MY PROGRAMMING CONCEPTS CHEATSHEET [PYTHON 3.x]

DATA TYPES

```
type(4)                 # int
type(5.6)               # float
type("string")          # str. Immutable
type( (4,5,6) )         # tuple. Immutable
type([1, 2, 3])         # list. Mutable
type({"key":"val"})     # dict.Mutable
type({1:"value"})       # dict
```

VARIABLES

Stores a piece of information at a time

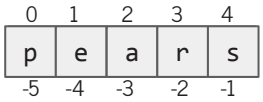
```
myName = "Ain"
mySite = "www.ainlovescode.com"
```

ARITHMETIC OPERATORS

```
>>> 5+2                 # add
7
>>> 5-2                 # subtract
3
>>> 5//2                # divide & round down
2
>>> 5/2                 # divide & show float result
2.5
>>> 5%2                 # remainder
1
>>> 5*2                 # multiply
10
>>> 4**2                # exponent OR power of
25
```

INDEX / INDICES

Usually, the first / leftmost / top item has index 0. The last / rightmost / bottom item has index -1.




```
word = "pears"
print(word[3])          # "r"
print(word[-4])         # "e"
```

STRING FORMATTING

```
print("My name is " + myName)
print("My name is {}".format(myName))

print("Name:{}; URL:{}".format(myName, mySite))
print("Name:{1}; URL:{0}".format(mySite, myName))
```

CHANGING LISTS

```
mList1 = [1,2,3]
mList2 = [5,6]

mList1.append("e")      # insert "e" as last item
mList1.insert(1,9)      # insert 9 at index 1
mList1.pop()            # pop out & show last item
mList1.pop(2)           # pop out & show item at
                        # index 2

mList1.extend(myList2)          # insert myList2
                                # at end of mList1
```

DICTIONARY FORMATTING

A dictionary is represented by key-value pairs with curly brackets like: {key:value}

```
squares= {0:0, 1:1, 2:4}

squares[5] = 25         # {0:0, 1:1, 2:4, 5:25}
squares.keys()          # dict_keys([0, 1, 2, 5])
squares.values()        # dict_values([0, 1, 4, 25])
del squares[5]          # {0:0, 1:1, 2:4}
```

BOOLEAN OPERATORS

Compares left and right sides of operator

```
<     # less than
>     # more than
<=    # less than/equals to
>=    # more than/equals to
==    # equals to / same as
!=    # not equals to
```

IF STATEMENTS

Checks for a certain condition, runs the inner code once

```
if age < 13:
    print("G only")
elif 13 <= age <18:
    print("PG13, G")
else:
    print("All movies")
```

FOR LOOPS

Runs the inner code a certain number of times

```
for i in range(0,10,2):
    print(i)            # 0, 2, 4, 6, 8

for i in range(6):
    print(i)            # 0, 1, 2, 3, 4, 5

for line in ["line1", "line2"]:
    print(line)         # line1, line2
```

WHILE LOOPS

Runs the inner code until condition is false

```
while isRaining:
    print("Bring an umbrella")

currentValue = 3
while currentValue > 0:
    print(currentValue)
    currentValue -= 1   # 3, 2, 1
```

FUNCTIONS

A shorter program that will be called multiple times, simplifies main program

```
def myFunction0():
    print("Print this")

def myFunction1(param = "some"):
    print(param)

def myFunction2(value):
    if value < 0:
        return "Negative"
```

LAMBDA FUNCTIONS

A one-line function that can take many parameters but only <u>one</u> expression / action

```
vol = lambda w,h,b: return w*h*b
print("The final volume of the tank is " +
vol(2,3,4))
```

MY PROGRAMMING CONCEPTS CHEATSHEET [PYTHON 3.x]

CLASSES

Stores the behaviour (methods) and identifying information (attributes). In this example, VIP is a child class of Person.. VIP inherits methods and attributes from Person.

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

    def hello(self):
        print("Hi! My name is" + self.name)

class VIP(Person):
    def __init__(self, name, title):
        super().__init__(name)
        self.title = title
    def hello(self):         #overwrites
Person.hello()
        print("Hello everyone. I'm " + self.title +
self.name)

todaysVIP= VIP("Ming", "Mr.")
todaysVIP.hello()
```

ZEN OF PYTHON

A set of design principles for Python programs, which arguably boosts readability and comprehension of your code! A more detailed analysis can be found here.

```
>>> import this
Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the
rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to
guess.
There should be one-- and preferably only one
--obvious way to do it.
Although that way may not be obvious at first
unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a
bad idea.
If the implementation is easy to explain, it may be
a good idea.
Namespaces are one honking great idea -- let's do
more of those!
```

USEFUL LINKS

- Python Documentation - Explains the syntax and behavior of Python functions and inbuilt libraries. Offline version here
- Python questions - Stack Overflow - Search results for just about any question on Python programming and errors
- Python Challenges - HackerRank - Challenges of increasing difficulty for Python. The main site also works as a portfolio website to answer user questions and set up a portfolio.

MY NOTES

Add your own notes here! You can scribble down common mistakes you make, other useful links or even your very own programming goals.