Beginner's Python Cheat Sheet

DeepFlow

General

Interpreted | Portable | High-level

- Python supports object-oriented programming and structured programming.
- Python is a general-purpose programming language that can be used in many applications: web development, software development, automation, machine learning, AI, ...

Syntax & Semantics

Syntax

Python is a popular language because of its elegant syntax structure.

```
club = "DeepFlow"
print("Welcome to", club)
# Welcome to DeepFlow
```

Comments

```
# This is an inline comment
""" This a multi-line comment """
```

Indentation

Python uses whitespace indentation to delimit blocks.

```
if 2>1:
    print("2 is bigger than 1.")
```

Variables

Python decides the data type of a variable based on the value assigned. The data type changes dynamically after reassigning a value of another type.

```
age = 25
print(age) # outputs: 25
```

Naming Conventions

A variable name must start with a letter or the underscore character. A variable name cannot start with a number. A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _). Variable names are case-sensitive.

Basic Data Types

Integer

An integer is a positive or negative number without floating point (e.g. 3).

```
x, y = 5, 2
z = int(2)
print(x + y) # 7
print(x // y) # 2
print(x % y) # 1
print(x ** y) # 25
print(int(3.14)) # 3
print(type(x)) # <class 'int'>
print(type(z)) # <class 'int'>
```

Float

A float is a positive or negative number with floating point precision (e.g. 3.1415).

```
x, y, z = 6.5, 3, float(1.5)
print(x // y) # 2.0
print(type(z)) # <class 'float'>
print(float(y)) # 3.0
print(type(x)) # <class 'float'>
```

String

A string is a sequence of characters. Python doesn't have a char data type.

```
a, b, c = "Deep", 'Flow', str(1.5)
print(type(a)) # = <class 'str'>
print(type(c)) # <class 'str'>
print(a + b) # = DeepFlow
print(a[0]) # = D
print(a[0:3]) # = Dee
print(a[-1]) # p
year = str(1912)
print(type(year)) # <class 'str'>
print(len(year)) # 4
```

Boolean

The boolean data type is a truth value, either True or False.

```
x, y, z = True, False, bool(1)
print(x and not y) # True
print(not x and y or x) # True
print(type(x)) # <class 'bool'>
print(z) # True
```

Data Structures

List

A list is an **indexed** container data type that stores a sequence of elements of different types. Lists are **mutable**: modification is possible.

```
a = [5, "d"]
print(a[0], a[1]) # 5 d
print(len(a)) # 2
a.append("e") # [5,'d','e']
a.insert(2,'e') # [5,'d','e','e']
a = a+['p'] # [5,'d','e','e','p']
a.remove(5) # ['d','e','e','p']
```

Set

A set is an **unordered** collection of elements of different types. Sets are **not indexed**.

```
x = {1, "d"}
print(type(x)) # <class 'set'>
x.add(2) # 2 gets added at a
    random position
x.remove(1)
y = set([1,2,3,2])
print(y) # {1, 2, 3}
print(y[0]) # TypeError
```

Tuple

Tuple is an **ordered** sequence of objects same as a list. Tuples are **immutable**. Tuples are used to write-protect data and are usually faster than lists.

```
a = (1,'D',9, 'F')
b = tuple(['e'])
print(type(a)) # <class 'tuple'>
print(a[0]) # 1
c = a + b # (1,'D',9,'F','e')
print(type(c)) # <class 'tuple'>
print(len(c)) # 5
```

Dictionary

Dictionary is an **unordered** collection that stores (key, value) pairs. Keys must be **unique**.

```
d = {'b':50, 1:'20'}
print(d.keys())
# dict_keys(['b', 1])
print(d.values())
# dict_values([50, '20'])
d['a'] = 5 # {'b':50,1:'20','a':5}
d.pop('b') # {1:'20','a':5}
```

Conditional Statements

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 .. else

```
a, l = 5, [1,2,5]
if a in l:
    print("in list")
else:
    print('not in list')
# in list
```

if .. elif .. else

```
a, b = 5, 3
if a*b > 0: print("postive")
elif a*b < 0: print("negative")
else: print("no sign")
# positive
```

Loops

for

For loops can be used to iterate over the elements of a data structure. Usually, it is used when the number of iterations is known.

```
s = ""
for i in ["d","e","e","p", "f"]:
    s = s + i
print(s) # deepf
# List comprehension
l = [(x + 1) for x in [1, 2 ,3]]
print(l) # [2, 3, 4]
```

while

While loops executes its body as long as the loop condition is True.

```
z = 7
while z < 12:
    z += 1
print(z) #12
```