Cheat Sheet For Python

Variables and Strings

Variables are containers for storing data values.

Creating

a = 3

b = ”hello”

a,b,c = ”pen”,”paper”,”book”

No need to declare with a particular data type

Casting

a = int(4)

b = str(4)

c = float(4)

Type

print(type(a))

Case Sensitive

Variable Names

Start with- letter or underscore character

Contain-(A-z,0-9 and_)

Cannot start with numbers

Strings

Collection of characters

Concatenation

name = ‘john’

place = ‘london’

address = name+’ ’+place

Fetching

print(name(0))

print(name(-1))

>>o

Print(place(1:3))

>>on

List

List are used to store multiple items in a single variable.

Mutable, ordered series, traditionally of the same type of object.

Advantages: Mutable and ordered. Easy to understand. Relatively efficient memory usage.

Disadvantages: Searching is O(n).

Creating

numbers=[1,4,6,23,9]

Fetching

numbers[0]

>>1

Different type of data

values = [5,3,'Jack',25.3]

Adding items to a list

numbers.append(60)

>>[1,4,6,23,9,60]

numbers.insert(2,10)

>>[1,4,10,6,23,9,60]

Remove the element

numbers.pop(1)

>>[1,6,23,9,60]

If we use pop without index value by using the concept of stack it delete the last elemen

numbers.pop()

>>[1,6,23,9]

Delete multiple values

del numbers[1:]

>>[1]

numbers.extend()

Replacing elements

numbers[1]=8

>>[1,8,6,23,9]

Inbuilt functions

min()

max()

sort()

reverse()

join()

index()

count()

clear()

copy()

Conversion from List To String

list1 = [‘c’ , ’java’ , ’python’ , ’js’]

>>print(‘ , ’.join(list1))

>> c, java, python, js

Conversion from String To List

sentence = “cheat sheet for python”

>> print(sentence.split())

>> [‘cheat’ ‘sheet’ ‘for’ ‘ python’]

Tuple

Tuples are used to store multiple items in a single variable.

Immutable, ordered series traditionally containing different objects

Advantages: Immutable and ordered. Relatively efficient memory usage (more than lists).

Disadvantages: Searching is O(n). Hard to understand for many Python newcomers.

Creating

values = (1,8,3,9,4)

Fetch

values[1]

>>8

Cannot change the value

values[1] = 40 !error

Iteration is faster in tuple as compared to list

Set

Mutable, unordered, unique objects. Elements must be hashable.

Advantages: Searching is O(1). Lots of useful methods.

Disadvantages: Not ordered. Elements must be hashable.

Creating

s = {12,5,7,3,2}

Print

>>s

>>{2,3,5,7,12} not in sequence

Indexing

s[2] not supported

Dictionary

Dictionaries are used to store data values in key:value pairs.

Mutable, unordered pairs (keys and values) of objects. Keys must be hashable.

Advantages: O(1) searching for keys. Makes it easy to create trees and other hierarchical data structures. Can be used to create self-documenting code. Many problems can be described in terms of key-value pairs.

Disadvantages: Only lookup by key. Uses more memory than lists and tuples. Keys must be hashable.

Creating

data = {1:'Ritz',2:'John',5:'Jack'}

Fetching

data[2]

>>John

data.get(1)

>>Ritz

Add a key-value pair

data[4] = Chris

>> data = {1:'Ritz',2:'John',5:'Jack',4:'Chris'}

Remove

del(data[2])

>>{1:'Ritz',5:'Jack',4:'Chris'}

Merge 2 list in a dictionary

a = ['Ritika','Abhi','Reena']

b = ['python', 'java','js']

data = dict(zip(a,b))

>>data

{'Ritika': 'python','Abhi': 'java','Reena': 'js'}

Data Types

Text Type:	str
Numeric Types:	int, float, complex
Sequence Types:	list, tuple, range
Mapping Type:	dict
Set Types:	set, frozenset
Boolean Type:	bool

Binary Types: bytes, bytearray, memoryview

- Range

  range(10)

  >>range(0,10)

  list(range(10))

  >>[0,1,2,3,4,5,6,7,8,9]

Operators

Operators are used to perform operations on variables and values.

- Arithmetic operators
- Assignment operators
- Comparison operators
- Logical operators
- Identity operators
- Membership operators
- Bitwise operators

Python Conditons and Statements

Python supports the usual logical conditions from mathematics:

- Equals: a == b
- Not Equals: a != b
- Less than: a < b
- Less than or equal to: a <= b
- Greater than: a > b
- Greater than or equal to: a >= b

If statement:

```
a = 33
b = 200
if b > a:
  print("b is greater than a")

>> b is greater than a
```

elif statement:

The elif keyword is pythons way of saying "if the previous conditions were not true

```
a = 33
b = 33
if b > a:
  print("b is greater than a")
elif a == b:
  print("a and b are equal")
```

```
>> a and b are equal
```

else statement:

The else keyword catches anything which isn't caught by the preceding conditions.

```
a = 200
b = 33
if b > a:
  print("b is greater than a")
elif a == b:
  print("a and b are equal")
else:
  print("a is greater than b")
```

```
>> a is greater than b
```

If-Not-statement:

Not keyword let’s you check for the opposite meaning to verify whether the value is NOT True:

new_list = [1, 2, 3, 4]

x = 10

if x not in new_list:

print(“‘x’ isn’t on the list, so this is True!”)

Pass statement:

If statements can’t be empty. But if that’s your case, add the pass statement to avoid having an error:

a = 33

b = 200

if b > a:

pass

Python Loops

- `while` loops
- `for` loops

while loop

With the while **loop** we can execute a set of statements as long as a condition is true.

```
i = 1
while i < 6:
  print(i)
  i += 1
```

for loop

A for loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string).

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  print(x)
```

How to break a loop

You can also stop the loop from running even if the condition is met. For that, use the break statement both in while and for loops:

i = 1

while i < 8:

print(i)

if i == 4:

break

i += 1

Function

A function is a block of coded instructions that perform a certain action. Once properly defined, a function can be reused throughout your program i.e. re-use the same code.

First, use def keyword followed by the function name():. The parentheses can contain any parameters that your function should take (or stay empty).

def name():

Next, you'll need to add a second code line with a 4-space indent to specify what this function should do.

```
def name():
        print("What's your name?")
```

Now, you have to call this function to run the code.

name.py

```
def name():
        print("What's your name?")
hello()
```

Now, let's take a look at a defined function with a parameter — an entity, specifying an argument that a function can accept.

```
def add_numbers(x, y, z):
 a = x + y
b = x + z
c = y + z
print(a, b, c)
add_numbers(1, 2, 3)
```

In this case, you pass the number 1 in for the x parameter, 2 in for the y parameter, and 3 in for the z parameter. The program will that do the simple math of adding up the numbers:

Output:

```
a = 1 + 2
b = 1 + 3
c = 2 + 3
```

How to Pass Keyword Arguments to a Function

A function can also accept keyword arguments. In this case, you can use parameters in random order as the Python interpreter will use the provided keywords to match the values to the parameters. Here's a simple example of how you pass a keyword argument to a function.

```
# Define function with parameters
def product_info(product name, price):
        print("productname: " + product name)
        print("Price " + str(dollars))


# Call function with parameters assigned as above
product_info("White T-shirt", 15 dollars)
```

```
# Call function with keyword arguments
 product_info(productname=”jeans”, price=45)
```

Output

```
Productname: White T-shirt
 Price: 15
 Productname: Jeans
Price: 45
```

Python Arrays

An array is defined as a collection of items that are stored at contiguous memory locations

Array Representation

An array can be declared in various ways and different languages. The important points that should be considered are as follows:

- Index starts with 0.
- We can access each element via its index.
- The length of the array defines the capacity to store the elements.

Array operations

- **Traverse** - It prints all the elements one by one.
- **Insertion** - It adds an element at the given index.
- **Deletion** - It deletes an element at the given index.
- **Search** - It searches an element using the given index or by the value.
- **Update** - It updates an element at the given index.

Create array by importing

```
from array import *
arrayName = array(typecode, [initializers])
```

Accessing array elements

```
import array as arr
a = arr.array('i', [2, 4, 6, 8])
print(array1[0])
```

>> First element: 2

Changing element

```
a[0] = 0
```

>>[0,2,4,6,8]

Deletion

```
del a[2]
```

>>[0,2,6,8]

Find length of array

```
len(array_name)
```

Concatenation

```
a=arr.array('d',[1.1 , 2.1 ,3.1,2.6,7.8])
b=arr.array('d',[3.7,8.6])
c=arr.array('d')
c=a+b
print("Array c = ",c)
```

>> Array c= array('d', [1.1, 2.1, 3.1, 2.6, 7.8, 3.7, 8.6])

Class and objects

Almost everything in Python is an object, with its properties and methods.

A Class is like an object constructor, or a "blueprint" for creating objects.

Create a Class

```
class MyClass:
  x = 5
```

Create object

```
a = MyClass()
print(a.x)
```

```
>> 5
```

Object Methods

Objects can also contain methods. Methods in objects are functions that belong to the object.

```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

  def myfunc(self):
    print("Hello my name is " + self.name)

p1 = Person("John", 36)
p1.myfunc()
```

```
>> Hello my name is
```

Update object properties

```
>> p1.age = 40
```

Delete object properties

```
>> del p1.age
```

Delete

```
>> del p1
```