Python Cheatsheet
virtuousprogrammer.com

Types:

bool: True
bytes: b'Hello World"
complex: 7+1.4j
dict: {'test':7, 1.4:False}
int: 7
float: 1.4
list: [7, True, 'hi']
tuple: (7, True, 'hi')
str: "Hello World"

Keywords:

and: Connect two booleans.
  ex. True and False
assert: Assert a value is True, if it isn't throws an exception.
  ex. assert a == b, "a is not b"
break: Breaks out of a loop.
class: Class declaration.
  ex. class NewPicture(Picture):
continue: Returns execution to the top of a loop.
def: Define a function.
  ex. def functionDefinition(argument1, argument2):
del: Remove an object from memory
  ex. del foo
elif: Python equivalent to 'else if'
  ex. elif(a != b):
else: Final option in a string of if/elif statements.
except: Catches a throw exception.
  ex. except NameError:
exec: Execute a string containing python code.
  ex. exec """print 'Hello World'"""
finally: Always executed following the try statement it's part of.
for x in xs: Loop over an iterable object.
  ex. for num in [1, 2, 3]
from: Indicates the module to import from.
  ex. from time import sleep
global: Indicatest that the given variables are global in the code block.
  ex. global foo, bar
if: Conditional statement.
  ex. if(x == 3):
import: Import a module.
  ex. import time
in: Returns True if x is in xs.
  ex. x in xs
is: Tests to see if two objects are the same.
  ex. obj1 is obj2
lambda: Creates adhoc functions.
  ex. lambda x: x * x
not: boolean inversion
  ex. not True
or: Connect two booleans.
  ex. True or False
pass: Noop, for creating an empty block
print: Prints the given value to screen
  ex. print "Hello World"
raise: Raise an exception.
  ex. raise Exception("An error")
return: Return from the given function.
  ex. return 7
try: First half of a try / except statement
while: Loop keyword.
  ex. while a == 3:
with: Creates a context to perform functions on.
  ex. with open("file.txt") as f:
yield: In a generator, acts like return, does not stop execution.
  yield foo

Comparitors:

<, >, ==, >=, <=, <>, !=, is, in, is not, not in

Conditionals:

```
if a == b:
  pass
elif a == c:
  pass
else:
  pass
```

Exception Handling:

```
try:
  open("file.txt")
except IOError:
  pass
else:
  pass
finally:
  pass
```

List Comprehension:

```
[x*2 for x in [1,2,3] if x%2 == 1]
```

Context Managers(with syntax):

```
with open("file.txt") as f:
  print f.read()
```

Function Definition:

```
def funName(arg, arg2=3, *arglist):
  return arg * optionalArg
```

Lambda Functions:

```
lambda arg1, arg2: arg1 + arg2
```

Generator Definition:

```
def genName():
  for x in [1,2,3]
    yield x
```

Class Creation:

```
class myClass(parentClass):
  def __init__(self, arg1):
    self.objectArg = arg1
  def method2(self):
    return self.objectArg
```