Python

Cheat Sheet

Python Programming Language

Variables

We use variables to temporarily store data in computer's memory.

```
price = 10
rating = 4.9
course_name = 'Python for Beginners'
is_published = True
```

In the above example,

- price is an integer (a whole number without a decimal point)
- rating is a float (a number with a decimal point)
- course_name is a string (a sequence of characters)
- is_published is a boolean. Boolean values can be True or False

string

We can define strings using single (‘ ‘) or double (“ “) quotes.
To define a multi-line string, we surround our string with tripe quotes (“””).
We can get individual characters in a string using square brackets [].

```
course = ‘Python for Beginners’
course[0]  # returns the first character
course[1]  # returns the second character
course[-1]  # returns the first character from the end
course[-2]  # returns the second character from the end
```

We can slice a string using a similar notation:

```
course[1:5]
```

The above expression returns all the characters starting from the index position of 1 to 5 (but excluding 5). The result will be ytho
If we leave out the start index, 0 will be assumed.
If we leave out the end index, the length of the string will be assumed.

integer

You can use the type method to check the value of an object.

```
>>> type(3)
< type 'int' >
```

You can use the basic mathematical operators:

```
>>> 3+3
6
```

when you divide a whole number by a whole number and the answer is a fractional number, Python returns a whole number without the remainder.

```
>>> 9 / 4
2
```

List

```
numbers = [1, 2, 3, 4, 5]

numbers[0]  # returns the first item
numbers[1]  # returns the second item
numbers[-1]  # returns the first item from the end
numbers[-2]  # returns the second item from the end
```

tuple

They are like read-only lists. We use them to store a list of items. But once we define a tuple, we cannot add or remove items or change the existing items.

```
coordinates = (1, 2, 3)
```

We can unpack a list or a tuple into separate variables:

```
x, y, z = coordinates
```

Dictionary

We use dictionaries to store key/value pairs.

```
customer = { “name”: “John Smith”, “age”: 30,
“is_verified”: True }
```

We can use strings or numbers to define keys. They should be unique. We can use any types for the values.

```
customer[“name”]  # returns “John Smith”
customer[“type”]  # throws an error
```

Set

Set items are unordered, unchangeable, and do not allow duplicate values.

```
#creat a Set
x = {"apple", "banana", "cherry"}
print (x)
```

the output:

{'apple', 'cherry', 'banana'}

```
#Duplicate values will be ignored
x = {"apple", "banana", "cherry", "apple"}
print(x)
```

the output:

{'banana', 'cherry', 'apple'}

frozenset

```
# creating a dictionary
Student = {"name": "Ankit", "age": 21, "sex": "Male",
  "college": "MNNIT Allahabad", "address":
"Allahabad"}

# making keys of dictionary as frozenset
key = frozenset(Student)

# printing dict keys as frozenset
print('The frozen set is:', key)
```

the output:

```
The frozen set is: frozenset({'address', 'name', 'age',
'sex', 'college'})
```

range

range(start, stop, step)

```
#Create a sequence of numbers from 3 to 19, increment by 2
x = range(3, 20, 2)
for n in x:
  print(n)
```

the output:

```
3
5
7
9
11
13
15
17
19
```

Boolean

The bool() function allows you to evaluate any value, and give you True or False in return

```
#boolean is true for nonzero value
y = bool(10)
print(y)
```

the output:
true

input

We can receive input from the user by calling the input() function.

```
birth_year = int(input('Birth year: '))
```

The input() function always returns data as a string. So, we're converting the result into an integer by calling the built-in int() function.

output

Using print() function pass to it parameter to print str or variable.

parameters: object, sep, end, file, flush.

```
print('09','12','2016', sep='-')
```

the output:
09-12-2016

```
print('G','F', sep=' ', end=' ')
print('G')
```

the output: GFG

python operators

- **Arithmetic Operators:**

OPERATOR	DESCRIPTION	SYNTAX
+	Addition: adds two operands	x + y
–	Subtraction: subtracts two operands	x – y
*	Multiplication: multiplies two operands	x * y
/	Division (float): divides the first operand by the second	x / y
//	Division (floor): divides the first operand by the second	x // y
%	Modulus: returns the remainder when the first operand is divided by the second	x % y
**	Power: Returns first raised to power second	x ** y

python operators

- **Comparison Operators:**

OPERATOR	DESCRIPTION	SYNTAX
>	Greater than: True if the left operand is greater than the right	x > y
<	Less than: True if the left operand is less than the right	x < y
==	Equal to: True if both operands are equal	x == y
!=	Not equal to – True if operands are not equal	x != y
>=	Greater than or equal to True if the left operand is greater than or equal to the right	x >= y
<=	Less than or equal to True if the left operand is less than or equal to the right	x <= y
is	x is the same as y	x is y
is not	x is not the same as y	x is not y

python operators

- **Logical Operators::**

Operator	Description	Syntax
and	Logical AND: True if both the operands are true	x and y
or	Logical OR: True if either of the operands is true	x or y
not	Logical NOT: True if the operand is false	not x

python operators

- **Bitwise Operators:**

Operator	Description	Syntax
&	Bitwise AND	x & y
|	Bitwise OR	x | y
~	Bitwise NOT	~x
^	Bitwise XOR	x ^ y
>>	Bitwise right shift	x>>
<<	Bitwise left shift	x<<

python operators

- Logical Operators: and , or , not
- Special Operators: is , is not , in , not in
  - is: return true if the operands are identical.
  - is not: return true if the operands are not identical.
  - in: return true if an item belongs to sequence.
  - not in: return false if an item not belongs to sequence.

conditional execution

If statement

Syntax:

```
if condition :
        statement(s) to execute
```

Example:

```
num = 3
if num > 0:
   print(num, "is a positive number.")
```

If-else statement

Syntax:

```
if condition :
        body of if
else :
        body of else
```

Example:

```
num = 3
if num > 0:
   print(num, "is a positive number.")
else :
   print(num, "is a negative number.")
```

If-elif-else statement

Syntax:

```
if condition :
        body of if
elif condition :
        body of elif
else :
        body of else
```

Example:

```
if num > 0:
   print("Positive number")
elif num == 0:
   print("Zero")
else:
   print("Negative number")
```

Nested if statement

Syntax:

```
if condition :
        if condition :
                body of inner if
```

Example:

```
num = float(input("Enter a number: "))
if num >= 0:
   if num == 0:
      print("Zero")
   else:
      print("Positive number")
else:
   print("Negative number")
```

loops

For loop is used to iterate through a sequence of elements in a list, tuples, directories, and strings.

list:

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

string:

```
for x in "banana":
  print(x)
```

range():

```
for x in range(6):
  print(x)
```

loops

While loop is used to iterate through a block of statements as long as the condition is true.

Print i as long as i is less than 6:

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

Python break statement

The break statement will terminate the loop once executed.

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

Python continue statement

The continue statement it will skip the remaining code in a current iteration and start a new iteration.

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

Python Functions

Syntax:

```
def function_name(parameters) :
        statement(s)
```

Example of non-returned type function:

```
def greet(name):
        print("Hello, " + name + ". Good morning!")
```

This function named "**greet**" it will print the user a greeting message.

How to call the function? Simply by the name of the function

```
def greet(name):
        print("Hello, " + name + ". Good morning!")
greet("AI and Robot Club") # call
```

Example of returned type function:

```
def absolute_value(num):
    if num >= 0:
        return num
    else:
        return -num

print(absolute_value(2))
```

Python Modules

A module is a file containing code to perform a specific task.

Three different ways to import modules:

First way

```
import module
module.function()
```

Second way

```
from module import function
function()
```

Third way

```
from module import *
function()
```

Python Modules

we can create or own modules in python.

save the code you want in a file with the file extension .py

file named mymodule.py:

```
def greeting(name):
  print("Hello, " + name)
```

To import module:

```
import mymodule
mymodule.greeting("Jonathan")
```