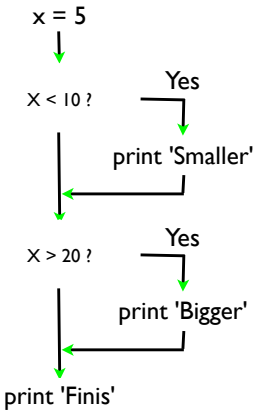
Conditional Execution

Chapter 3

Python for Informatics: Exploring Information
www.py4inf.com

Conditional Steps



Program:

```
x = 5
if x < 10:
    print 'Smaller'

if x > 20:
    print 'Bigger'

print 'Finis'
```

Output:

```
Smaller
Finis
```

Comparison Operators

- Boolean expressions ask a question and produce a Yes or No result which we use to control program flow
- Boolean expressions using comparison operators evaluate to - True / False - Yes / No
- Comparison operators look at variables but do not change the variables

Python	Meaning
<	Less than
<=	Less than or Equal
==	Equal to
>=	Greater than or Equal
>	Greater than
!=	Not equal

Remember: "=" is used for assignment.

http://en.wikipedia.org/wiki/George_Boole

Comparison Operators

```
x = 5
if x == 5 :
    print 'Equals 5'

if x > 4 :
    print 'Greater than 4'
if  x >= 5 :
    print 'Greater than or Equal 5'

if x < 6 : print 'Less than 6'

if x <= 5 :
    print 'Less than or Equal 5'
if x != 6 :
    print 'Not equal 6'
```



```
Equals 5
Greater than 4
Greater than or Equal 5
Less than 6
Less than or Equal 5
Not equal 6
```

Boolean Operations

- We can do calculations with boolean variables just like with integer variables
- The boolean operations are: and or not
- Comparison operators < > <= >= == != return boolean (True or False)

Boolean Operators

P	Q	P and Q
T	T	T
F	T	F
T	F	F
F	F	F

(x == 4) and (y ==2)

True if both expressions are true.

(x == 4) or (y == 2)

Evaluates to true if either expression is true.

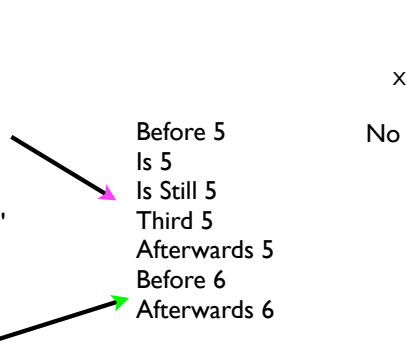
P	Q	P or Q
T	T	T
F	T	T
T	F	T
F	F	F

P	not P
T	F
F	T

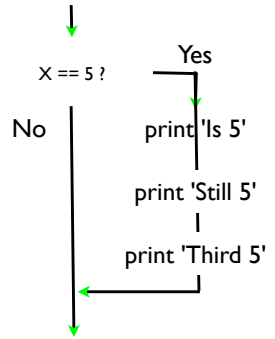
not (x == 4)

Not "flips" the logic - True becomes False and False becomes True.

```
x = 5
print 'Before 5'
if  x == 5 :
    print 'Is 5'
    print 'Is Still 5'
    print 'Third 5'

print 'Afterwards 5'

print 'Before 6'
if x == 6 :
    print 'Is 6'
    print 'Is Still 6'
    print 'Third 6'

print 'Afterwards 6'
```



```
Before 5
Is 5
Is Still 5
Third 5
Afterwards 5
Before 6
Afterwards 6
```



One-Way Decisions

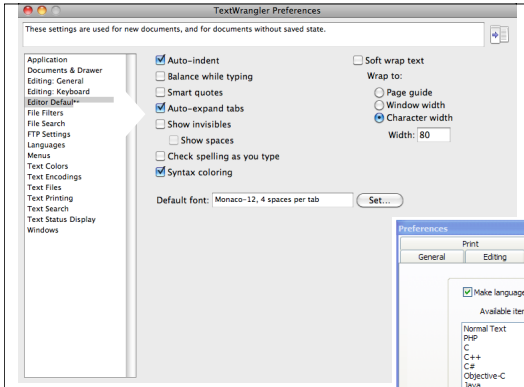
Indentation

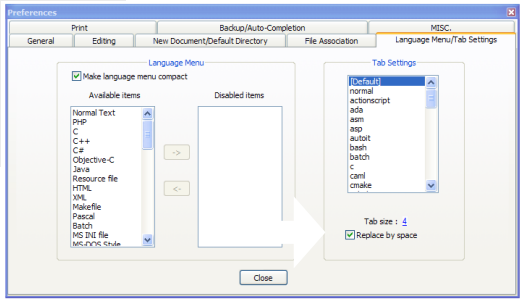
- Increase indent indent after an if statement or for statement (after :)
- Maintain indent to indicate the scope of the block (which lines are affected by the if/for)
- Reduce indent to *back to* the level of the if statement or for statement to indicate the end of the block
- Blank lines are ignored - they do not affect indentation
- Comments on a line by themselves are ignored w.r.t. indentation

Warning: Turn Off Tabs

- Most text editors can turn tabs into spaces - make sure to enable this feature
  - NotePad++: Settings -> Preferences -> Language Menu/Tab Settings
  - TextWrangler: TextWrangler -> Preferences -> Editor Defaults
- Python cares a *lot* about how far line is indented. If you mix tabs and spaces, you may get "indentation errors" even if everything looks fine

Please do this now while you are thinking about it so we can all stay sane...





This will save you much unnecessary pain.

increase / maintain after if or for
decrease to indicate end of block
blank lines and comment lines ignored

```
x = 5
if x > 2 :
    print 'Bigger than 2'
    print 'Still bigger'
print 'Done with 2'

for i in range(5) :
    print i
    if i > 2 :
        print 'Bigger than 2'
    print 'Done with i', i
```

```
x = 5
if x > 2 :
# comments

    print 'Bigger than 2'
        # don't matter
    print 'Still bigger'
# but can confuse you

print 'Done with 2'
    # if you don't line
        # them up
```

Mental begin/end squares

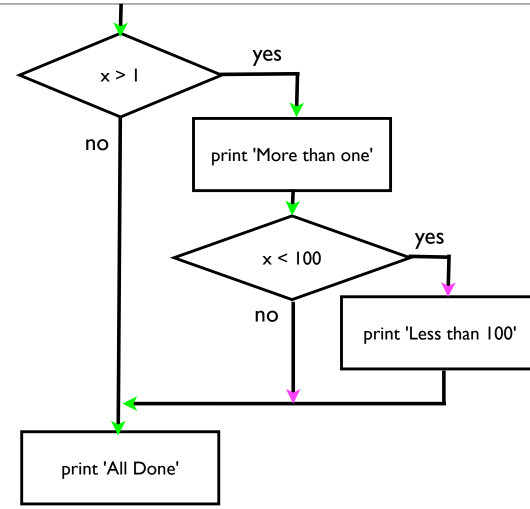
```
x = 5
if x > 2 :
    print 'Bigger than 2'
    print 'Still bigger'
print 'Done with 2'
```

```
for i in range(5) :
    print i
    if i > 2 :
        print 'Bigger than 2'
    print 'Done with i', i
```

```
x = 5
if x > 2 :
# comments

    print 'Bigger than 2'
        # don't matter
    print 'Still bigger'
# but can confuse you

print 'Done with 2'
    # if you don't line
        # them up
```

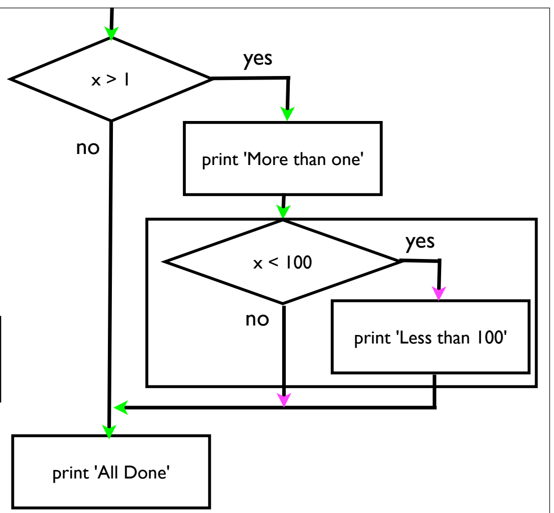
Nested Decisions

```
x = 42

if x > 1 :
    print 'More than one'
    if x < 100 :
        print 'Less than 100'

print 'All done'
```



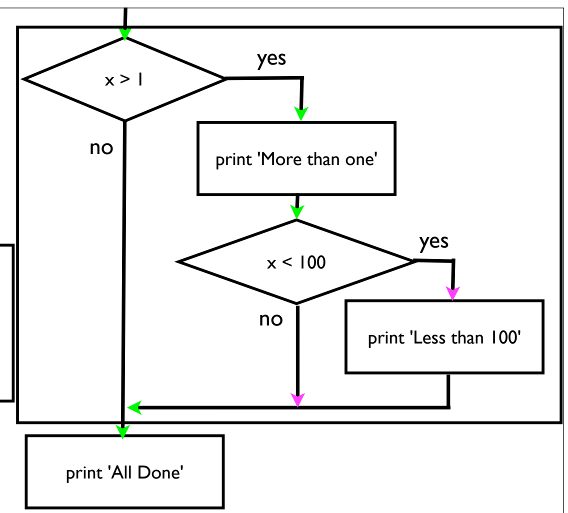
Nested Decisions

```
x = 42

if x > 1 :
    print 'More than one'
    if x < 100 :
        print 'Less than 100'

print 'All done'
```



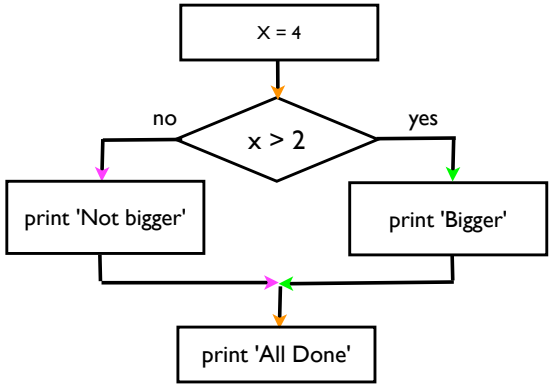
Nested Decisions

```
x = 42

if x > 1 :
    print 'More than one'
    if x < 100 :
        print 'Less than 100'

print 'All done'
```



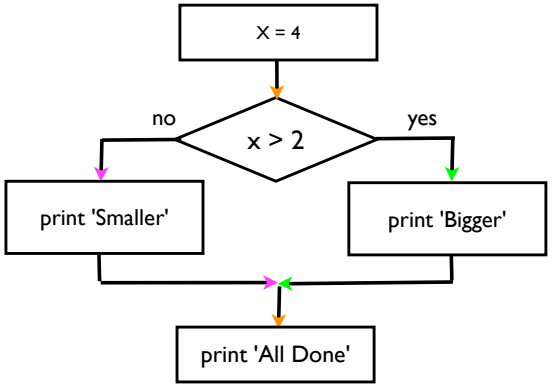
Two Way Decisions

- Sometimes we want to do one thing if a logical expression is true and something else if the expression is false
- It is like a fork in the road - we must choose one or the other path but not both



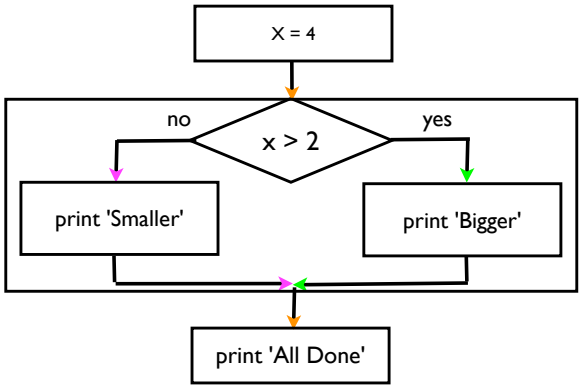
Two-way using else :

```
x = 4

if x > 2 :
    print 'Bigger'
else :
    print 'Smaller'

print 'All done'
```



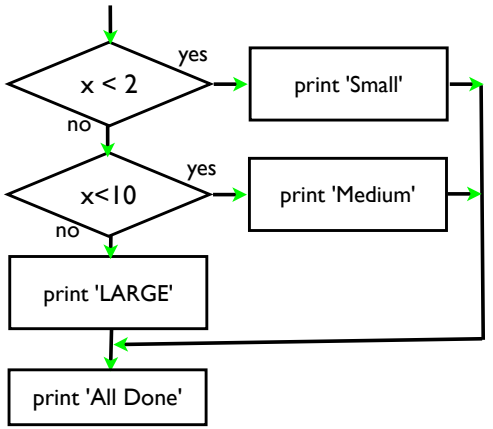
Two-way using else :

```
x = 4

if x > 2 :
    print 'Bigger'
else :
    print 'Smaller'

print 'All done'
```



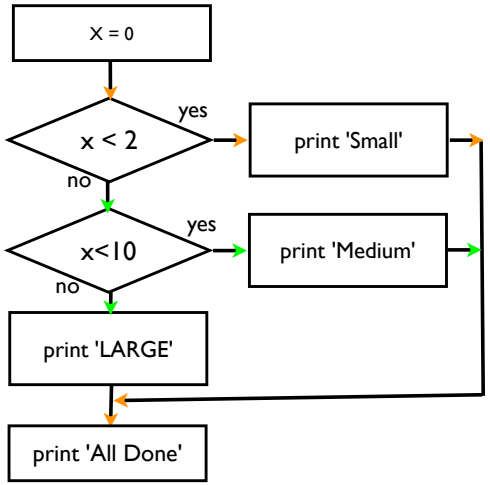
Multi-way

```
if x < 2 :
    print 'Small'
elif x < 10 :
    print 'Medium'
else :
    print 'LARGE'
print 'All done'
```



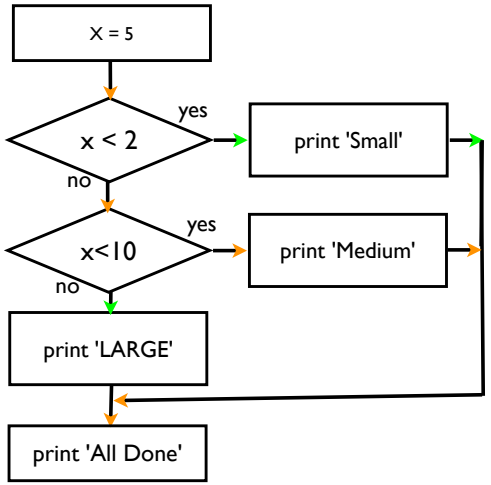
Multi-way

```
x = 0
if x < 2 :
    print 'Small'
elif x < 10 :
    print 'Medium'
else :
    print 'LARGE'
print 'All done'
```



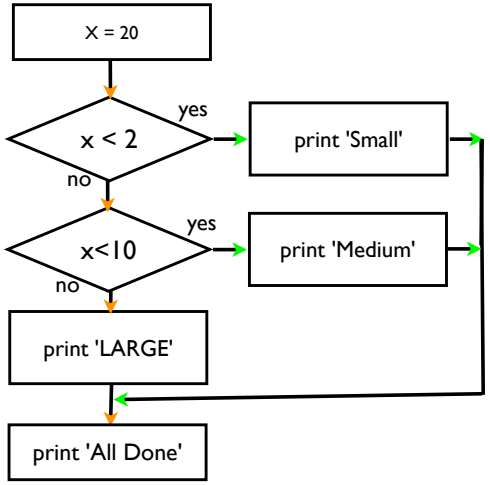
Multi-way

```
x = 5
if x < 2 :
    print 'Small'
elif x < 10 :
    print 'Medium'
else :
    print 'LARGE'
print 'All done'
```



Multi-way

```
x = 20
if x < 2 :
    print 'Small'
elif x < 10 :
    print 'Medium'
else :
    print 'LARGE'
print 'All done'
```



Multi-way

```
# No Else
x = 5
if x < 2 :
    print 'Small'
elif x < 10 :
    print 'Medium'

print 'All done'
```

```
if x < 2 :
    print 'Small'
elif x < 10 :
    print 'Medium'
elif x < 20 :
    print 'Big'
elif x< 40 :
    print 'Large'
elif x < 100:
    print 'Huge'
else :
    print 'Ginormous'
```

Multi-way Puzzles

Which will never print?

```
if x < 2 :
    print 'Below 2'
elif x >= 2 :
    print 'Two or more'
else :
    print 'Something else'
```

```
if x < 2 :
    print 'Below 2'
elif x < 20 :
    print 'Below 20'
elif x < 10 :
    print 'Below 10'
else :
    print 'Something else'
```

The try / except Structure

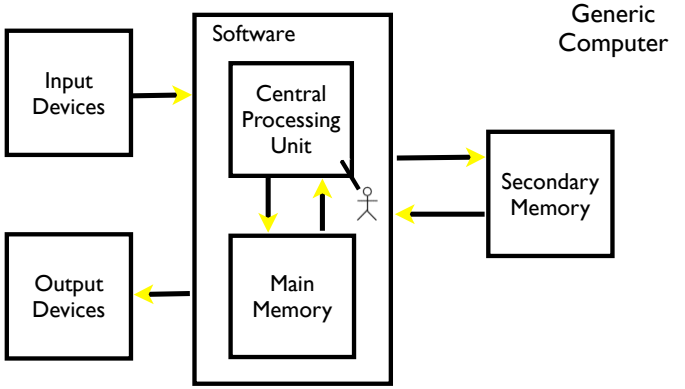
- You surround a dangerous section of code with try and except.
- If the code in the try works - the except is skipped
- If the code in the try fails - it jumps to the except section

```
$ cat notry.py
astr = 'Hello Bob'
istr = int(astr)
```

The program stops here

```
$ python notry.py
Traceback (most recent call last):
  File "notry.py", line 6, in <module>
    istr = int(astr)
ValueError: invalid literal for int() with
base 10: 'Hello Bob'
```

All Done



```
$ cat tryexcept.py
astr = 'Hello Bob'
try:
    istr = int(astr)
except:
    istr = -1

print 'First', istr

astr = '123'
try:
    istr = int(astr)
except:
    istr = -1

print 'Second', istr
```

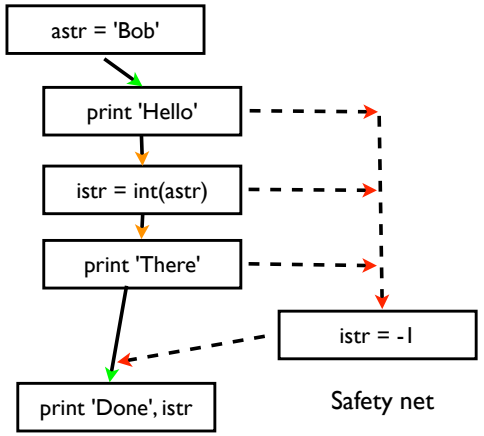
When the first conversion fails - it just drops into the except: clause and the program continues.

```
$ python tryexcept.py
First -1
Second 123
```

When the second conversion succeeds - it just skips the except: clause and the program continues.

try / except

```
astr = 'Bob'
try:
    print 'Hello'
    istr = int(astr)
    print 'There'
except:
    istr = -1

print 'Done', istr
```



Sample try / except

```
rawstr = raw_input('Enter a number:')

try:
    ival = int(rawstr)
except:
    ival = -1

if ival > 0 :
    print 'Nice work'
else:
    print 'Not a number'
```

```
$ python trynum.py
Enter a number:42
Nice work
$ python trynum.py
Enter a number:fourtytwo
Not a number
$
```

Exercise

Rewrite your pay computation to give the employee 1.5 times the hourly rate for hours worked above 40 hours.

```
Enter Hours: 45
Enter Rate: 10
Pay: 475.0
```

475 = 40 * 10 + 5 * 15

Exercise

Rewrite your pay program using try and except so that your program handles non-numeric input gracefully.

```
Enter Hours: 20
Enter Rate: nine
Error, please enter numeric input

Enter Hours: forty
Error, please enter numeric input
```

Summary

- Comparison operators == <= >= > < !=
- Logical operators: and or not
- Indentation
- One Way Decisions
- Two way Decisions if : and else :
- Nested Decisions
- Multiway decisions using elif
- Try / Except to compensate for errors
- Short circuit evaluations