AMP:CS Python Toolbox - 2024

Curated collection of Python essentials

Data Types

```
bool    #values that are True or False
int     #integers of unlimited length
float   #decimal, imprecise, scientific notation
list    #Mutable ordered sequence of data
str     #Immutable ordered sequence of characters
range   #Immutable ordered sequence of numbers
tuple   #Immutable ordered sequence of data
set     #Unordered collection of unique immutable data
dict    #Unordered collection of key:value mappings
```

Type Conversion and Checking

```
str(val), int(val), float(val)
list(str1), tuple(list1), set(list1)
ord("A")/char(65)          #ASCII value conversion
bin(intVal), hex(intVal)   #base conversion
(3.2).is_integer()         #integer check
isinstance(val, str)       #general type check
str2.isalpha()             #only a-z or A-Z?  → True
str3.isdigit()             #only 0-9? → True
```

Truthy/Falsey

Truthy values evaluate to `True` in Boolean expressions:

```
True                #constants defined to be true
1, -1, 5            #Non-zero numeric values
[1, 2, 3], "false"  #Non-empty sequences
{"Name" : "Kanye"}  #Non-empty collections
```

Falsey values evaluate to `False` in Boolean expressions:

```
False, None           #constants defined to be false
0, 0.0, 0j            #zero of any numeric type
"", (), [], range(0) #empty sequences
{}, set()             #empty collections
```

Assignment Statements

```
n = 0               #single value assignment
i = j = 75          #multiple-target assignment
n += 3              #augmented assignment
a, b, c = "NYC"     #sequence assignment
l, *w = "NYC"       #sequence unpacking
```

Viewing variable values

```
locals()/globals()  #dictionary of symbol tables
```

Comments

```
# This is a comment
```

```
"""
This is a
multi-line comment
"""
```

Operator Precedence

`( )`	Parentheses
`**`	Exponent
`+, -, ~`	Unary plus, unary minus, bitwise NOT
`*, /, //, %`	Multiplication, Division, Floor Division, Modulus
`+, -`	Addition, Subtraction
`<<, >>`	Bitwise shift
`&`	Bitwise AND
`^`	Bitwise XOR
`|`	Bitwise OR
`==, !=, >, >=, <, <=, is, is not, in, not in`	Comparisons, Identity, Membership
`not`	Logical NOT
`and`	Logical AND
`or`	Logical OR
`= %= /= //= -= += *= **=`	Assignment

Sequences: `str`, `list`, `tuple`, `range`

Creating/Concatenating Strings

```
str1 = "algorithm"/'puzzles'
str2 = "a" * 2   #"aa"
str3 = f'{str1} {str2}!'
str4 = str1 + "," + str2
str5 =" ".join(list1)
```

Useful String Methods: returns a new string re: immutability

```
len(str1)
str1.upper()/lower()
str1.isupper()/islower()
str3.strip()
str3.count(" ")
str2.index("z")
.split(",")
```

Creating/Combining Lists/Tuples

```
list1 = ['abcd', 786 , 2.23, True, 70.2] #heterogenous
list2 = list1.copy()    #avoids aliasing
list3 = list1 + ["NYC", 2022]  #appends to first list
list4 = [0] * 4   #[0,0,0,0]
tuple1 = ('abcd', 786 , 2.23, 'True', 70.2)  #immutable
tuple2 = ("new york",)    #1 item tuple needs comma
```

Useful list Methods

```
sum(list1), min(list1), max(list1)
list1.index('abcd')  #also tuples
list1.sort()
len(list1), len(tuple1)
```

```
list1.insert(2, 'a')
list2.append('a')
list3.remove('NYC')
list1.reverse()
```

Return a new sorted list version of an ordered collection

```
sorted(seq1)          sorted(seq1, reverse=True)
```

Slicing a Sequence:

`#[start:stop:step]  returns [start, stop)`

`seq1[2]`	`seq1[2:5]`	`seq1[2:]`	`seq1[-1]`
`seq1[::2]`	`seq1[5:2:-1]`	`seq1[-4::-1]`	`seq1[:-1]`

Unordered Collections: `set, dict`

Creating Sets: Elements must be immutable and unique

```
set1 = {"abc", 34, True, 40, "male", (1, 2, 3)}
set2 = set1.copy()  #avoids aliasing
set3 = set(list1)
```

Useful Set Methods

```
set1.add('a')
set1.remove('a')
set1.update(set2)
```

```
set1.union(set2)
set1.difference(set2)
set1.intersection(set2)
```

Creating Dictionaries: Keys must be immutable and unique

```
dict1 = {} #empty dict
dict1['one']= 1
dict1[2]= "two"
```

```
dict2 = {
 'name':'john',
 'addr': {
  'city': 'Manhattan',
  'state': "New York"
 }
}
```

Useful Dictionary Methods

```
dict1.keys()
dict1.values()
```

```
dict2.items()
del dict2['name']
```

Selection

```
if b > a:
  print("b is greater than a")
elif a > b:
  print("a is greater than b")
else:
  print("a and b are equal")
```

Combining boolean/logical/membership operators

```
if a > b and a not in badNumbers:
  print("a is bigger than b and in the list")
```

Match statements: simplify complex selection structures

```
match color:
  case "RED":
     return "#FF0000"
  case "GREEN":
    return "#00FF00"
  case "BLUE":
    return "#0000FF"
  case _:
    return "#000000"
```

Iteration

Variable-repetition

```
count, n = 0, -1
while (n <= 0):
  n = float(input('Enter a positive number'))
  count=count+1
```

Iterating over Elements in a Collection

```
for x in range(2, 30, 3):
  print(x)
```

```
fruit = ["a1", "b2", "c3"]
for x in fruit:
 print(x)
```

Accessing the Index of an Ordered Sequence

```
fruit = ["apple", "banana", "cherry", "banana"]
for index, value in enumerate(values, start=2):
   print(index, value)
```

Iterating over keys/values in a Dictionary

```
for value in dict1.keys():
   print(dict1[value])
```

Loop Control Statements

```
i = 1
while i < 10:
  i += 1
  if i == 7: break    #goto first line after loop body
  if i == 3: continue #skip remaining lines in loop
  if i == 5: pass     #completes syntax, no effect
  print(i)
```

List comprehensions

```
fruit = ["apple", "banana", "cherry", "mango"]

a_basket = [x for x in fruit if "a" in x]  #filter

backwards_basket = [f[::-1] for f in fruit] #map
```

Programmer-defined Functions

Defining a function

```
def nameFormatter(fname, lname="Doe"):
  return fname.capitalize() + " " + lname.capitalize()
```

Unknown non-keyword/keyword arguments

```
def mysteryArgs(*args, **kwargs):
  for arg in args: print(arg)
  for key, value in kwargs.items(): print(key, value)
```

Returning multiple values

```
def getSmallestAndBiggest(seq):
  sortedSeq = sorted(seq)
  return sortedSeq[0], sortedSeq[-1]
```

Functions as variables

```
def orderReceipts(items):
  return len(items)  #define numeric equivalent

cars.sort(key=orderReceipts)
```

Specifying global scope within a function

```
c = 0
def add():
    global c
    c = c + 2 # increment by 2

add()
print(c)   #prints 2
```

Libraries and Built-In Functions

Math

```
import math
math.pi, math.sqrt(64), math.ceil(1.4),
math.floor(1.4)
```

Random Numbers

```
import random
random.uniform(20, 60)  #[start, stop], decimals
random.randint(3, 9)  #[start, stop], integers
random.choice(list1)  #random element in list
```

Permutations and Combinations

```
from itertools import combinations, permutations
letters = list("Hello")
permutations(letters)
combinations(letters, 4)
```

Input, Output, Files

```
n = input("message")    # input is returned as a string
print(value)

f = open("file1.txt")      #returns a file object
f.read()                   #content as one string
f.readlines()              #content as list of lines

f = open("file1.txt", "a") #a for append
f.write("More content!")   #added to end of file
f.close()                  #tells OS you're done
```

Command Line Arguments

```
import sys              #python3 prog.py test1 test2
for arg in sys.argv:    # [test1, test2]
  print(arg)
```

json

```
import json
str_version = json.dumps(dict_data)
obj_version = json.loads(string_data)
```

HTTP Requests

```
import requests
requests.get('https://example.com/')
```