciclos.

Función `range`

Generación de secuencias numéricas:

- `range(stop)` genera números de 0 a `stop-1`.
- `range(start, stop, d)` genera números desde `start` hasta `stop-1`, con distancia de `d` entre ellos.

Nótese que `range` no es una lista, pero puede convertirse a una mediante `list(range(n))`.

Ciclos con `for`

Repite la acción por cada elemento de la secuencia:

```
for elemento in secuencia:
    acciones_a_repetir
```

La variable `elemento` es definida en el `for` y su valor es cada elemento de la secuencia en orden.
Entre las secuencias, se incluyen archivos, `list`, `str`, `range`, etc.

Banderas (*Flags*)

Parte de la `<condición>` puede ser una variable booleana: esta indica si se debe continuar o no, según qué ocurra en el ciclo:

```
i = 1
keep_going = True
while keep_going and i <= 5:
    if i % 2 == 0:
        keep_going = False
    i += 1
```

Haciendo uso de decisiones (`if`) y de una bandera (`keep_going`), se puede verificar si se cumple una condición adicional bajo la cual detener el ciclo antes de que el iterador `i` llegue a su límite.

Slices y Copias

Pueden accederse cortes (*slices*) de una secuencia como una lista o string mediante la notación slice. Su notación básica es

`objeto[a:b:c]`,

retornando un objeto del mismo tipo del seccionado, con `a` el índice inicial del corte, `b` el índice final (él último índice retornado será siempre el mayor valor posible menor a `b`) y `c` la distancia entre dos elementos consecutivos recuperados. `a` y `b` deben ser índices válidos y `c` debe ser un número entero (positivo o negativo).
Esta notación tiene las siguientes variaciones:

- Si se omite `a` (pero no `:`), inicia desde el comienzo.
- Si se omite `b` (pero no `:`), llega hasta el final.
- Si se omite `c` (incluyendo o no `:`), se asume 1.

Listas

`lista = [1, 2, 3]`	Define un objeto de tipo lista.
`lista[i]`	Retorna el elemento en la posición `i`. Soporta de `0` a `n-1`[a] de izquierda a derecha y de `-1` a `-n` de derecha a izquierda.
`lista.append(4)`	Añade el elemento `4` al final de la lista.
`lista[j] = z`	Redefine el valor del elemento en la posición `j` de la lista a `z`.
`lista.pop(k)`	Retorna el elemento en la posición `k` y lo elimina de `lista`. Sin parámetros retorna y elimina el último elemento.
`lista.count(c)`	Retorna la cantidad de apariciones del elemento `c`.
`lista.index(d)`	Retorna la posición del elemento `d`.
`lista.remove(e)`	Elimina la primera aparición del elemento `e`.
`lista.insert(i, f)`	Inserta el elemento `f` en la posición `i`[b].
`lista.sort()`	Ordena `lista` en orden creciente.
`lista1 + lista2`	Retorna una lista que concatena `lista1` y `lista2`.
`lista * n`	Retorna una lista que concatena `lista` `n` veces[c].

[a] Con `n` equivalente al largo de la lista.
[b] Los elementos siguientes son desplazados en 1 posición a la derecha.
[c] `n` debe ser entero.

Largo de una Secuencia

El largo de la secuencia `seq` se obtiene mediante la función nativa `len(seq)`. Se entiende por "largo" de una secuencia como el total de elementos que hay en esta.
En el caso de una lista, si tiene como elemento otra lista, este elemento sigue contando como uno solo.

String a Lista y Viceversa

- `s.split(sep)` separa el string utilizando `sep` como separador y retorna una lista cuyos elementos son los fragmentos del string.
- `sep.join(lista)` une los elementos de `lista`, separados por `sep`, en un único string. Solo funciona si los elementos de la lista son strings.