Cheatography

Programming with Python Cheat Sheet

by Padma (padma-it) via cheatography.com/121896/cs/22458/

Python Operators

Operator	Example	Result
Assignment =	a,b,c = 5,6,2 inc=dec=10	
Exponential **	a ** c	25
Unary operators ~ + -	print(-8) print(~7)	Negative number-8 -8;Negate opeartor changes all 1 to 0;0 to 1
Float Division /	a/b	0.8333333333333334
Floor Division //	a//b, b//1	0, 1
Remainder(Modulo) %	a%b, b%a	5, 1
Bitwise operators & | >> << ^	a = 10 = 1010 (Binary) b = 4 = 0100 (Binary)	a&b = 0 a|b=14(1110) a^b=14 a>>1=5 5<<1 = 10
Increment +=	inc += 1	11
Decrement -=	dec -= 1	9
Identity	a=10 b=5+2 c=a	a is b : False a is c: True
Membership	in, not in	
Logical and, or	True or False	True
Boolean True, False	True and False	False

Input and Output

	Example	Result
int(input()) gets input and converts to integer	score = int(input("Enter number: "))	10 20
input() gets input in string	name = input()	Test
ast.literal_eval(input()) gets input in Python data type format	list1 = ast.literal_eval(input())	[1,2,3,4] ('a','e','i') {'a':1,'b':2}
print format	print("{0} {1} using {2}".format("data","analysis","Python"))	data analysis using Python

Conditional Statements

Construct	Example	Result

if...elif...else	if score==100:
	  print("Perfect")
	elif 60<=score<100:
	  print("First Class")
	else:
	  print("Failed")	

for
break
else
	for i in range(1, 4):
	  print(i)
	  if i==2:
	  break
	else:
	  # Executed when no break in for loop
	  print("No Break")
		1
		2

in	my_string = "Mary"
	for alphabet in my_string:
	  print(alphabet)
	my_string = "Mary had a little lamb"
	for word in my_string.split():
	  print(word)
		M
		a
		r
		y
		Mary
		had a
		little
		lamb

while	start =1
	total= 0
	while start<5:
	  total+=start
	  start+=1
	  print(total)
		10

Inbuilt Functions

Function Syntax	Description	Example
abs()	returns the absolute value of number	abs(-7.25) = 7.25
all(iterable)	Check if all items in a list are True	all(True,False) is False all{True,True) is True
any(iterable)	Check if any of the items in a list are True	any(empty_iterable) is False any(True,False) is True

Inbuilt Functions (cont)		
divmod(divident, divisor)	returns a **tuple** containing the quotient and the remainder	print(divmod(5, 2)) is (2,1)
eval(expression, globals, locals)	evaluates the smaller expression and runs it	x = 1 print(eval('x + 1')) is **2**
exec()	executes all size Python code	x = 'name = "John"\nprint(name)' exec(x) is **John**
help()	Gives help content of the function	
id()	returns id of the object	
isinstance()	returns **True** if the specified object is of the specified type	isinstance(5, float, int, str, list, dict, tuple)
issubclass()	returns **True** if the specified object is a subclass	
range(start, stop, step)	**start:** Optional. Integer Specifying at which position to start. Default is 0 **stop:** Required. Integer. Runs till **stop-1** **step:** Optional. Integer number specifying the incrementation. Negative for decrement; Default is 1;	
reversed(sequence)	returns a reversed iterator object	same as list.reverse()
round(number, digits)	returns a floating point number	print(round(5.76543, 2)) is 5.77
sorted(iterable, key=function, reverse=reverse)	**iterable:**Required. The sequence to sort, list, dictionary, tuple etc. **key:**Optional. A Function to execute to decide the order e.g lambda x:x%5 Default is None **reverse**Optional. A Boolean. False will sort ascending, True will sort descending. Default is False	

Inbuilt Functions (cont)

sum(iterable, start)	returns sum of all items in an iterable	sum(list,value) = value+list_elements
type(object, bases, dict)	returns the type of the object	
zip(iterator1, iterator2, iterator3 ...)	an iterator of tuples where the first item in each passed iterator is paired together, and then the second item in each passed iterator are paired together	a = ("A", "B", "C") b = (1, 2, 3, 4) z = zip(a, b) print(list(z)) **OUTPUT**: [('A', 1), ('B', 2), ('C', 3)]

Custom Functions
default return is None
	def f1(*args):
	  print(args)
	  return(sum(args))
		f1(1,2)
		f1(3,4,5,6)

lambda arguments : expression	expression is executed and the result is returne	x = lambda a : a + 10 print(x(5)) **OUTPUT:**15

List Data Structure [ ] - Mutable

Operation	Example	Result
Assignment	lst=['ant','bat','cat',42]	
Creation	my_list = ["Hi"] *5	['Hi', 'Hi', 'Hi', 'Hi', 'Hi']
List concatenation	my_list = ["Hi"] + ["Padma"]	['Hi', 'Padma']
Accessing the list	print(lst)	['ant','bat','cat',42]
Indexing	print(lst[0]) print(lst[-1])	ant 42
Slicing	print(lst[1:2]) print(lst[2:]) print(lst[: -3])	['bat', 'cat'] ['cat',42] ['ant','bat']
Remove Last Element Stack implementation	lst.pop() lst.pop()	42 'cat'
Remove Last Element Queue implementation	lst.pop(0) lst.pop(0)	'ant' 'bat'
Remove any element	lst.remove(42) print(lst)	['ant','bat','cat']
Add new element at the end	lst.append('R') print(lst)	['ant','bat','cat','R']
Add new element at index	lst.insert(3, 'o')	['ant','bat','cat','o','R']

List Data Structure [ ] - Mutable (cont)

Add all items of a list	lst2 = ['dog','fish'] lst.extend(lst2) lst.append(lst2)	['ant','bat','cat','o','R','dog','fish'] ['ant','bat','cat','o','R',['dog','fish']]
Length	len(lst)	5
Sort Elements	print(sorted(lst))	['R', 'ant', 'bat', 'cat', 'o']
Maximum	n=[2,6,9,3,2,1] print(max(n)) print(max(lst, key=len))	 9 ant
Minimum	print(min(nums)) print(min(lst, key=len))	1 'o'
Nested lists	nest = [[1, 2, 3, 4], [5, 6, 7], [8, 9, 10]] print(nest[1]) print(nest[0][2])	[5, 6, 7] 3
List copies	print(id(lst)) l1=lst print(id(l1)) l2= DA.copy() print(id(l2))	1224125667976 1224125667976 1224126395656
Integer List to string	print(''.join(str(e) for e in n))	269321
String list to String	print('&'.join(lst))	ant&bat&cat&o&R

Tuple Data Structure ( ) - Immutable

Operation	Example	Result
Single value tuple creation	tpl = (27,)	(27,)
Repetition	tpl2 = 2*(27,)	(27,27)
Assignment	tpl2 = tpl print(id(tpl)) print(id(tpl2))	 151922812821615192281282 16 1519228128216 1519228128216
Concatenation	t1 = (1, 2, 3) t2=(4, 5, 6) t3=t1+t2	 (1, 2, 3, 4, 5, 6)

Tuple Data Structure ( ) - Immutable (cont)

Convert to Tuple	tup3 = tuple([1,2,3]) print(tup3) tup4 = tuple('Hello') print(tup4)	(1, 2, 3) ('H', 'e', 'l', 'l', 'o')
Indexing	print(tup3[1]) print(tup3[-1])	2 3
Slicing	print(tup4[:3])	('H', 'e', 'l')
Count Elements	print(tup4.count("l"))	2
Get Index of Element	print(tup3.index(2))	1
Membership test	3 in (1, 2, 3)	True
Iteration	for x in (1, 2, 3): print(x,end='')	123
Length	print(len(t3))	6

Dictionary Data Structure { } - Mutable

Operation	Example	Result
Create Empty Dictionary	d1 = {}	{}
Creation	d={'a':1, 'b':2}	{'a': 1, 'b': 2}
Creation using dict	d=dict(a=1, b=2)	{'a': 1, 'b': 2}
Create from tuples and lists	dict([(1,'apple'), (2,'ball')])	{1: 'apple', 2:'ball'}
Mixed keys	d3 = {'name': 'Padma', 1: [2, 4, 3]}	
Create using fromkeys()	keys = {'a', 'e', 'i', 'o', 'u' } vowels = dict.fromkeys(keys) print(vowels)	{'o': None, 'u': None, 'i': None, 'e': None, 'a': None}
Add multiple key,values	d = {'a':1,'c':3} d2 = {'d': 4,'e':5} d.update(d2) print(d)	 {'c': 3, 'e': 5, 'd': 4, 'a': 1}
Add/Change new key,value	d['c'] = 3 **#overwrites value if key exists**	{'a': 1, 'b': 2, 'c':3}
Using key as index	print(d['a'])	1
get(key,default_value)	print(d.get('c','NA'))	NA
Get all keys lazy fn	print(**list**(d.keys()))	['a','b','c']
Get all values lazy fn	print(**list**(d.values()))	[1,2,3]
Delete key,value	del d['b']	{'a': 1, 'c':3}
Remove key,value	print(d.pop('c'))	3

Dictionary Data Structure { } - Mutable (cont)		
Remove any key,value	print(d.popitem())	('a',1)
Delete key,value	del d['a']	
Delete dictionary	del d	
Key Membership Test	print('c' in d) print(3 in d)	True False
Iteration d.keys() d.values() d.items()	for i in d: print(d[i])	Default is keys iterated. Use d.keys() or d.values() Use d.items() to get (key,value)
Sort keys	print(sorted(d))	['a','b','c']
Get number of keys	print(len(d))	3
Delete all key,values	d.clear()	

Sets - Mutable DS of immutable elements		
Operation	**Example**	**Result**
Create Empty Set	set()	set()
Create Non-empty Set	set2 = set(['c','d','e']) set3 = set(['c','f'])	{'c','d','e'} {'c','f'})
List to set	lst=['a','a','b','c'] set1 = set(lst)	{'a','b','c'}
Union	print(set1.union(set2)) print(set1 | set2)	{'a', 'e', 'd', 'c', 'b'} {'a', 'e', 'd', 'c', 'b'}
Intersection	print(set1.intersection(set2)) print(set1 & set2)	{'c'} {'c'}
Difference	print(set1.difference(set2)) print(set1-set2)	{'a', 'b'} {'a', 'b'}
Symmetric difference	print(set1.symmetric_difference(set2)) print(set1^set2)	{'e', 'b', 'a', 'd'} {'e', 'b', 'a', 'd'}
Intersection function does not take in list or tuples of sets but sets itself.	print(set.intersection(set1, set2, set3)) print(set1.intersection(set2, set3))	{'c'} {'c'}
Remove dupliates from a list	print(list(set(lst)))	

Comprehension

Why comprehensions?

> Comprehensions are used to create iterable objects in a simpler and concise fashion.
> They are the complete substitute of for loops, map, reduce or nested loop
> Comprehensions are very compact and can be initialized in a single statement and occupies less space in the memory and it has less execution time

Types of comprehensions:

> 1. List comprehension
> 2. Nested list comprehension
> 3. Dictionary comprehension
> 4. Set comprehension
> 5. Generator comprehension

1. LISTS COMPREHENSION:

> **Syntax:**
> output_list = [output_exp for var in input_list if (var satisfies this condition)]
>
> **Example:**
> list_using_comp = [var x 2 for var in range(1, 10)]
>
> Each time var is picked from iterable i.e. range here and the operation var x 2 is performed and the final value is added to the list.
>
> **OUTPUT:**
> [2, 4, 6, 8, 10, 12, 14, 16, 18]

2. Nested List comprehension

> **Syntax:**
> [inner_list_element for outer_list_element in outer_list for inner_list_element in outer_list_element]
>
> **Example:**
> [z for x in y for z in x.split()]
>
> **Without nested list comprehension:**

```
for x in y:
  for z in x.split():
    a.append(z)
```

3. Dictionary Comprehensions:

> **SYNTAX:**
> output_dict = {key:value for (key, value) in iterable if (key, value satisfy this condition)}
>
> **Example:**
> dict_using_comp = {var:var xx 3 for var in input_list if var % 2 != 0}
> Similar to above, var is picked from input_list and is set as key. The value is assigned by the result of var xx 3
>
> **OUTPUT:**
> {1: 1, 3: 27, 5: 125, 7: 343}

4. Set Comprehensions:

This is similar to dict comprehension as it will have {}
But differs in the fact, retaining set principle, does not add duplicates.

EXAMPLE:

```
input_list = [1, 2, 3, 4, 4, 5, 6, 6, 6, 7, 7]
set_using_comp = {var for var in input_list if var % 2 == 0}
```

OUTPUT:

```
{2, 4, 6}
```

5. Generator Comprehensions:

One difference between this and list comprehension is that generator comprehensions use ().
Generators don't allocate memory for the whole list.
Instead, they generate each value one by one which is why they are memory efficient.
List comprehensions impact performance for huge data.

EXAMPLE:

```
input_list = [1, 2, 3, 4, 4, 5, 6, 7, 7]
output_gen = (var for var in input_list if var % 2 == 0)
print(output_gen)
```

OUTPUT:

```
2 4 4 6
```

Maps, Filters and Reduce

1. What is a map?

Map applies a function to all the items in an input_list

Syntax

```
map(function_to_apply, iterable)
```

Maps, Filters and Reduce (cont)

Why do we need maps?

Most of the times we want to pass all the list elements to a function one-by-one and then collect the output.
items = [1, 2, 3, 4, 5] squared = [] for i in items: squared.append(ixx2)

Map implementation:

squared = list(map(lambda n: nxx2, items))

Advantages:

Instead of a list of inputs we can even have a list of functions

Return Type

lazy function; list() or tuple() to be used to list or tuple of items.

2. What is a filter?

To filter some elements what we want and remove what we dont want

Syntax

filter(function or None, sequence)

Why do we need filters?

filter resembles a for loop but it is a builtin function and faster

Filter implementation:

less_than_zero = list(filter(lambda x: x < 0, number_list))

Return Type

lazy function; list() or tuple() to be used to list or tuple of items.

3. What is a reduce function?

Reduce is a really useful function for performing some computation on a list and returning the result.

Why do we need reduce?

If you want to compute the product of a list of integers. So the normal way you might go about doing this task in python is using a basic for loop

reduce implementation:

functools.reduce((lambda x, y: x * y), [1, 2, 3, 4])

Return Type

Returns final element.

String Data Structure - Mutable

Operation	Example	Result
String notation	w1='Hello' w2=" world " w3 = """Good morning"""	single ('), double (") and triple ('" or """) Same type of quotes in start and end
Multi-Line Statements	str1 = w1+ \ w2 + \ w3	*Hello world Good morning*
Comments notation	# Comment	Hash sign (#) that is not inside a string literal

String Data Structure - Mutable (cont)

Multi line comments	''' Comments '''	Triple-quoted string
Indexing [ ]	print(str1[1]) print(str1[-5])	e r
Slicing[:]	print(str1[1:5]) print(str1[:5]) print(str1[1:])	ello Hello ello world Good morning
Update String	str1 ="Maths"	Reassign is **allowed**
Concatenation	print(w1 + w2)	Hello world
Repetition	print(w1*2)	HelloHello
Membership in, not in	H in w1 e not in "Hello"	True False
Raw String r/R	print('Hi\nHello') print(r'Hi\nHello')	Hi Hello Hi\nHello
String format opearator%	print("Scores %s - %d" %(str1,90)) print("List: %s" % [1,2,3]) print('%6.2f' % (3.141592653589793,))	Scores Maths - 90 List: [1, 2, 3] 3.14
ASCII to Unicode	print(ord("A"))	65
Unicode to ASCII	print(chr(65))	A
**Change cases*	print(w1.lower()) print(w2.upper()) print(str1.title())	hello WORLD Hello World Good Morning

Miscellaneous