Python-3.4 Quick Reference
TalentSprint

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. Arithmetic Operations

3 + 5 computes the sum of *3* and *5*
x + y computes the sum of values in *x* and *y*
73 - 27 computes *27* substracted from *73*
x - y computes the value of *y* substracted from *x*
x * 11 computes the product of *x* and *11*
x ** 5 computes the value of *x* raised to *5*
x // y computes the value of *x* divided by *y*
Note: Integer divison truncates.
Ex: 17 // 4 gives 4 and not 4.25, 17 / 4 gives 4.25

3. Storing result of an operation

s = 9 + 5 stores the sum *14* in *s*
s = x + y stores the sum of *x* and *y* in *s*
p = x * y stores the product of *x* and *y* in *p*
r = 11 % 4 stores the remainder *3* in *r*
x = x + 2 increments *x* by *2*
x += 2 also does the same
x = 17 // 3 stores the result *5* in *x*
x = 7 + 3 * 2 stores the result *13* in *x*

4. Comparison Operations

x == 5 checks if *x* is equal to *5*
it gives True if *x* is equal to *5* and False if not.
x == y checks if the values in *x* and *y* are equal
x != 5 checks if *x* is not equal to *5*
x > 5 checks if value in *x* is greater than *5*
x > y checks if value in *x* is greater than that in *y*
x ≥ 5 checks if *x* is greater than or equal to *5*
x < y checks if value in *x* is less than that in *y*
x ≤ 5 checks if value in *x* is less than or equal to *5*

5. Logical Operations

x == 5 and y != 7 checks for both the conditions
it gives True if *x* is equal to *5* **and** *y* is not equal to *7*
x == 5 or y != 7 checks for atleast one condition
it gives True if *x* is equal to *5* **or** *y* is not equal to *7*
not x > 7 checks if *x* is **not** greater than *7*

Note: The operations in Section 4 and Section 5 are also applicable to strings.

6. boolean datatypes

The *True* and *False* returned by comparison operations and logical operations are of *boolean* datatype.
Only variables of *boolean* datatype must be used in conditional statements and loops.

7. 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"*
Note: int("65.75") gives an error

8. Simple Input

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

9. 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="\t", end = "::") prints the values of *x*, *y* seperated by a tab and instrad of ending with a newline, print ::

10. Indentation

```
statement 1
    statement 2
    statement 3
```

statements 2 and 3 are a block.
statement 1 must end in a colon. it can be an *if statement* or a *while statement* or a *for statement* or a *def statement*
Similarly,

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

statements 2, and 5 are a block.
statements 3 and 4 are a block inside statement 2.
statements 1 and 2 must end in colon
Note: Use only 4 spaces for an indent.

11. if statement

```
if x > 0:
    print(''positive'')
```

Output *positive* if *x* is positive.

12. if...else statement

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

Output *positive* if *x* is positive and *not positive* otherwise.

13. if...elif statement

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

Output *positive* or *negative* or *Zero* based on *x*.

14. while statement

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

Keep printing *x* value and incrementing it until the condition *x < 10* fails.

15. break statement in while

```
while n > 0:
    d = n % 10
    if d % 2 == 0:
        print(''Even digit found'')
        break
    d /= 10
```

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*.
Note: 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*.
Note: 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.
Note: All the above operations return new strings. The original string remains unaltered.

20. 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.

21. 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*.
biggest = getMax(biggest, currentValue)

22. 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.

23. 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.

24. 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]*
Note: Operations mentioned above, modify the list itself.

25. range function

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

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)
```

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