# Cheatsheet
# Python-3.x

## 1. Storing values in variables

x = 5 stores the integer *5* in *x*
y = 2.5 stores the float *2.5* in *y*
s = “Hello World” stores string *Hello World* in *s*

## 2. Boolean Constants

The Boolean constants are *True* and *False*.
Note the capitalization.

## 3. Arithmetic Operations

x + y computes the sum of *x* and *y*
x - y computes the value of *y* subtracted from *x*
x * y computes the product of *x* and *y*
x ** y computes *x* raised to *y*
x % y computes the remainder when *x* is divided by *y*
x / y computes the float value of *x* divided by *y*.
17 / 4 gives 4.25
x // y computes the quotient when *x* is divided by *y*
17 // 4 gives 4

## 4. Comparison Operations

Returns Boolean values True or False
x == y checks if *x* is equal to *y*
x != y checks if *x* is not equal to *y*
$x > y$ checks if value in *x* is greater than *y*
$x \geq y$ checks if *x* is greater than or equal to *y*
$x < y$ checks if value in *x* is less than that in *y*
$x \leq y$ checks if value in *x* is less than or equal to *y*
$x < y < z$ checks if value in *y* is in between *x* and *z*

## 5. Logical Operations

x == 5 and y != 7 returns True if both conditions are True
x == 5 or y != 7 returns True if either condition is True
not $x > 7$ not negates the condition

## 6. Membership Operators

x in y results in True if x is a member of sequence y.
x not in y results in True if x is not a member of sequence y.

## 7. Identity Operators

x is y Evaluates to True if the variables on either side of the operator point to the same object.
x is not y Evaluates to False if the variables on either side of the operator point to the same object.

## 8. Conversions

int(“65”) gives the integer *65*
int(65.75) gives the integer *65*
float(“65.75”) gives the float *65.75*
float(65) gives the float *65.0*
str(65) gives the string *“65”*
str(65.75) gives the string *“65.75”*
int(“65.75”) gives an error

## 9. Indentation

```
In Python blocks are identified by
indentation.
statement 1:
    statement 2
    statement 3
```

statement 1 must end in a colon. It can be an *if statement*, *while statement*, *for statement* or a *def statement*
Similarly,

```
statement 1
    statement 2
        statement 3
        statement 4
        statement 5
    statement 6
```

Use only 4 spaces for an indent.

## 10. Simple Input

x = input() for taking input.
x = input("Enter number: ") display a prompt while taking input.
The value given by input is always a string.

## 11. Simple Output

print(x) print the value in *x* and a new line.
prin(x, y) print the value in *x* and a space.
print(x,y, sep=“...”) prints the values of *x*, *y* separated by “...” instead of the default space. print(x, y, sep=“”, end = “::”) prints the values of *x*, *y* seperated by a tab and instead of ending with a newline

## 12. if statement

```
if x > 0:
    print(‘‘positive’’)
```

## 13. if...else statement

```
if x > 0:
    print(‘‘positive’’)
else:
    print(’’not positive’’)
```

## 14. if...elif statement

```
if x > 0:
    print(‘‘positive’’)
elif x < 0:
    print(‘‘negative)
else:
    print(‘‘Zero’’)
```

## 15. while statement

```
x = 1
while x < 10:
    print(‘‘The value of x is’’, x)
    x += 1
```

Prints *x* value from 1 to 9

## 16. Defining Strings

s = "I am a string"
enclosed in double quotes.
s = ’He said "Good Morning", to the class’
use single quotes if there is a double quote in the string.
s = "It’s time"
use double quotes if there is a single quote in the string.

## 17. Accessing characters in strings

s[0] accesses the first character in the string *s*.
s[4] accesses the fifth character in the string *s*.
Indexing starts with 0 for the first character.
s[-1] accesses the last character in the string *s*.
s[-2] accesses the last but one character in *s*.
Negative indexing starts with -1 from last.

## 18. Slicing strings

s = "Hello World"
s[3:] returns *"lo World"*
substring from character with index 3 to end.
s[:7] returns *"Hello W"*
substring from start to character with index 6.
s[3:7] returns *"lo W"*
substring from character with index 3 to character with index 6.
s[2:-2] returns "llo Wor"
substring from third character to the third character from the end.

## 19. string methods

s = "Hello" + 'World" stores *HelloWorld* in *s*.
len(s) length of the string *s*
"ell" in s checks for the presence of *"ell"* in *s*.
s.lower() returns *"helloworld"*
a new string with characters of *s*, in lower case.
s.upper() returns *"HELLOWORLD"*
a new string with characters of *s*, in upper case.
s.replace("l", "m") returns *"Hemmo Wormd"*
a new string with all the *l* replaced with *m*.
s.split() returns *["Hello", "World"]*
a list of words in the string.
All the above operations return new strings. The original string remains unaltered.

## 20. range function

range(8) returns list of numbers from *0* to *7*.
range(3, 13, 2) returns odd numbers from *3* to *12*.
range returns a "generator", converts it to list to see the values,
Example: print(list(range(8)))

## 21. Defining functions

```
def add_one(x):
    return x + 1
```

defines the *add_one* function that takes one argument and returns the value of argument plus one.

```
def getMax(x, y):
    if x > y:
        return x
    return y
```

defines the *getMax* function that takes two arguments and returns the greater one from them.

## 22. Calling functions

add_one(5) returns 6.
x = add_one(8) stores the value 9 in x.
x = add_one(x) increments x by one.
y = getMax(4, 8) stores the return value *8* in *y*.

## 23. lists

pr = [2, 3, 5, 7, 11, 13] creates the list *pr*.
len(pr) returns the length of the list, *6*
15 in pr checks for the presence of *15* in the list *pr*.
pr + [17, 19, 23] adds the lists and returns a new list.

## 24. slicing lists

pr[0] accesses the first item, *2*.
pr[-4] accesses the fourth item from end, *5*.
pr[2:] accesses *[5, 7, 11, 13]*
list of items from third to last.
pr[:4] accesses *[2, 3, 5, 7]*
list of items from first to fourth.
pr[2:4] accesses *[5, 7]*
list of items from third to fifth.
pr[1::2] accesses *[3, 7, 13]*
alternate items, starting from the second item.

## 25. list methods

pr.append(17) adds *17* at the end of the list *pr*.
*pr* becomes *[2, 3, 5, 7, 11, 13, 17]*
pr.extend([17, 19, 21]) appends *17*, *19*, *21*
*pr* becomes *[2, 3, 5, 7, 11, 13, 17, 19, 21]*
Operations mentioned above modify the list itself.

## 26. for loop

```
for i in pr:
    print(i)
```

iterates over the list *pr* one item at a time.

## 27. dictionaries

mm2num = {"jan": 1, "feb": 2, "mar": 4}
creates the dictionary *mm2num*
mm2num["feb"] gives the corresponding value, *2*
mm2num["mar"] = 3
changes the value for the key *'mar"* to *3*
mm2num["apr"] = 4
creates the key *"apr"* with *4* as the value
mm2num.values() returns list of values, *[1, 2, 3, 4]*
mm2num.keys() returns list of keys,
*["jan", "feb", "mar", "apr"]*

## 28. sets

prs = set([2, 3, 2, 5, 3, 7, 7, 2, 3])
creates the set *set([2, 3, 5, 7])* and stores in *prs*.
ods = set([1, 3, 5, 9, 3, 7, 7, 9, 3])
creates the set *set([1, 3, 5, 7, 9])* and stores in *ods*.
prs | ods gives the union of the sets, *set([1, 2, 3, 5, 7, 9])*
prs & ods gives the intersection of the sets, *set([3, 5, 7])*
ods - prs gives the difference of sets
items in *ods* that are not in *prs*, which is *set([1, 9])*
ods ^ prs gives the symmetric difference
items in *ods* or in *prs* but not in both, *set([1, 2, 9])*

## 29. Reading from files

```
fileLoc = ''/home/tsprint/primes.txt''
for line in open(fileLoc):
    prime = int(line)
    print(prime * prime)
```

Data in the file is read as a **string** line by line.