

Control of Flow

- There are several Python expressions that control the flow of a program. All of them make use of Boolean conditional tests.
 - If Statements
 - While Loops
 - Assert Statements

If Statements

```
if x == 3:  
    print "X equals 3."  
elif x == 2:  
    print "X equals 2."  
else:  
    print "X equals something else."  
print "This is outside the 'if'."
```

If Statements - Another Ex.

```
x = 'killer rabbit'
if x == 'roger':
    print "how's jessica?"
elif x == 'bugs':
    print "What's up, doc?"
else:
    print "Run away! Run away!"
```

Run away! Run away!

While Loops

```
x = 3
```

```
while x < 10:
```

```
    x = x + 1
```

```
    print "Still in the loop."
```

```
print "Outside of the loop."
```

While Loops, more examples

```
While 1:
```

```
    print "Type Ctrl-C to stop me!"
```

```
x = 'spam'
```

```
While x:
```

```
    print x
```

```
    x = x[1:]
```

```
'spam'  'pam'  'am'  'm'
```

Break and Continue

- You can use the keyword **break** inside a loop to leave the while loop entirely.
- You can use the keyword **continue** inside a loop to stop processing the current iteration of the loop and to immediately go on to the next one.

While Loop with Break

```
while 1:  
    name = raw_input('Enter name:')  
    if name == 'stop': break  
    age = raw_input('Enter age:')  
    print 'Hello', name, '=>', \  
        int(age)**2
```

While Loop with Break, cont.

```
Enter name: mel  
Enter age: 20  
Hello mel => 400
```

```
Enter name: bob  
Enter age: 30  
Hello, bob => 900
```

```
Enter name: stop
```

While Loop with Continue

```
x = 10
```

```
while x:
```

```
    x = x-1
```

```
    if x % 2 != 0: continue    #Odd? Skip print
```

```
    print x
```

Assert

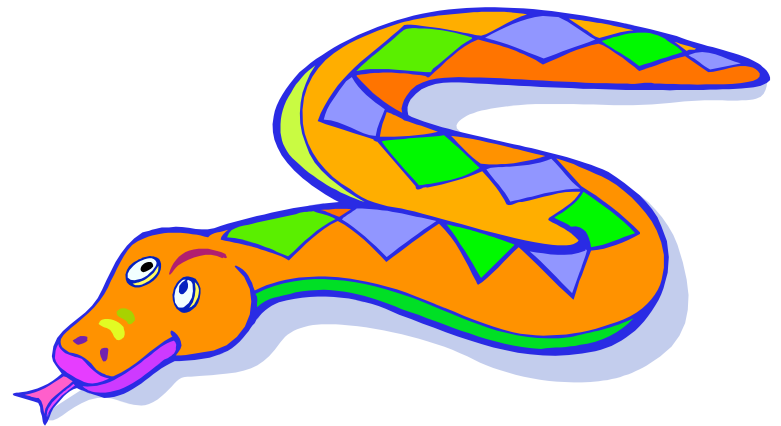
- An assert statement will check to make sure that something is true during the course of a program.
 - If the condition is false, the program stops.

```
assert (number_of_players < 5)
```

```
if number_of_players >= 5:
```

```
    STOP
```

Logical Expressions



True and False

- True and False are constants in Python.
 - Generally, True equals 1 and False equals 0.
- Other values equivalent to True and False:
 - False: zero, None, empty container or object
 - True: non-zero numbers, non-empty objects.
- Comparison operators: `==`, `!=`, `<`, `<=`, etc.
 - X and Y have same value: `X == Y`
 - X and Y are two names that point to the same memory reference: `X is Y`

Boolean Logic Expressions

- You can also combine Boolean expressions.
 - True if a is true and b is true: `a and b`
 - True if a is true or b is true: `a or b`
 - True if a is false: `not a`
- Use parentheses as needed to disambiguate complex Boolean expressions.

Special Properties of And and Or

- Actually 'and' and 'or' don't return True or False. They return the value of one of their sub-expressions (which may be a non-Boolean value).
- **X and Y and Z**
 - If all are true, returns value of Z.
 - Otherwise, returns value of first false sub-expression.
- **X or Y or Z**
 - If all are false, returns value of Z.
 - Otherwise, returns value of first true sub-expression.

The "and-or" Trick

- There is a common trick used by Python programmers to implement a simple conditional using 'and' and 'or.'
 - `result = test and expr1 or expr2`
 - When test is True, result is assigned expr1.
 - When test is False, result is assigned expr2.
 - Works like (test ? expr1 : expr2) expression of C++.
- But you must be certain that the value of expr1 is never False or else the trick won't work.

- I wouldn't use this trick yourself, but you should be able to understand it if you see it

Regular expressions in Python

```
>>> import re
>>> p = re.compile("` +")
>>> line = "this is a test  a  test a  test"
>>> line.split(' ')
>>> ?
>>> p.split(line)
>>> ?
>>> line.replace(' ', '=')
>>> ?
>>> p.sub('= ', line)
>>> ?
>>> p.subn('= ', line)
```

More regular expression functions

- `p.match(string)`
 - Matches `p` from the beginning of `string`
- `p.search(string)`
 - Matches `p` anywhere in `string`, stops at first match
- `p.findall(string)`
 - Return all matched strings in a list

An example

```
>>> line = "These are test1 test2 test3"
>>> p2 = re.compile(r"test\d")
>>> if p2.search(line):
        print p2.search(line).group()
>>> ?
>>> p2.findall(line)
>>> ?
```

Escape! Escape!

```
>>> string1 = "\\section"
>>> p3 = re.compile("\\\\section")
>>> p3.search(string1).group()
>>> ?

>>> p3 = re.compile("\\\\\\section")
>>> p3.search(string1).group()
>>> ?

>>> p3.search(string1).group()
>>> p3 = re.compile(r'\\section')
>>> ?
```