Python 3 Cheat Sheet

This cheat sheet aims to be a reference for the main syntax rules of Python 3 that are covered in the online course Fundamentals of Programming.

VARIABLES

Variables are reserved spaces in the memory that can hold any type of content.

Creating a variable

```
age = 36
```

Using a variable

```
age *  365
```

TIP: To make our code more understandable and clear, the identifier (variable name) must reflect the use we give it within the program.

DATA TYPES

All the values that appear in a program have a type. These are some basic Python data types.

Integer (integers are those positive or negative numbers that do not have decimals)

```
a = 5 ; x = -100 ; b = -62 ; y = 555
```

Float (numbers with decimals. The whole part is written first, then the period, then the decimal)

```
x= 1.22    --> the number 1.22 to x
a= -5.6    --> the number -5.6 to a
```

String (values representing alphanumeric text are called strings and have the type str. They can be defined with text in single or double quotes)

```
"Hello World!"

'February'

# We assign the str "FUNDAMENTALS" to a
a = "FUNDAMENTALS"
```

NOTE: Everything that starts with a # is considered a comment and is *not* executed by Python: It's there for humans only!

Boolean (the values `True` (true) and `False` (false) are of type bool and represent logical values. Numerically the `False` equals `0` and any other number equals `True` but its default is `1`)

```
isSunny = True
```

OPERATORS

Arithmetic Operators

```
+     Sum
-     Subtraction
*     Multiplication
**    Exponent
/     Division
//    Integer division
%     Module or Residue
```

Examples

```
365 + 1 - 2    --> 364
25*9/5 + 32    --> 77
2**8           --> 256
```

Chain operators

```
+     Concatenation
*     Repetition
```

Examples

```
"Hello" + "World" --> "HelloWorld"
"Grüezi"*2 --> "GrüeziGrüezi"
```

Relationship operators

```
==   a == b (a equal to b?)
!=   a != b (a different from b?)
>    a >  b (a greater than b?)
<    a <  b (a less than b?)
>=   a >= b (a great. or equal to b?)
<=   a <= b (b less or equal to b?)
```

examples

```
"hello" == 'hello' --> True
"hello" != 'Hello'    --> True
5 <= 3                --> False
2 * 9 == 6            --> False
```

Logical operators

```
or  a or b   (Any of a or b True?)
and a and b  (Are a and b both True?)
not not x    (The opposite of x)
```

examples

```
True or True      --> True
True and True     --> True
True and False    --> False
True or False     --> True
```

FUNCTIONS

A function is a block of code with a name. It can (1) receive some arguments as input, (2) perform a series of tasks, and (3) return a value. This block can be called when needed. If we need to create our own function we have to define it, for that we use the sentence def.

NOTE: The definition of a function does not execute the body of the function; this is executed only when the function is called!

Function definition

```
def FUNCTION_NAME (PARAMETERS):
  STATEMENTS
  return EXPRESSION
```

- `PARAMETERS`: is the data or information that the function receives
- `STATEMENTS`: are the sentences in code that perform certain operation.
- `return`: Indicates what the function returns when we call it.
- `EXPRESSION`: is the expression or variable returned by the return statement.

example

```
# A function that receives two integers
# and returns their sum.
def add(value1, value2):
    sum = value1 + value2
    return sum

a = add(3,7)

a                        --> 10
```

LISTS

Lists are a data structure and a data type with special characteristics. The special thing about lists in Python is that they can store any type of value such as numbers, decimal numbers, strings and even other lists.

Creating a list:

A list must have a name then a = and brackets. Within the brackets, the elements are separated by commas.

```
a_list=[element1, element2,...]
```

Accessing a list:

To access elements in a specific position, we call the name of our list followed by an index within brackets.

```
sample = [1, 2.5, "START", [5,6]]

sample[0]       --> 1
sample[1]       --> 2.5
sample[2]       --> "START"
sample[3]       --> [5,6]
sample[3][0]    --> 5
sample[3][1]    --> 6
```

append()

Use to add new items at the end of an existing list.

```
sample.append(10)
      --> [1,2.5,"START",[5,6],10]
sample.append([2,5])
      --> [1,2.5,"START",[5,6],10,[2,5]]
```

TIP: See https://docs.python.org/3/tutorial/datastructures.html#more-on-lists for other functions that can be called on the type list.

LOOPS

Loops are used to repeat a set of statements for a number of times, for example on different variables.

For loops

The for cycle repeats the instruction block a predetermined number of times. The block of instructions that is repeated is called the body of the loop.

```
for each_variable in a_list:
  # body of the loop
```

- `each_variable`: is a control variable that takes the value of each element in the list `a_list`, from the first element (`a_list[0]`) to the last element (`a_list[len(a_list)-1]`).

example

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

--> apple
    banana
    cherry
```

CONDITIONALS

Conditional statements provide the developer with the ability to check conditions and change the behavior of the program accordingly. This is very powerful and is necessary in many situations.

If statement

Allows a program to execute instructions when a condition is met.

```
if BOOLEAN_EXPRESSION:
    STATEMENTS
```

If else statement

Used if want one thing to happen when a condition is true, and something else to happen when it is false.

```
if B_EXPR:
    STATEMENTS_1  # when B_EXPR is True
else:
    STATEMENTS_2  # when B_EXPR is False
```

Chained conditionals

Used when are more than two possibilities and we need more than two branches.

```
if x < y:
    STATEMENTS_A  # when x < y is True
elif x > y:
    STATEMENTS_B  # when x > y is True
else:
    STATEMENTS_C  # in all other cases
```

NOTE: Conditionals can be nested, so any STATEMENTS can contain a new nested conditional itself!

example

```
# we assume x is an integer
if x > 0:
  if x < 10:
    print("x is positive single digit.")
  else:
    print("x is positive, many digits.")
elif x > -10:
  print("x is negative single digit.")
else:
  print("x is negative, many digits.")
```

CLASSES

Python is an object oriented programming language. Almost everything in Python is an object, with its properties and methods. A Class is like an object constructor, or a "blueprint" for objects.

Class definition

```
class ClassName():
  def __init__(self, attr1, attr2):
    self.attribute1 = attr1
    self.attribute2 = attr2

  def method1 (self, arg):
    # Something I want my object to do
```

- `__init__`: is the method called everytime a new object of class `ClassName` is created.
- `self.attribute1`: It is an instance variable of the object of class `ClassName`. It is associated to its object until the object exists.
- `method1`: is a method, or action, that can be called on any object of class `ClassName`.

example

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

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

p1 = Person("John", 36)
p1.myfunc()          --> My name is John
```