

[Python] Cheat Sheet

Este documento presenta un pequeño repaso sobre los conceptos fundamentales que vimos en la materia y que sirven para resolver los ejercicios. Bajo ningún punto asuma que esta lista repasa todos los conceptos necesarios para aprobar dicha materia.

Types:

En todo lenguaje es importante entender los tipos de datos básicos que tenemos disponible. A continuación veremos un breve resumen de los tipos existentes en la distribución estándar de Python.

- tipos de texto → `str`
- tipos numéricos → `int` , `float` , `complex` *
- tipos de secuencias → `list` , `tuple` , `range` **
- tipos de mapas → `dict`
- tipos de conjuntos → `set` , `frozenset` *
- tipos booleanos → `bool`
- tipos binarios → `bytes` * , `bytearray` * , `memoryview` *
- tipo nulo → `None`

▼ Obs:

* → no es utilizado en la materia

** → es un tipo especial de datos y lo veremos en profundidad

Es importante destacar aquellos factores característicos de cada tipo de datos nativo para tomar decisiones informadas.

Factores principales de cada tipo build-in

Aa Type name	≡ Properties
<u>String</u>	Sirve para representar conjunto de caracteres.
<u>Integer</u>	El tipo entero es capaz de representar cualquier numero entero dentro de su capacidad de representación (existe el máximo pero es muy grande)
<u>Float</u>	Los Float son capaces de representar números reales, tal como los integers tiene una limitación en cuanto a su capacidad de representación.
<u>List</u>	Las listas permiten agrupar ordenadamente cualquier conjunto de elementos.

Aa Type name	☰ Properties
<u>Tuple</u>	Las tuplas, tienen un comportamiento idéntico a las listas con la particularidad que son inmutables. Esto último implica que no puede eliminarse ni agregarse elementos a una tupla.
<u>Dict</u>	Los diccionarios permiten generar un mapeo de tipo clave-valor entre elementos. Recordar que cualquier elemento de tipo clave debe tener una particularidad de ser "hasheable" por lo que se recomienda utilizar tipos numéricos o de texto.
<u>Set</u>	Los conjuntos no ordenados permiten hacer operar con aritméticas de conjuntos logrando obtener: la unión, la intersección, la diferencia y el complemento. Recordar que el set no permite repetidos.
<u>Bool</u>	Los booleanos solo tienen dos valores siendo True y False.
<u>NoneType</u>	El elemento None pertenece a todos los tipos y representa el nulo de cada conjunto. Se puede considerar como la ausencia de un valor.

Casting:

El casteo es la acción de tomar un valor de un tipo dado y transformarlo en su representación correspondiente en otro tipo.

Para los tipos Build-In encontramos que el lenguaje tiene funciones definidas para este propósito:

- `int()` Nos permite pasar un valor de cualquier otro tipo a su representación entera. Siempre y cuando esta representación exista, la función devolvera un resultado, en caso contrario lanza un `error`.

Ejemplo:

```
'''
The following code will leave the int value of 17 assign to the variable age_as_int
'''
age_as_txt = '17'
age_as_int = int(age_as_txt)

'''
The following code will not work because the text does not match with and actual integer
'''

age_as_txt = "I'm 17"
age_as_int = int(age_as_txt)

# Obs! For all purposes think of each example as individual cases
# that do not share any relation.
```

- `float()` Nos permite pasar un valor de cualquier tipo a su representación en números reales cuando dicha relación exista. Caso contrario lanza un `error`

Ejemplo:

```
'''
The following code will leave the float value of 17 assign to the variable asg_as_float
'''

age_as_txt = '17'
age_as_float = float(age_as_txt)

'''
The following code does not work because there is no relationship between does values.
'''
age_as_list = [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
age_as_float = float(age_as_list)
```

- `str()` Nos permite obtener la representación de un valor en su formato texto.

Ejemplo:

```
this_is_a_list = [1,2,3, 'leo', 'Alice', (1.0, 4.5)]
this_is_text = str(this_is_a_list)
# if we print the value of 'this_is_text' we obtain the following:
# >>> print(this_is_text)
# >>> '[1,2,3, 'leo', 'Alice', (1.0, 4.5)]'
```

- `list()` Permite generar una lista de cualquier valor que sea `iterable`.

Ejemplo:

```
text = 'this is a sequence of characters'
text_as_list = list(text)
# as a result we cna operate with the same value form the perspective of a list

'''
The following example does not work because the numeric value is not an iterable
'''
number = 1
number_as_list = list(number)
```

Sequence Handling:

En Python encontramos 4 tipos build-in para el manejo de colecciones que son: `list`, `tuple`, `dict`, `set`. Cada uno tiene sus fortalezas y sus cualidades que lo hacen especialmente útil para diferentes situaciones.

Dichos tipos comparten ciertas características:

- Son `iterables` por lo que se puede utilizar un `for` para iterar por cada valor de la colección.

- Tienen una cardinalidad (cantidad de elementos) asociada. Esto significa que se puede aplicar la función `len()` y obtener un resultado.
- Tienen la posibilidad de trabajar con aritmética de conjuntos según las limitaciones de cada tipo.

List:

Nos permiten coleccionar multiples valores bajo una sola variable. Acepta repetidos y modificaciones, eliminaciones e inserciones.

Las listas son ordenables, por lo que si los elementos tienen un "orden natural" python nos permite ordenar dichos elementos.

Como las listas son un tipo de conjunto, las mismas pueden unirse.

- Creación de una lista:

```
# Using [ and ]
a_list = [1, 2, 4]

# using the special function list
a_list = list((1, 2, 3)) # Note that we are using double (( and ))
```

- Operaciones:

```
# Modify:

a_list = [1, 2, 3]
a_list[1] = '2'

'''
Now if we print the value:
>>> print(a_list)
>>> [1, '2', 3]
'''

# Insertion:
a_list = [1, 2, 3]
a_list.append(4) # add new item to the end of the list
a_list.insert(1, 5) # will insert the value 4 at the 2 position (zero base index)
'''
As a result of the above:
>>> print(a_list)
>>> [1, 5, 2, 3, 4]
'''

# Deletion:
a_list = [1, 2, 3]
a_list.pop() # removes the last item and returns the value
a_list.pop(0) # removes the item at the specified position
'''
As a result:
```

```

>>> print(a_list)
>>> [2]
'''

# applying len

a_list = [1, 23, 4]
'''
In the case of lists, len returns the amount of items inside the collection
>>> len(a_list)
>>> 3
'''

# using a list as argument of a for-loop

for number in a_list:
    print(number)

'''
The above code will produce the following output:
>>> 1
>>> 23
>>> 4
'''

# Join 2 lists

a_list = [1, 2, 3]
another_list = [4, 3, 5]
a_list.extend(another_list)
final_list = a_list
# also you can simply 'add' both lists
final_list = a_list + another_list
'''
As a result of the above code:
>>> final_list
>>> [1, 2, 3, 4, 3, 5]
'''

```

Operadores sobre listas

Aa Nombre	Descripción
<u>append()</u>	Inserta al final de la lista
<u>clear()</u>	elimina todos los elementos de una lista
<u>copy()</u>	retorna una copia de la lista (shallow copy)
<u>count()</u>	retorna el número de elementos con su valor asociado
<u>extend()</u>	inserta los elementos de cualquier iterable al final de la lista actual
<u>index()</u>	retorna el primer índice de aquel elemento que coincide en valor
<u>insert()</u>	Inserta un valor en la posición indicada
<u>pop()</u>	Elimina un valor en la posición indicada. Por Default es el último
<u>reverse()</u>	ordena la lista en sentido inverso al actual
<u>sort()</u>	Ordena la lista según el orden natural de sus elementos



`sort` toma el valor de orden natural entre los elementos, es decir el orden que conocen entre números. Intente entender como es su comportamiento con el resto de los tipos.

Describe como se comporta el el siguiente ejemplo

```
lista = [3, 4, 5, 1, 'a']
lista.sort()
```

Tuple:

Las tuplas nos permiten tener seguridad de que **una vez creadas no serán modificadas**.


```
# Creation using ( )
a_tuple = (1, 2, 3)
# Creation using function
a_tuple = tuple([1, 2, 3, [1, 5, 6]])

# Len function
'''
The len function returns the amount of elements inside the tuple.
>>> len(a_tuple)
>>> 4
'''

# Join 2 tuples
a_tuple = (1, 2, 4)
another_tuple = tuple([6, 7, 89])
final_tuple = a_tuple + another_tuple

'''
With tuples we can only add tuples by creating a new one.
>>> final_tuple
>>> (1, 2, 4, 6, 7, 89)
'''
```

Operadores sobre Tuplas

<u>Aa</u> Nombre	 Descripción
<u>count()</u>	Devuelve la cantidad de ocurrencias del elemento especificado
<u>index()</u>	busca el valor especificado en la tupla y devuelve su primer ocurrencia
<u>Untitled</u>	

Set:

Los conjuntos no ordenados (`sets`) tienen la particularidad de que **no son ordenados** y **no son indexables**. Estas características hacen que el `slicing` y el `sort` ing no sean aplicables.

→ Los `sets` permiten trabajar aritméticamente como los diagramas de Venn.

→ No permiten duplicados!

```
# Creation
a_set = set([1, 3, 4, 5, 6])
# Do notice that sets can only be created by the use of the set function

# Sets are useful to eliminate duplicates
a_set = set([1, 1, 1, 4, 4, 4, 4])
'''
When you create a set, it removes all duplicates. The order isn't guaranteed
>>> a_set
>>> [1, 4]
'''

# Len
a_set = set([1, 3, 4, 5, 5, 3])
'''
When calling the len function, it returns the amount of unique values it contains.
>>> len(a_set)
>>> 4
'''

# Venn's operations

# union

a_set = set([1, 2, 3])
another_set = set([1, 3, 4, 5])
final_set = a_set + another_set
'''
The + sign symbolize the union for sets.
>>> final_set
>>> [1, 2, 3, 4, 5]
Order not guaranteed
'''

# Diference
a_set = set([1, 2, 3])
another_set = set([1, 3, 4, 5])
final_set = a_set - another_set
'''
The - sign symbolize the diference between sets
>>> final_set
>>> [2]
'''

# Intersection
# You can infer this one ;)

# Adding element
a_set = set([1,2,3])
a_set.add(1) # in this case, the operation does nothing because the set already contains the element
a_set.add(5) # this adds the value fo 5 to the set
'''
The operation only takes effect if the value isn't already contained.
>>> a_set
>>> [1, 2, 3, 5]
Order not guaranteed
```

```
'''
# Deletion
a_set = set([1, 2, 3])
a_set.remove(3) # removes the specified element
a_set.pop() # removes a random element

'''

Python presents 2 method for deletion, remove let us specified the element to be removed.
While pop removes whichever elements happens to be at the last position of the set at the moment.
>>> a_set
>>> [1] or [2]
We can't be sure as the pop call could have remove either
'''

# Using sets in for-loops
a_set = set([1, 23, 4, 4, 5, 6])
for elem in a_set:
    print(elem)
'''

The above code will produce the following output, order is not guaranteed:
>>> 1
>>> 23
>>> 4
>>> 5
>>> 6
'''
```

Operadores sobre Sets

Aa Nombre	Descripción
<u>add()</u>	Inserta un nuevo elemento en el set, solo si este no se encuentra
<u>clear()</u>	Elimina todos los elementos
<u>copy()</u>	Shallow copy de los elementos del set
<u>difference()</u>	retorna la diferencia entre 2 o más sets
<u>difference_update()</u>	remueve los elementos del set actual que se encuentren en otro set
<u>discard()</u>	si el elemento especificado se encuentra en el set, se lo remueve
<u>intersection()</u>	retorna la intersección entre 2 sets
<u>intersection_update()</u>	remueve los elementos del set actual que no se encuentren en el otro set
<u>isdisjoint()</u>	retorna si el set actual es disjunto respecto del otro o no
<u>issubset()</u>	retorna si el set actual es un subconjunto del otro
<u>issuperset()</u>	retorna si el set actual es un superset del otro
<u>pop()</u>	remueve el elemento que se encuentre en la última posición a la hora de ejecutarlo
<u>remove()</u>	remueve un elemento especificado del set, si el mismo no se encuentra lanza un error

Aa Nombre	Descripción
<code>symmetric_difference()</code>	retorna un nuevo set con los elementos de la diferencia simétrica entre el set actual y el otro
<code>symmetric_difference_update()</code>	inserta los elementos del set actual que se encuentren en la diferencia simétrica con el otro set
<code>union()</code>	devuelve la unión del set actual con el otro
<code>update()</code>	modifica el set actual para insertar la unión con el otro

Dict:

Los diccionarios son colecciones de elementos **clave-valor** , para generalidad de este curso tomaremos que son **NO ordenados, modificables y no permiten repetidos**.

Los diccionarios mantienen valores en pares **key:value** , donde las claves no pueden repetirse y deben ser **Hasheables**.

- El uso de corchetes en **dicts** hace referencia al componente **key** del par. Si se ejecuta una asignación, modificamos el **value** y si lo usamos en la referencia estamos accediendo al **value** .

```
# Creation
a_dict = {
    'key_1': 1,
    'key_2': 2,
    'key_3': 3
}

a_dict = dict() # empty dict

# Insert element
a_dict = dict()
a_dict[2] = 'hi!' # the key 2 contains the value of 'hi!'

a_dict[2] = 'something else' # inserting another value to the same key replaces the value
'''
Insertion of keys replaces the last known value
>>> a_dict
>>> {2: 'something else'}
'''

# Access Item
a_dict = {'a': 1, 'b': 2}
'''
To acces a value just reference the key, in case the key does not exist python raise an Error
>>> a_dict['a']
>>> 1

You can also use the get function with 2 arguments.
a_dict.get(key, default) where key means the key you want to access and the default is
the specified value to return in case the key does not exist (by default is None).
>>> a_dict.get('n', False)
>>> False
'''
```

```

# Remove item:
a_dict = {'a': 1, 'b': 2}
a_dict.pop('a') # this will remove the pair which key value is 'a'
'''
Dictionaries pay special attention to keys, so to remove a value you need to specify
the key which contains that value.
>>> a_dict
>>> {'b': 2}
'''

# keys
a_dict = {'a': 1, 'b': 2}
keys = a_dict.keys()
'''
the keys function returns a list of all available keys within the dict
>>> keys
>>> ['a', 'b']
Assume unordered list
'''



# values
a_dict = {'a': 1, 'b': 2}
vals = a_dict.values()
'''
the values function returns a list containing all values
>>> values
>>> [1, 2]
Assume unordered
'''

# items
a_dict = {'a': 1, 'b': 2}
items = a_dict.items()
'''
The items function returns a list of pairs key-value
>>> items
>>> [('a', 1), ('b', 2)]
Assume unordered
'''

# len
a_dict = {'a': 1, 'b': 2}
'''
Len function returns the amount of items a dictionary contains
>>> len(a_dict)
>>> 2
'''

```

Operadores sobre diccionarios

 Nombre	 Descripción
<u>clear()</u>	remueve todos los elementos del diccionario
<u>copy()</u>	retorna una shallow copy del diccionario
<u>fromkeys()</u>	retorna un nuevo diccionario con las keys y valores especificados
<u>get()</u>	retorna el valor de la key especificada, o el default en caso de no encontrarla
<u>items()</u>	retorna una lista con los pares key-value del diccionario

Aa Nombre	Descripción
<u>keys()</u>	retorna una lista con las claves del diccionario
<u>values()</u>	retorna una lista con los valores del diccionario
<u>pop()</u>	remueve el valor de la clave especificada
<u>setdefault()</u>	retorna el valor de la key, en caso de no existir setea el valor especificado en dicha clave
<u>update()</u>	actualiza el diccionario con los key-value pairs especificados

Slicing

Las secuencias (incluidos los strings) tienen la posibilidad de tomar **slice**s o sub conjuntos de los mismos.

Ejemplos:

```
# List
a_list = [1, 2, 3, 4, 5, 6, 7, 8]

a_list[-3:] # => [6,7,8]
a_list[-1] # => 8
a_list[:3] # => [1, 2, 3]
a_list[3:4] # => [4]

# Tuple
a_tuple = (1, 2, 3, 4, 5, 6, 7, 8)

a_tuple[-3:] # => [6,7,8]
a_tuple[-1] # => 8
a_tuple[:3] # => [1, 2, 3]
a_tuple[3:4] # => [4]
```



Recordar que por default Python toma la convención de que los intervalos son de tipo semi-abiertos a derecha (incluir el primer elemento pero no el último).

Matemáticamente:

ConjuntoA = [desde, hasta_sin_incluir)

String Manipulation:

El tipo de String es muy utilizado, el mismo representa text en forma de una colección ordenada de caracteres (símbolos).

Operadores para Strings

 Nombre	 Descripción
<code>capitalize()</code>	<u>convierte el primer carácter en mayúscula.</u>
<code>casefold()</code>	<u>convierte el texto en minúscula</u>
<code>count()</code>	<u>retorna la cantidad de veces que un carácter especificado se utiliza en el string</u>
<code>endswith()</code>	<u>retorna True si el string termina con el valor especificado</u>
<code>expandtabs()</code>	<u>settea el tamaño de los tabs</u>
<code>find()</code>	<u>Busca el valor especificado y retorna la posición donde se encuentra o -1 en caso contrario</u>
<code>format()</code>	<u>aplica un formato especificado sobre el string</u>
<code>index()</code>	<u>Idem find().</u>
<code>isalnum()</code>	<u>retorna True si todos los caracteres del string son alfanuméricos</u>
<code>isalpha()</code>	<u>retorna True si todos los valores del string están en la el alfabeto</u>
<code>isdecimal()</code>	<u>retorna True si todos los valores del string son decimales</u>
<code>isdigit()</code>	<u>retorna True si todos los valores del string son dígitos</u>
<code>isidentifier()</code>	<u>retorna true si el string es un identificador válido (alfanumérico o _).</u>
<code>islower()</code>	<u>retorna True si todos los valores del string son minúsculas</u>
<code>isnumeric()</code>	<u>retorna True si todos los valores del string son numéricos</u>
<code>isprintable()</code>	<u>retorna True si todos los valores del string son printables (ascii).</u>
<code>isspace()</code>	<u>retorna True si todos los valores del string son espacios</u>
<code>isupper()</code>	<u>retorna True si todos los valores del string son mayúsculas</u>
<code>join()</code>	<u>Inserta los valores del un elemento iterable en el final del string</u>
<code>lower()</code>	<u>convierte un string en minúsculas</u>
<code>lstrip()</code>	<u>retorna un string sin espacio a izquierda</u>
<code>replace()</code>	<u>retorna un string donde el valor especificado fue reemplazado</u>
<code>rfind()</code>	<u>Busca el valor especificado y retorna su última aparición o -1</u>
<code>rindex()</code>	<u>idem rfind().</u>
<code>rsplit()</code>	<u>divide un string en el identificador especificado y retorna una lista con sus partes</u>
<code>rstrip()</code>	<u>retorna un string sin espacios a derecha</u>
<code>split()</code>	<u>idem rsplit().</u>
<code>splitlines()</code>	<u>retorna una lista con el valor de cada renglón</u>
<code>startswith()</code>	<u>retorna True si el string comienza con el valor especificado</u>
<code>strip()</code>	<u>retorna un string sin espacios a izquierda ni a derecha</u>
<code>swapcase()</code>	<u>cambia mayúsculas por minúsculas y viceversa</u>
<code>title()</code>	<u>convierte el primer carácter de cada palabra en mayúscula</u>
<code>upper()</code>	<u>convierte el string en mayúsculas</u>

