

Appendix A: Introduction to Python

Dynamic Typing

```
x = 123
```

```
x = 'hello'
```

```
int x, y;
```

Multi-line Statements

```
x = (1 + 2      # parenthesized expression still open at newline
    3)         # so stmt continues to this line
```

Code Blocks

```
if x == 5:      # test if x is equal to 5
    print(1)    # first stmt in block
    print(2)
    print(3)    # last stmt in block
print(4)
```

```
print('hello')
```

Arithmetic Operations

```
print(5 + 2)    # addition: displays 7
print(5 - 2)    # subtraction: displays 3
print(5 * 2)    # multiplication: displays 10
print(5 ** 2)   # exponentiation: displays 25
print(5 / 2)    # floating-point division: displays 2.5
print(5 // 2)   # integer division: displays 2
print(5 % 2)    # remainder: displays 1
```

Strings

```
print('hello')      # displays hello
print("hello")      # displays hello
print("it's")       # displays it's
print('it\'s')      # displays it's
print('A\\B')       # displays A\B
```

```
if s.startswith('.t'):
    print('it does')
```

```
s = 'x'*5
```

```
i = 30
```

```
t = ' '*i           # t is assigned string consisting of 30 spaces
```

```
print(t + '^')      # position of caret determined by value in i at run time
```

Concatenation

```
print(3 + 2)      # addition: displays 5  
print('3' + '2') # concatenation: displays 32
```

Assignment Operator

`x = 1`

`x = x + 1`

`x += 1`

Functions

```
1 def sample():           # this is the start of the function definition
2     print('morning')
3     print('bye')         # end of function definition
4 print('good')
5 sample()                 # this stmt "calls" sample()
```

```
good
morning
bye
```



```
1 def f():
2     print('hello')
3     g()          # forward reference ok here
4
5 def g():
6     print('bye')
7 f()             # this call must follow the definition of f()
```

```
1 def g(x, y):    # x and y are parameters
2     print(x + y) # displays 75
3 z = 20
4 def f():
5     g(z + 5, 50) # z + 5 and 50 are arguments
6 f()
```

```
1 def addorconcat(x, y):  
2     result = x + y  
3     return result
```

```
4 y = addorconcat(3, 2)      # y is assigned 5  
5 y = addorconcat('3', '2') # y is assigned '32'
```

y = 5

y = '32'

```
1 def r():
2     print('hello')
3     r()                # r() calls itself
4 r()
```

```
1 def countdown(n)
2     if n > 0:           # recurse only if n is positive
3         print(n)
4         countdown(n - 1) # next n is one less the current n
5 countdown(10)
```

Global and Local Variables

```
1 x = 1          # x is global here
2 y = 2          # y is global here
3 z = 3          # z is global here
4 def f():
5     global x    # makes x global in this function
6     x = y       # x and y are global variables here
7     z = 20      # z is a local variable here
8 f()
9 print(x)        # displays 2
10 print(y)       # displays 2
11 print(z)       # displays 3
```

```
1 def f():  
2     x = 1      # this is the local variable x in f()  
3     g()  
4     print(x)  # displays 1  
5 def g():  
6     x = 7      # does not change local variable x in f()  
7 f()
```

```
tempcount = 0
```

```
def gettemp():  
    global tempcount  
    ...  
    tempcount += 1 # increment tempcount for next call of cg_gettemp()
```

print Statement

```
print()          # nothing displayed, goes to next line
print(1)         # displays 1, goes to next line
print(1, 2)      # displays 1 2 then goes to the next line
print(1, 2,)     # displays 1 2 then goes to the next line

print(1, 2, end = '') # does not move cursor to next line
print(3)
```

```
print('x = %5d y = %d' % (x, y))
```

```
x =      2 y = 3
```

```
print('%50s' % 'hello')
```

```
hello
```

input() Function

```
x = input('enter integer')
```

```
enter integer
```


if Statement

true/false expression

```
if x == 5:  
    print('hello')
```

true/false expression

```
if x == 5:  
    print('hello')  
else:  
    print('bye')
```

```
x = 1  
if x:  
    print('hello')
```

1 is treated as True
hello is displayed

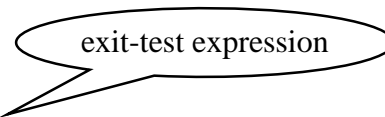
Relational operator	Meaning
==	equal
!=	not equal
<	less than
<=	less than or equal
>	greater than
>=	greater than or equal

```
if grade >= 90:  
    print('A')  
elif grade >= 80:    # elif means else if  
    print('B')  
elif grade >= 70:  
    print('C')  
elif grade >= 65:  
    print('D')  
else:  
    print('F')
```

```
if x == 1 or y == 2:  
    print('got 1 or 2 or both')
```

```
if not(x == 1 or y == 2):  
    print('x not 1 and y not 2')
```

while Statement



```
1 i = 1
2 while i <= 10:    # loop body executed while i <= 10 is True
3     print(i)
4     i += 1
```

```
1 i = 1
2 while i <= 10:
3     print(i)
4     if i == 5:
5         break    # causes a break out of the loop
6     i += 1
```

```
while True:
    :                # stmts before the exit test
    if exit-test_expression:    # this is the exit test
        break
    :                # stmts after the exit test
```

```
1 sum = 0
2 while True:
3     x = int(input('enter integer: ')) # prompt user
4     if x < 0:                        # check for a neg number
5         break                        # exit loop
6     sum += x                         # add x to sum
7 print('sum = ' + str(sum))          # display final sum
```

```
enter integer: 5
enter integer: 3
enter integer: -1
sum = 8
```

Files

```
infile = open('t.in', 'r')
```

```
source = infile.read()
```

```
outfile = open('c1.s', 'w')
```

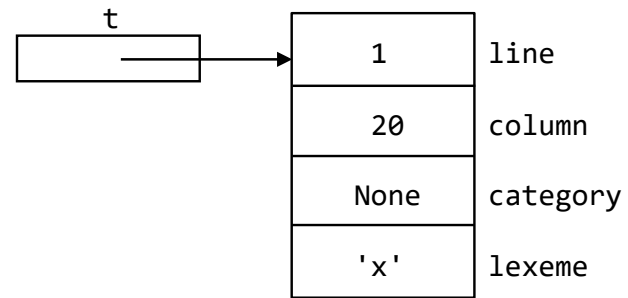
```
outfile.write('hello\n')
```

```
outfile.close()
```

Classes

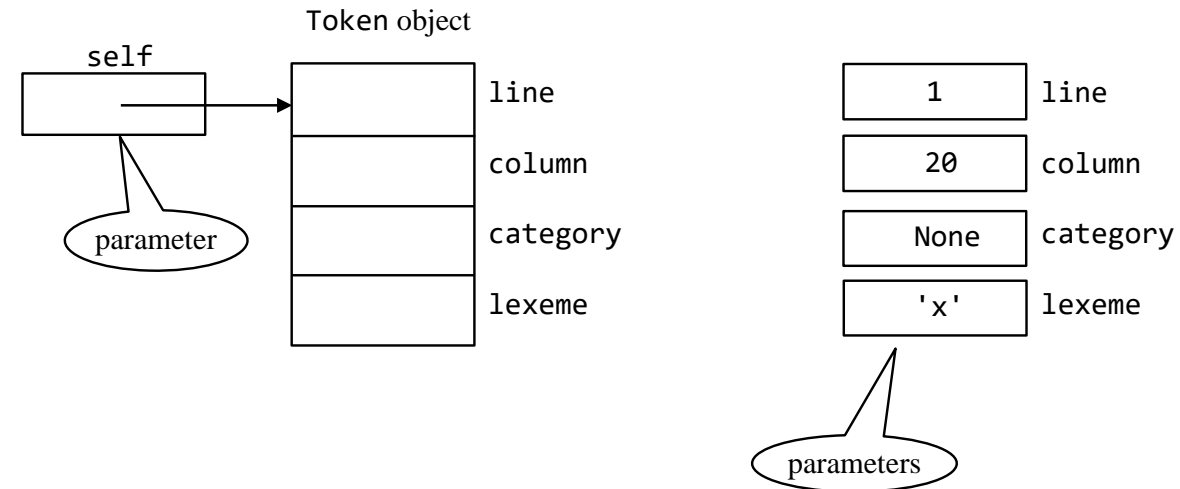
```
1 class Token:
2     def __init__(self, line, column, category, lexeme):
3         self.line = line          # line number of the token
4         self.column = column      # column in which token starts
5         self.category = category  # category of the token
6         self.lexeme = lexeme      # token in string form
```

```
t = Token(1, 20, None, 'x')
```



```
t.category = NAME
```

```
t = Token(1, 20, None, 'x')
```



Exceptions

```
1 try
2     parser()
3 except RuntimeError as emsg      # catches RuntimeError exceptions
4     print(emsg)                  # displays error message
5     sys.exit(1)                  # terminates program
```

```
    raise RuntimeError('Illegal operation')
```

```
    sys.exit(1)
```

```
1 try:
2     infile = open('i1.in', 'r') # opens file for reading
3     source = infile.read()      # reads the entire file
4 except IOError:                 # catches IOError exceptions
5     print('Cannot read input file i1.in')
6     sys.exit(1)                 # terminates the program
```

```
class Returnsignal(Exception)
    pass
```

```
    raise Returnsignal()
```

Lists

```
zlist = []
```

```
zlist.append('hello')
```

```
zlist.append(234)
```

```
print(zlist)
```

```
['hello', 234]
```

```
print(zlist[0]) # displays element at index 0
```

```
print(zlist[1]) # displays element at index 1
```

```
x = zlist.pop()
```

```
if 234 in zlist:
```

```
    print('it is there')
```

```
index = zlist.index('hello')
```

```
for i in zlist:
```

```
    print(i)
```

splitlines() Method

```
infile = open('i1.in', 'r')
```

```
source = infile.read()
```

```
lines = source.splitlines()
```

Dictionaries

```
d = {}
```

```
d['x'] = 'up'
```

```
d['y'] = 77
```

```
print(d['y'])      # displays 77
```

```
if 'y' in d:
```

```
    print('it is there')
```


Accessing Command Line Arguments

```
python cla.py hello 123
```

```
import sys
result = int(sys.argv[1]) + int(sys.argv[2])
print(result)
```

```
import sys
```

len() Function

```
print(len(zlist))
```

```
print(len(sys.argv))
```

isalpha(), isdigit(), isalnum(), and isspace() Methods

```
1 a = 'hello'
2 print(a.isalpha())    # displays True
3 print(a.isdigit())    # displays False
4 print(a.isalnum())    # displays True
5 print(a.isspace())    # displays False
```

Time and Date

```
print(time.strftime('%c'))
```

To use `time.strftime('%c')`, insert the following statement at the beginning of your program:

```
import time
```