Cheatography

Python3 para principiantes Cheat Sheet

by juliancnn via cheatography.com/106895/cs/21454/

Operadores Matematicos

`+ | - | * | %`	Suma | resta | multiplicacion | resto (modulo)
`a / b`	Division float(a/b)
`a // b`	Devision entera int(a/b)
`a ** b`	Exponente a^b

Además de int y float, soporta otros tipos de números, como Decimal y Fraction. y numeros complejos usando el sufijo **`j`** o **`J`**

Giladas especiales

`_`	[modo interactivo] última expresión impresa es asignada a la variable _
`pass`	No hace nada, se usa para rellenar algo obligatorio en futura implementacion
`del a[0]` **`del a`**	Elimina el item a[0] Elimina a

Cadena de caracteres

`'cadena'` `"cadena"`	Cadenas con comiilas simplles o dobles
`""" ... cadena multilinea ... """`	Comillas triples para multilinea. Fin de línea son incluídos automáticamente, es posible prevenir esto agregando una \ al final de la línea.
`r"sin \n caracteres especiales"`	r, para raw, cadenas crudas
`str_a + str_a`	Concatenacion

Cadena de caracteres (cont)

`2*str_a`	Repeticion
`str[a:b]`	rebanada desde el indice a (incluido) hasta b sin incluir
`str[:b]`	rebanada parcial hasta b (b excluido)
`str[b:]`	rebanada parcial desde b (b incluido)

`str = 'cadadena negativa'`
`hola[:-5]`
`hola[-5:]`
	'cadadena neg'
	'ativa'

`texto = ('cadenas en ' 'difrentes lineas.')`	Multilinea

Las cadenas son inmutables

Las cadenas de texto se pueden indexar (subíndices), el primer carácter de la cadena tiene el índice 0. No hay un tipo de dato para los caracteres; un carácter es simplemente una cadena de longitud uno.

Loops

`while condicion:`
`    ...sentencias...`
`else:`
`    ...sentencias...`
	else opcional, se ejecuta luego de terminado el bucle

`for element in iterable:`
`    ...sentencias...`
`else:`
`    ...sentencias...`
	else opcional, se ejecuta luego de terminado el bucle

`break`	corta el bucle

Loops (cont)

`continue`	salta a la siguiente iteracion

El continue y el brack funcionan como en c.
Si el break se ejecuta no se ejecuta el else.
Si necesitás modificar la secuencia sobre la que estás iterando mientras estás adentro del ciclo primero hace una copia.

```
for p in palabras[:]: # hace una copia
```

Conjuntos

`canasta = {'manzana', 'naranja', 'manzana', 'pera', 'naranja', 'banana'}`	{'pera', 'manzana', 'banana', 'naranja'}
`'naranja' in canasta`	True
`'yerba' in canasta`	False
`a = set('abracadabra')`	{a', 'r', 'b', 'c', 'd'}
`a - b`	Elementos en a pero no en b
`a & b`	Elementos en a y b
`a ^ b`	Elementos en a y b pero no en ambos

Tambien soporta `a + b`,`a | b`

Es una colección no ordenada y sin elementos repetidos.
Incluyen verificación de pertenencia, eliminación de entradas duplicadas. Soportan operaciones matemáticas como la unión, intersección, diferencia, y diferencia simétrica.
Las llaves o la función set() pueden usarse para crear conjuntos.
Soportada la comprensión de conjuntos.

Diccionario

```
>>> tel = {'jack': 4098, 'sape': 4139}
>>> tel['guido'] = 4127
>>> tel
{'sape': 4139, 'jack': 4098, 'guido': 4127}
>>> tel['jack']
4098
>>> del tel['sape']
>>> tel['irv'] = 4127
>>> tel
{'jack': 4098, 'irv': 4127, 'guido': 4127}
>>> list(tel.keys())
['irv', 'guido', 'jack']
>>> sorted(tel.keys())
['guido', 'irv', 'jack']
>>> 'guido' in tel
True
>>> 'jack' not in tel
False
El constructor dict() crea un diccionario directamente desde secuencias de pares clave-valor:
>>> dict([('sape', 4139), ('guido', 4127), ('jack', 4098)])
{'sape': 4139, 'jack': 4098, 'guido': 4127}
```

Las tuplas pueden usarse como claves si solamente contienen cadenas, números o tuplas; si una tupla contiene cualquier objeto mutable directa o indirectamente, no puede usarse como clave. No podés usar listas como claves

Cheatography

Python3 para principiantes Cheat Sheet

by juliancnn via cheatography.com/106895/cs/21454/

Formateo de datos

`str()`	representaciones legibles por humanos
`repr()`	representaciones que pueden ser leídas por el el intérprete

`repr(x).rjust([n])`
`str.ljust([n])`
`str.center([n])`
	Ajusta

`str.zfill()`	Rellena cadena de numeros con 0
`str.format()`	Las llaves y caracteres son reemplazadas
`print('{1} y {0}'.format('carne', 'huevos'))`	Un número en las llaves se refiere a la posición del objeto pasado en el métod
`print('Esta {comida} es {adjetivo}.'.format( comida='carne', adjetivo='espantosa'))`	Argumentos nombrados en el método, serán referidos usando el nombre del argumento.

Modulos & packeage

`import` *`fibo`*	Importa *fibo.py*. No mezcla el espacio de nombres.
`fibo.nombreMetodo(args)`	llamada a metodo
`from fibo import fib, fib2`	importa fib y fib2 mezcla el espacio de nombres
`__name__`	Nombre del modulo
`from fibo import *`	Importa todaslo que no tengan prefijo _)

Modulos & packeage (cont)

`if __name__ == "__main__":` `...sentencias...`	Se ejecuta solo como script y no como import
`dir([`*`nombreModulo`*`])`	Lista las definiciones de un modulo
`from paquete import submodulo_especifico`	n se recomienda a menos que el módulo que estás importando necesite usar submódulos con el mismo nombre desde otros paquetes.

Si en __init__.py define una lista __all__, se toma como la lista de los nombres de módulos que deberían ser importados cuando se hace from package import *.

Un módulo puede contener declaraciones ejecutables y definiciones de funciones. Las declaraciones son para inicializar el módulo.

Los archivos __init__.py se necesitan para hacer que Python trate los directorios como que contienen paquete

`from .. import` *`nombremod`*	Importa relativo

Errores y excepciones

```
class B(Exception):
  pass
class C(B):
  pass
class D(C):
  pass
for cls in [B, C, D]:
  try:
    raise cls()
  except D:
    print("D")
  except C:
    print("C")
  except B as instanciaExepcion:
    print("B")
  else:
    pass
  finally:
    pass
```

Imprimirá B, C, D, en ese orden.Si las clausulas de except estuvieran invertidas (con except B primero), habría impreso B, B, B.

Una clase en una clausula except es compatible con una excepción si la misma esta en la misma clase o una clase base de la misma

Errores y excepciones (cont)

Una clausula except listando una clase derivada no es compatible con una clase base

El último except puede omitir nombrar qué excepción captura, para servir como comodín

Tienen un bloque `else` opcional,cuando está debe seguir a los except. Se ejecutasi el bloque try no genera una excepción.

Tienen un bloque `finally` opcional,cuando está se ejecuta siempre a la salida sin importar si se lanzo la excepcion

Ambitos

```
def prueba_ambitos():
    def hacer_local():
        algo = "algo local"
    def hacer_nonlocal():
        nonlocal algo
        algo = "algo no local"
    def hacer_global():
        global algo
        algo = "algo global"
    algo = "algo de prueba"
    hacer_local()
    print("Luego de la asignación local:", algo)
    hacer_nonlocal()
    print("Luego de la asignación no local:",
algo)
    hacer_global()
    print("Luego de la asignación global:", algo)
prueba_ambitos()
print("In global scope:", algo)
```

Luego de la asignación local: algo de prueba
Luego de la asignación no local: algo no local
Luego de la asignación global: algo no local
En el ámbito global: algo global

Variables

No existen variables privadas de instancia

Por convención un nombre prefijado con un guión bajo debería tratarse como una parte no pública de la API

Cualquier identificador con 2 guiones bajos *__algo* es textualmente reemplazado por
_nombredeclase__algo
). Se modifica el nombre del identificador sin importar su posición sintáctica, siempre y cuando ocurra dentro de la definición de una clase.

La modificación de nombres es útil para dejar que las subclases sobreescriban los métodos sin romper las llamadas a los métodos desde la misma clase.

```
class Mapeo:
  def __init__(self, iterable):

self.lista_de_items = []
    self.__actualizar(iterable)
  def actualizar(self, iterable):
    for item in iterable:
      self.lista_de_items.append(item)
  __actualizar = actualizar # copia privada del
actualizar() original class SubClaseMapeo(Mapeo):
  def actualizar(self, keys, values):
      # provee una nueva signatura para
actualizar()

# pero no rompe __init__()

for item in zip(keys, values):
      self.lista_de_items.append(item)
```

Iteradores / Generadores

la sentencia for llama a iter() en el objeto contenedor. La función devuelve un objeto iterador que define el método __next__() que accede elementos en el contenedor de a uno por vez. Cuando no hay más elementos, __next__() levanta una excepción StopIteration que le avisa al bucle del for que hay que terminar.

Para implementar un iterador un método __iter__() que devuelva un objeto con un método __next__(). Si la clase define __next__(), entonces alcanza con que __iter__() devuelva self:

Iteradores / Generadores (cont)

Se escriben como funciones regulares pero usan la sentencia yield cuando quieren devolver datos. Cada vez que se llama next() sobre él, el generador continúa desde donde dejó (y recuerda todos los valores de datos y cual sentencia fue ejecutada última).

Expresiones generadoras: Algunos generadores simples pueden ser codificados concisamente como expresiones usando una sintaxis similar a las listas por comprensión pero con paréntesis en vez de corchetes. Estas expresiones se utilizan en situaciones donde el generador es usado inmediatamente por una función que lo contiene. Las expresiones generadoras son más compactas pero menos versátiles que definiciones completas de generadores, y tienden a utilizar menos memoria que las listas por comprensión equivalentes.

Interprete

`python -c` *comando [arg]*	ejecuta las sentencias en *comando* python -c "print(2+2)"
`python -m module [arg]`	ejecuta el código de module como si se hubiese ingresado su nombre completo en la línea de comandos
# -*- coding: encoding -*-	Espeficica codificacion diferente a UTF8 # -- *coding: cp-1252* --
`python -i` *script.py*	Ejecuta el script y entra en modo interactivo

Listas

`list.append (x)`	Agrega un ítem al final de la lista Equivale a a[len(a):] = [x].
`list.extend (iterable)`	Extiende la lista agregándole todos los ítems del iterable. Equivale a a[len(a):] = iterable.
`list.insert (i, x)`	Inserta un ítem en una posición dada.
`list.remove (x)`	Quita el primer ítem de la lista cuyo valor sea x. Es un error si no existe tal ítem.
`list.pop ([, i])`	Quita el ítem en la posición dada y lo devuelve. Si no se especifica un índice, a.pop() quita y devuelve el último ítem de la lista.
`list.clear ()`	Quita todos los elementos de la lista. Equivalente a del a[:]
`list.index (x[, start[, end]])`	Devuelve un índice basado en cero del primer ítem cuyo valor sea x. Excepción ValueError si no existe tal ítem

Listas (cont)

`list.count (x)`	Devuelve el número de veces que x aparece en la lista.
`list.sort (key=None, reverse=False)`	Ordena los ítems de la lista in situ
`list.reverse ()`	Invierte los elementos de la lista in situ.
`list.copy ()`	evuelve una copia superficial de la lista. Equivalente a a[:]

Tipo de datos mutables.
Pueden ser indexadas y rebanadas.
Pueden contener diferentes tipos (mala practica)
Las rebanada devuelve una copia superficial de la lista
Ineficiente para tratar como FIFO, usar 'from collections import deque'

Funciones comunes

`range(end)`
`range(start,end)`
`range(start,end,step)`
	el objeto devuelto por range() se comporta
	como si fuera una lista, pero no lo es.
	Es un objeto que devuelve los ítems sucesivos
	de la secuencia deseada cuando iterás sobre
	él, pero realmente no construye la lista,

`enumerate(`*`iterable`*`)`	Agrega un contador al iterable y devuelve un enumerate
`reversed(`*`iterable`*`)`	Invierte el orden in place
`sort(`*`iterable`*`)`	Ordena

Sobre funciones

La palabra reservada def se usa para definir funciones. Debe seguirle el nombre de la función y la lista de parámetros

La primer sentencia del cuerpo de la función puede ser opcionalmente una cadena de texto literal; esta es la cadena de texto de documentación de la función, o docstring.

La ejecución de una función introduce una nueva tabla de símbolos usada para las variables locales de la función (tabla de símbolos loca)

La referencia a variable mira las tablas en orden:
1) Tabla de simbolos local.
2)Tabla de simbolos local de la funcion externa
3) Tabla de simbolos global
4) Nombres predefinidos

Cadenas de texto de documentación

La primer línea debe ser siempre un resumen corto y conciso del propósito del objeto. No se debe mencionar explícitamente el nombre o tipo del objeto.
Esta línea debe empezar con una letra mayúscula y terminar con un punto.
Si hay más líneas en la cadena de texto de documentación, la segunda línea debe estar en blanco, separando visualmente el resumen del resto de la descripción. Las líneas siguientes deben ser uno o más párrafos describiendo las convenciones para llamar al objeto, efectos secundarios, etc

Argumento de funciones

```
def ventadequeso(tipo, *argumentos,
**palabrasclaves):
    print("-- ¿Tiene", tipo, "?")
    print("-- Lo siento, nos quedamos sin", tipo)
    for arg in argumentos:
        print(arg)
    print("-" * 40)
```

Argumento de funciones (cont)

```
    for c in palabrasclaves:
        print(c, ":", palabrasclaves[c])
ventadequeso("Limburger",
    "Es muy liquido, sr.",
    "Realmente es muy muy liquido, sr.",
    cliente="Juan Garau",
    vendedor="Miguel Paez",
-- ¿Tiene Limburger ? -- Lo siento, nos quedamos sin Limburger Es muy liquido, sr. Realmente es muy muy liquido, sr.
------------------------------------ cliente : Juan Garau vendedor : Miguel Paez puesto : Venta de Queso Argentino
    puesto="Venta de Queso Argentino"
# Otro ejemplo
>>> def loro(tension, estado='rostizado', accion-
='explotar'):
... print("-- Este loro no va a", accion, end=' ')
... print("si le aplicás", tension, "voltios.",
end=' ')
... print("Está", estado, "!")
>>> d = {"tension": "cinco mil", "estado": "dem-
acrado",
... "accion": "VOLAR"}
>>> loro(**d)
-- Este loro no va a VOLAR si le aplicás cinco mil voltios. Está demacrado !
```

Cuando un parámetro formal de la forma **nombre está presente al final, recibe un diccionario conteniendo todos los argumentos nombrados excepto aquellos correspondientes a un parámetro formal.
Puede ser combinado con un parámetro formal de la forma *nombre que recibe una tupla conteniendo los argumentos posicionales además de la lista de parámetros formales. (*nombre debe ocurrir antes de* *nombre).

Lista de comprensión

```
[(x, y) for x in [1,2,3] for y in [3,1,4] if x != y]
[(1, 3), (1, 4), (2, 3), (2, 1), (2, 4), (3, 1), (3, 4)]
# Es equivalente a:
combs = []
for x in [1,2,3]:
    for y in [3,1,4]:
        if x != y:
            combs.append((x, y))
# Printeo
combs
[(1, 3), (1, 4), (2, 3), (2, 1), (2, 4), (3, 1), (3, 4)]
```

Una lista de comprensión consiste de corchetes rodeando una expresión seguida de la declaración for y luego cero o más declaraciones for o if. El resultado será una nueva lista que sale de evaluar la expresión en el contexto de los for o if que le siguen.

Iteracion

```
caballeros = {'gallahad': 'el puro', 'robin': 'el valiente'}
for k, v in caballeros.items():
    print(k, v)
```

Obtener al mismo tiempo
la clave y su valor

```
for i, v in enumerate(['ta', 'te', 'ti']):
    pint(i,v)
```

Obtener al mismo tiempo
indice y valor

```
preguntas = ['nombre', 'objetivo', 'color favorito']
respuestas = ['lancelot', 'el santo grial', 'azul']
    print('Cual es tu {0}? {1}.'.format(p, r))
```

Sobre condiciones

Las condiciones usadas en las instrucciones `while` e `if` pueden contener cualquier operador, no sólo comparaciones.

Los operadores de comparación **`in`** y **`not in`** verifican si un valor está (o no está) en una secuencia.

Los operadores **`is`** e **`is not`** comparan si dos objetos son realmente el mismo objeto; esto es significativo sólo para objetos mutables como las listas.

Todos los operadores de comparación tienen la misma prioridad, la cual es menor que la de todos los operadores numericos

Las comparaciones pueden encadenarse. Por ejemplo, `a < b == c` verifica si a es menor que b y además si b es igual a c.

Las comparaciones pueden combinarse mediante los operadores booleanos and y or, y el resultado de una comparación puede negarse con not. Estos tienen prioridades menores que los operadores de comparación; entre ellos not tiene la mayor prioridad y or la menor, o sea que A and not B or C equivale a (A and (not B)) or C.

Operadores booleanos **`and`** y **`or`** son los llamados operadores cortocircuito: sus argumentos se evalúan de izquierda a derecha, y la evaluación se detiene en el momento en que se determina su resultado.

Cuando se usa como un valor general y no como un booleano, el valor devuelto de un operador cortocircuito es el último argumento evaluado

```
non_nulo = cadena1 or cadena2 or cadena3
non_nulo
'Trondheim'
```

Tuplas y secuencias

Una tupla consiste de un número de valores separados por comas

```
t = 12345, 54321, 'hola!'
# t[0]
12345
t (12345, 54321, 'hola!')
```

Las tuplas son inmutables, perosí se puede crear tuplas que contengan objetos mutables, como las listas.

`x, y, z = t` desempaqueta la tupla

Manejo de archivos

`open('archivo', 'modo')`	devuelve un objeto archivo r/w(overwrite)/a
`with open('archivo') as f:` `  datos_leidos = f.read()`	El archivo es cerrado luego de que el bloque termina, incluso si se generó una excepción.
`f.read()` `f.read(nbytes)`	lee todo el archivo o n bytes
`f.readline()`	Lee una linea (deja \n si existe)
`for linea in f:` `  print(linea, end='')`	Para leer líneas de un archivo, podés iterar sobre el objeto archivo. Esto es eficiente en memoria y rápido
`list(f)` `f.readlines()`	Lee todas las lineas, devuelve una lista
`f.write(str)`	escribe en el archivo

Manejo de archivos (cont)

`f.tell()`	devuelve un entero que indica la posición actual en el archivo representada como número de bytes desde el comienzo del archivo en modo binario y un número opaco en modo texto.
`f.seek(desplazamiento, desde_donde)`	cambiar la posición del objeto archivo.La posición es calculada agregando el desplazamiento a un punto de referencia; el punto de referencia se selecciona del argumento desde_donde. Un valor desde_donde de 0 mide desde el comienzo del archivo, 1 usa la posición actual del archivo, y 2 usa el fin del archivo como punto de referencia. desde_donde puede omitirse, el default es 0

Si se agrega b al modo el archivo se abre en modo binario.
Se debería usar este modo para todos los archivos que no contengan texto ya que cuando se lee en modo texto, por defecto se convierten los fines de lineas que son específicos a las plataformas (\n en Unix, \r\n en Windows) a solamente \n ropiendo si es binario el archivo

Clases

Soporta herencia multiple

(dinamico) Se crean en tiempo de ejecución, y pueden modificarse luego de la creación.

Los miembros de la clase son publicos (En terminologia de c++)

Todas las funciones miembro son virtuales.

No hay atajos para hacer referencia a los miembros del objeto desde sus métodos: la función método se declara con un primer argumento explícito que representa al objeto, el cual se provee implícitamente por la llamada

Como en Smalltalk, las clases mismas son objetos

Los tipos de datos integrados pueden usarse como clases base para que el usuario los extienda.

Hay sobrecarga de operadores

objetos clases, metodos e instancias

una definición de clase, se crea un nuevo espacio de nombres, el cual se usa como ámbito local. y Cuando una definición de clase se finaliza normalmente se crea un objeto clase.

soportan dos tipos de operaciones: hacer referencia a atributos e instanciación.

__doc__ también es un atributo válido, que devuelve la documentación asociada a la clase

x = MiClase(), crea una nueva instancia de la clase

Los miembros pueden ser de existencia dinamica, la clase es mutable

x.f() es exactamente equivalente a MiClase.f(x)

Los atributos de datos tienen preferencia sobre los métodos con el mismo nombre;

El ámbito global asociado a un método es el módulo que contiene su definición. (Una clase nunca se usa como un ámbito global.)

objetos clases, metodos e instancias (cont)

Todo valor es un objeto, y por lo tanto tiene una clase (también llamado su tipo). Ésta se almacena como objeto.__class__.

`def __init__(self):
  self.datos = []`
Cuando se instancia se llama al metodo init de la clase.
usar verbos para los métodos y sustantivos para los atributos.

Herencia

```
class ClaseDerivada(ClaseBase):
```

Las referencias a métodos se resuelven de la siguiente manera:
*se busca el atributo de clase correspondiente,descendiendo por la cadena de clases base si es necesario, y la referencia al método es válida si se entrega un objeto función.

un método de la clase base que llame a otro método definido en la misma clase base puede terminar llamando a un método de la clase derivada que lo haya redefinido. por que todas los metodos son virtuales

Hay una manera simple de llamar al método de la clase base direct-amente: simplemente llamás a ClaseBase.metodo(self, argumentos)