

Module 1:

Why Program?

- Computers want to be helpful...
- Computers are built for one purpose - to do things for us
- But we need to speak their language to describe what we want it to do
- Users have it easy - someone already put many different programs (instructions) into the computer and users just pick the ones we want to use

Users vs. Programmers

Users see computers as a set of tools - word processor, spreadsheet, map, todo list, etc.

Programmers

- learn the computer “ways” and the computer language
- They have some tools that allow them to build new tools
- They sometimes write tools for (other) users and sometimes they write little “helpers” for themselves to automate a task

From a software creator’s point of view, we build the software. The end users (stakeholders/actors) are our masters - who we want to please - often they pay us money when they are pleased. But the data, information, and networks are our problem to solve on their behalf. The hardware and software are our friends and allies in this quest.

Why be a programmer?

To get some task done –

- we are the user and programmer

To produce something for others to use –

- a programming job

What is Code? Software? A Program?

A sequence of stored instructions

- It is a small piece of our intelligence in the computer
- It is a small piece of our intelligence we can give to others - we figure something out and then we encode it and then give it to someone else, to save them the time and energy of figuring it out
- A piece of creative art - particularly when we do a good job on user experience

Programs for Python...

the clown ran after the car and the car ran into the tent and the tent fell down on the clown and the car

A python program to find and display the word and count of the word used maximum number of times in the given file.

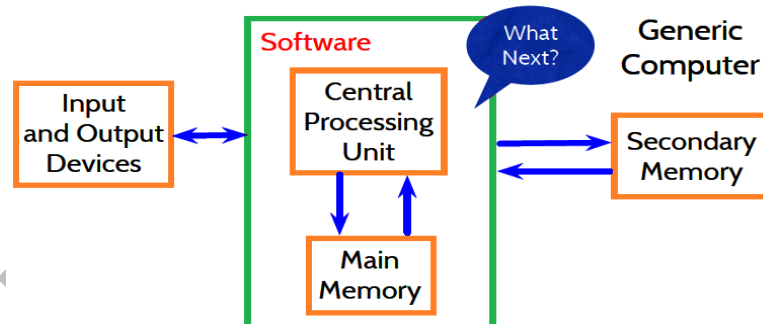
```

name = input('Enter file:')
handle = open(name, 'r')
text = handle.read()
words = text.split()
counts = dict()
for word in words:
    counts[word] = counts.get(word,0) + 1
bigcount = None
bigword = None
for word,count in counts.items():
    if bigcount is None or count > bigcount:
        bigword = word
        bigcount = count
print bigword, bigcount

```

python words.py
Enter file: clown.txt
the 7

Hardware Architecture



Definitions

Central Processing Unit: Runs the Program - The CPU is always wondering “what to do next”? Not the brains exactly - very dumb but very very fast

Input Devices: Keyboard, Mouse, Touch Screen

Output Devices: Screen, Speakers, Printer, DVD Burner

Main Memory: Fast small temporary storage - lost on reboot - aka RAM

Secondary Memory: Slower large permanent storage - lasts until deleted – disk drive / memory stick

Python as a Language

Python is the language of the Python Interpreter and those who can converse with it. An individual who can speak Python is known as a Pythonista. It is a very uncommon skill, and may be hereditary. Nearly all known Pythonistas use software initially developed by Guido van Rossum.

Early Learner: Syntax Errors

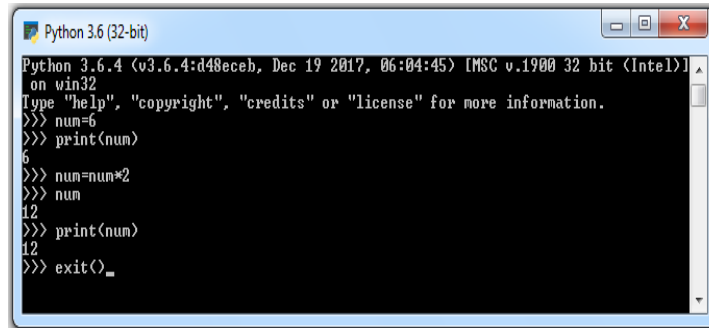
Learn the Python language so we can communicate our instructions to Python.

When you make a mistake, the computer says “syntax error”.

Even though the computer is simple and very fast, but cannot learn. But you are intelligent and can learn. So it is easier for you to learn Python.

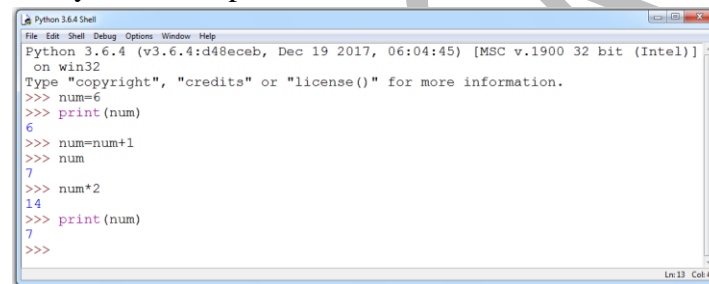
Interacting with Python

- Using Command Line: it is interactive



```
Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017, 06:04:45) [MSC v.1900 32 bit (Intel)]
> on win32
> Type "help", "copyright", "credits" or "license()" for more information.
>>> num=6
>>> print(num)
6
>>> num=num*2
>>> num
12
>>> print(num)
12
>>> exit()
```

- Using IDLE: where you can script



```
Python 3.6.4 Shell
File Edit Shell Debug Options Window Help
Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017, 06:04:45) [MSC v.1900 32 bit (Intel)]
> on win32
> Type "copyright", "credits" or "license()" for more information.
>>> num=6
>>> print(num)
6
>>> num=num+1
>>> num
7
>>> num*2
14
>>> print(num)
14
>>>
```

Elements of Python

Vocabulary / Words - Variables and Reserved words

Sentence structure - valid syntax patterns

Story structure - constructing a program for a purpose

How to count words in a file in Python

```
name = raw_input('Enter file:')
handle = open(name, 'r')
text = handle.read()
words = text.split()
counts = dict()
for word in words:
    counts[word] = counts.get(word,0) + 1
bigcount = None
bigword = None
for word,count in counts.items():
    if bigcount is None or count > bigcount:
        bigword = word
        bigcount = count
```

print bigword, bigcount

Reserved Words

You cannot use reserved words as variable names / identifiers

as	if	in	is	or	and	del	for	def	not	try	elif
else	exec	from	pass	with	raise	assert	break	global	class	except	while
return	print	yield	import		finally		lambda		continue		

Sentences or Lines

x = 2	: Assignment statement
x = x + 2	: Assignment with expression
print x	: Print statement

Variable: x

Operator: +, =

Constant: 2

Reserved Word: print

Programming Paragraphs

Python Scripts

- Interactive Python is good for experiments and programs of 3-4 lines long.
- Most programs are much longer, so we type them into a file and tell Python to run the commands in the file.
- In a sense, we are “giving Python a script”.
- As a convention, we add “.py” as the suffix on the end of these files to indicate they contain Python.

Writing a Simple Program

Interactive versus Script

Interactive

- You type directly to Python one line at a time and it responds

Script

- You enter a sequence of statements (lines) into a file using a text editor and tell Python to execute the statements in the file

Program Steps or Program Flow

- A program is a sequence of steps to be done in order.
- Some steps are conditional - they may be skipped.
- Sometimes a step or group of steps are to be repeated.
- Sometimes we store a set of steps to be used over and over as needed several places throughout the program.

Boolean expressions:

- A boolean expression can be evaluated as True or False. An expression evaluates to False if it is...
 - the constant False,
 - the object None,
 - an empty sequence or collection,
 - or a numerical item of value 0
- Everything else is considered True
- Boolean expressions ask a question and produce a True or False 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
- Remember: “=” is used for assignment

Booleans: True and False

```
>>> type (True)
```

```
<type 'bool'>
```

```
>>> type (true)
```

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

NameError: name 'true' is not defined

```
>>> 2+2==5
```

```
False
```

Note: True and False are of type bool. *The capitalization is required for the booleans!*

Comparison operators: are used in Boolean expressions

== : is equal to?

!= : not equal to

> : greater than

< : less than

>= : greater than or equal to

<= : less than or equal to

is : do two references refer to the same object?

is not: do two references refer to different object?

Can “chain” comparisons:

```
>>> a = 42
```

```
>>> 0 <= a <= 99
```

```
True
```

Logical operators:

and, or, not

```
>>> 2+2==5 or 1+1==2
```

```
True
```

```
>>> 2+2==5 and 1+1==2
```

False

```
>>> not(2+2==5) and 1+1==2
```

True

Note: We do NOT use &&, ||, !, as in C!

If statements

Simple one-line if:

```
if (1+1==2):
```

```
    print "I can add!"
```

```
ifelse:
```

```
if (1+1==2):
```

```
    print "1+1==2"
```

```
    print "I always thought so!"
```

```
else:
```

```
    print "My understanding of math must be faulty!"
```

elif statement:

- Equivalent of "else if" in C

```
elif:
```

```
x = 3
```

```
if (x == 1):
```

```
    print "one"
```

```
elif (x == 2):
```

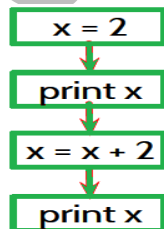
```
    print "two"
```

```
else:
```

```
    print "many"
```

Sequential Steps

When a program is running, it flows from one step to the next. As programmers, we set up "paths" for the program to follow.



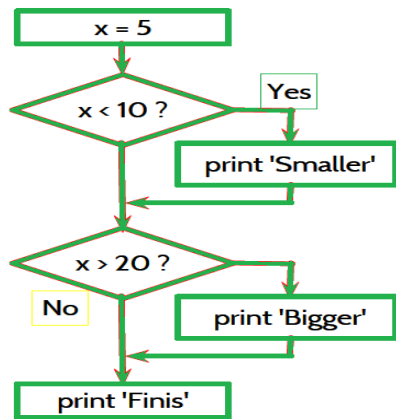
Program:

```
x = 2  
print x  
x = x + 2  
print x
```

Output:

2
4

Conditional Steps



Program:

```

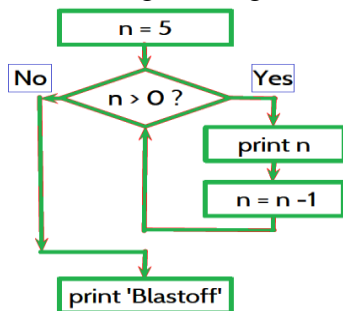
x = 5
if x < 10:
    print 'Smaller'
if x > 20:
    print 'Bigger'
print 'Finis'
  
```

Output:

Smaller
Finis

Repeated Steps

Loops (repeated steps) have iteration variables that change each time through a loop. Often these iteration variables go through a sequence of numbers.



Program:

```

n = 5
while n > 0:
    print n
    n = n - 1
print 'Blastoff!'
  
```

Output:

5
4
3
2
1
Blastoff!

```

name = raw_input('Enter file:')
handle = open(name, 'r')
text = handle.read()
words = text.split()
counts = dict()
  
```

Sequential

```

for word in words:
    counts[word] = counts.get(word,0) + 1
  
```

Repeated

```

bigcount = None
bigword = None
  
```

Sequential

```

for word,count in counts.items():
  
```

Repeated

```

    if bigcount is None or count > bigcount:
        bigword = word
        bigcount = count
  
```

Conditional

```

print bigword, bigcount
  
```

Sequential

```

name = raw_input('Enter file:') # A word used to read data from a user
handle = open(name, 'r')
text = handle.read()
words = text.split()
counts = dict()
for word in words:
    counts[word] = counts.get(word,0) + 1 # A sentence about updating the counts
  
```

```

bigcount = None
bigword = None
for word, count in counts.items():
    if bigcount is None or count > bigcount: } #A paragraph to find the largest item
        bigword = word
        bigcount = count
print bigword, bigcount

```

Variables, Expressions, and Statements

Values and types

A value is one of the basic things a program works with, like a letter or a number.

Eg: 1, 2, and “Hello, World!”

These values belong to different types:

2 is an integer, and “Hello, World!” is a string,

The interpreter can tell what type a value has

```

>>> type('Hello, World!')      #strings belong to the type str
<class 'str'>
>>> type('17')
<class 'str'>
>>> type("3.3")
<class 'str'>
>>> type(17)
<class 'int'>      # integers belong to the type int
>>> type(3.2)
<class 'float'>    # numbers with a decimal point belong to a type called float, (represented
in a format called floating point)
>>> print(1,000,000)    #Python interprets 1,000,000 as a comma separated sequence of integers
1 0 0

```

Constants

Fixed values such as numbers, letters, and strings are called “constants” because their value does not change

Numeric constants

```
>>> print 123
```

```
123
```

```
>>> print 98.6
```

```
98.6
```

String constants use single quotes (') or double quotes (")

```
>>> print 'Hello world'
```

```
Hello world
```

Variables

A variable is a named place in the memory where a programmer can store data and later retrieve the data using the variable “name”

Programmers get to choose the names of the variables

You can change the contents of a variable in a later statement



Python Variable Name Rules

- Must start with a letter or underscore _
- Must consist of letters and numbers and underscores
- Case Sensitive

Good: sum sum_of_n

Bad: 2sum #sign var.12

Different: sum Sum SUM

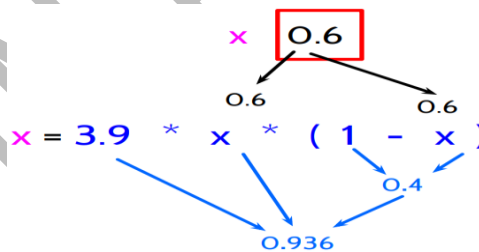
Assignment Statements

We assign a value to a variable using the assignment statement (=)

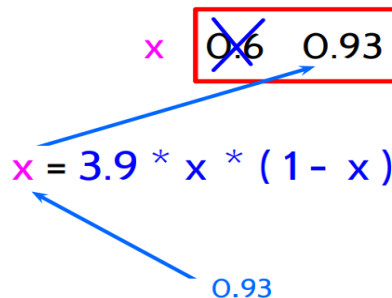
An assignment statement consists of an expression on the right-hand side and a variable to store the result

$$x = 3.9 * x * (1 - x)$$

A variable is a memory location used to store a value (0.6)



A variable is a memory location used to store a value. The value stored in a variable can be updated by replacing the old value (0.6) with a new value (0.93).



Numeric Expressions

Asterisk is multiplication

Exponentiation (raise to a power) looks different from in math.

Operator	Operation
+	Addition
-	Subtraction
*	Multiplication
/	Division
**	Power
%	Remainder

```

>>> x = 2
>>> x = x + 2
>>> print x
4
>>> y = 440 * 12
>>> print y
5280
>>> z = y / 1000
>>> print z
5
>>> j = 23
>>> k = j % 5
>>> print k
3
>>> print 4 ** 3
64

```

Order of Evaluation

- When we string operators together - Python must know which one to do first
- This is called “operator precedence”
- Which operator “takes precedence” over the others?

x = 1 + 2 * 3 - 4 / 5 ** 6

Operator Precedence Rules

Highest precedence rule to lowest precedence rule:

- Parenthesis
- Exponentiation
- Multiplication, Division, and Remainder
- Addition and Subtraction
- Left to right

```

>>> x = 1 + 2 ** 3 / 4 * 5
>>> print x
11

```

```
>>>
```

1 + 2 ** 3 / 4 * 5

1 + 8 / 4 * 5

1 + 2 * 5

1 + 10

11

- Remember the rules top to bottom
 - When writing code - use parenthesis
 - When writing code - keep mathematical expressions simple enough that they are easy to understand
 - Break long series of mathematical operations up to make them more clear
- Question: $x = 1 + 2 * 3 - 4 / 5$

Python Integer Division is Weird!

Integer division

Floating point division produces floating point numbers

```
>>> print (10 / 2)
```

5.0

```
>>> print (9 / 2)
```

4.5

```
>>> print( 99 / 100)
```

0.99

```
>>> print (10.0 / 2.0)
```

5.0

```
>>> print (99.0/100.0)
```

0.99

Mixing Integer and Floating

When you perform an operation where one operand is an integer and the other operand is a floating point, the result is a floating point

The integer is converted to a floating point before the operation

```
>>> print 99 / 100
```

0

```
>>> print 99 / 100.0
```

0.99

```
>>> print 99.0 / 100
```

0.99

```
>>> print 1 + 2 * 3 / 4.0 - 5
```

-2.5

```
>>>
```

What does “Type” Mean?

- In Python variables, literals and constants have a “type”
- Python knows the difference between an integer number and a string
- For example “+” means “addition” if something is a number and “concatenate” if something is a string

```
>>> ddd = 1 + 4
```

```
>>> print ddd
```

```
5
```

```
>>> eee = 'hello ' + 'there'
```

```
>>> print eee
```

```
hello there
```

Type Matters

- Python knows what “type” each value is
- Some operations are prohibited
- Cannot “add 1” to a string
- We can ask Python what type something is by using the type() function

```
>>> eee = 'hello ' + 'there'
```

```
>>> eee = eee + 1
```

```
Traceback (most recent call last): File "<stdin>", line 1, in <module>
```

```
TypeError: cannot concatenate 'str' and 'int' objects
```

```
>>> type(eee)
```

```
<type 'str'>
```

```
>>> type('hello')
```

```
<type 'str'>
```

```
>>> type(1)
```

```
<type 'int'>
```

Several Types of Numbers

Numbers have two main types

- Integers are whole numbers:
-14, -2, 0, 1, 100, 401233
- Floating Point Numbers have decimal parts:
-2.5 , 0.0, 98.6, 14.0

There are other number types - they are variations on float and integer

```
>>> xx = 1
```

```
>>> type(xx)
```

```
<type 'int'>
```

```
>>> temp = 98.6
```

```
>>> type(temp)
```

```
<type 'float'>
```

```
>>> type(1)
```

```
<type 'int'>
>>> type(1.0)
<type float'>
>>>
```

Type Conversions

- When you put an integer and floating point in an expression, the integer is implicitly converted to a float
- You can control this with the built-in functions `int()` and `float()`

```
>>> print float(99) / 100
0.99
>>> i = 42
>>> type(i)
<type 'int'>
>>> f = float(i)
>>> print f
42.0
>>> type(f)
<type 'float'>
>>> print 1 + 2 * float(3) / 4 - 5
-2.5
>>>
```

String Conversions

- You can also use `int()` and `float()` to convert between strings and integers
- You will get an error if the string does not contain numeric characters

```
>>> sval = '123'
>>> type(sval)
<type 'str'>
>>> print sval + 1
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
TypeError: cannot concatenate 'str' and 'int'
>>> ival = int(sval)
>>> type(ival)
<type 'int'>
>>> print ival + 1
124
>>> nsv = 'hello bob'
>>> niv = int(nsv)
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
ValueError: invalid literal for int()
```

User Input

- We can instruct Python to pause and read data from the user using the `input()` function
- The `input()` function returns a string

```
nam = input('Who are you?')
```

```
print 'Welcome', nam
```

```
Who are you? Charles
```

```
Welcome Charles
```

Converting User Input

If we want to read a number from the user, we must convert it from a string to a number using a type conversion function

Later we will deal with bad input data

```
inp = input('Europe floor?')
```

```
usf = int(inp) + 1
```

```
print ('US floor', usf)
```

```
Europe floor? 0
```

```
US floor 1
```

Comments in Python

- Anything after a `#` is ignored by Python
- Why comment?
 - Describe what is going to happen in a sequence of code
 - Document who wrote the code or other ancillary information
 - Turn off a line of code - perhaps temporarily

```
# Get the name of the file and open it
```

```
name = raw_input('Enter file:')
```

```
handle = open(name, 'r')
```

```
text = handle.read()
```

```
words = text.split()
```

```
# Count word frequency
```

```
counts = dict()
```

```
for word in words:
```

```
    counts[word] = counts.get(word,0) + 1
```

```
# Find the most common word
```

```
bigcount = None
```

```
bigword = None
```

```
for word,count in counts.items():
```

```
    if bigcount is None or count > bigcount:
```

```
        bigword = word
```

```
        bigcount = count
```

```
# All done
```

```
print bigword, bigcount
```

String Operations

- Some operators apply to strings
 - + implies “concatenation”
 - * implies “multiple concatenation”
- Python knows when it is dealing with a string or a number and behaves appropriately

```
>>> print 'abc' + '123'
```

```
abc123
```

```
>>> print 'Hi' * 5
```

```
HiHiHiHiHi
```

```
>>>
```

Mnemonic Variable Names

Since we programmers are given a choice in how we choose our variable names, there is a bit of “best practice”

We name variables to help us remember what we intend to store in them (“mnemonic” = “memory aid”)

This can confuse beginning students because well-named variables often “sound” so good that they must be keywords

```
x1q3z9ocd = 35.0
```

```
x1q3z9afd = 12.50
```

```
x1q3p9afd = x1q3z9ocd * x1q3z9afd
```

```
print x1q3p9afd
```

```
a = 35.0
```

```
b = 12.50
```

```
c = a * b
```

```
print c
```

```
hours = 35.0
```

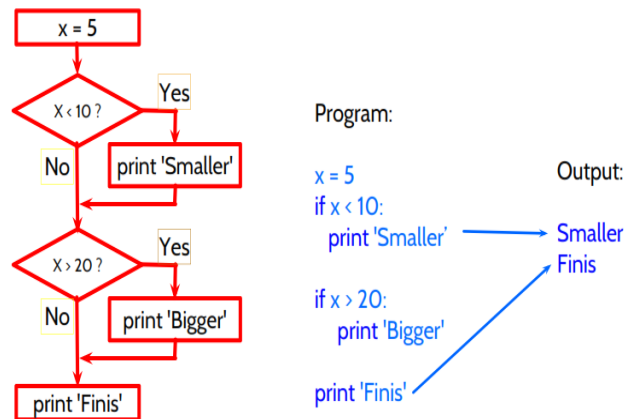
```
rate = 12.50
```

```
pay = hours * rate
```

```
print pay
```

Conditional Execution

Conditional Steps



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
- Remember: “=” is used for assignment

x = 5

if x == 5 :

 print 'Equals 5' #Equals 5

if x > 4 :

 print 'Greater than 4' # Greater than 4

if x >= 5 :

 print 'Greater than or Equals 5' #Greater than or Equals 5

if x < 6 : print 'Less than 6'

 # Less than 6

if x <= 5 :

 print 'Less than or Equals 5' # Less than or Equals 5

if x != 6 :

 print 'Not equal 6' # Not equal 6

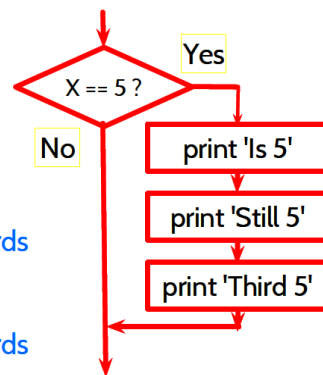
One-Way Decisions

```

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



Indentation

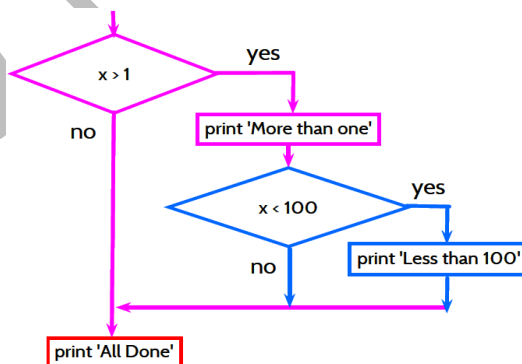
- Increase indent: indent after an if statement or for statement (after :)
- Maintain indent: to indicate the scope of the block (which lines are affected by the if/for)
- Reduce indent: *back to the level of the if statement or for statement* to indicate the end of the block
- Blank lines are ignored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

```
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)
print ('All Done')
```

Note: **increase** / **maintain** after if or for and **decrease** to indicate end of block

Nested Decisions

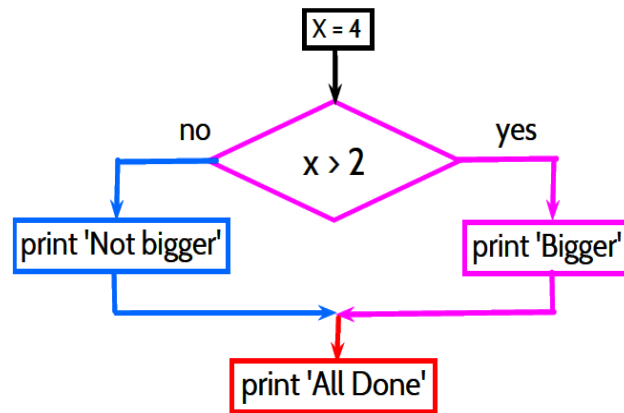
```
x = 42
if x > 1 :
    print ('More than one')
    if x < 100 :
        print ('Less than 100')
print ('All done')
```



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'

```

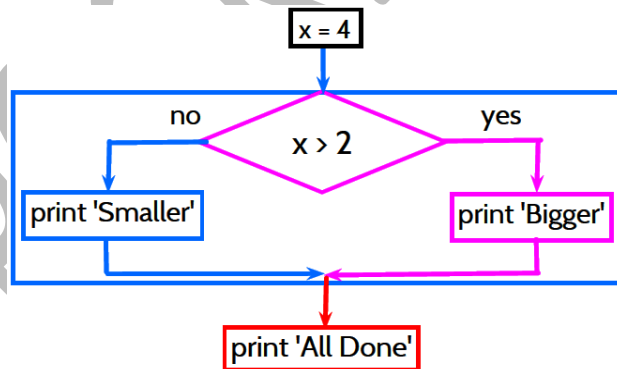
x = 4

```

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

```

print 'All done'

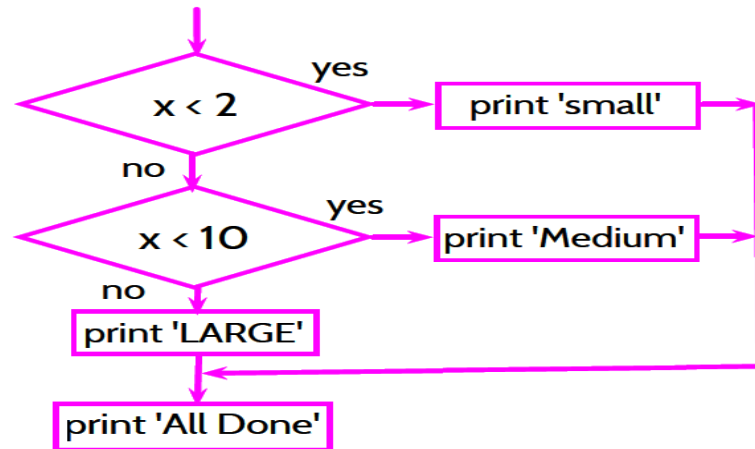


Multi-way:

```

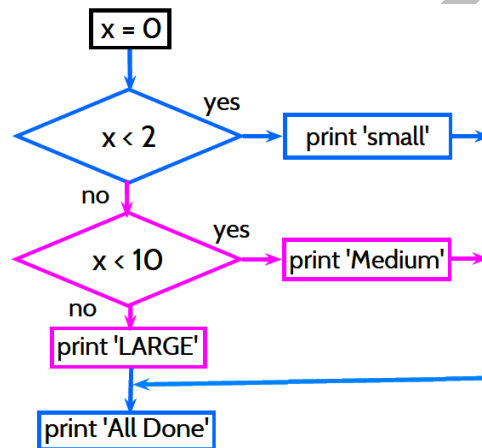
if x < 2 :
    print 'small'
elif x < 10 :
    print 'Medium'
else :
    print 'LARGE'
print 'All done'

```



```

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



Multi-way

Eg1: # No Else

x = 5

```

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

Eg 2:

```

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'
```

The try / except Structure

- You surround the code of statements which can produce an error 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

```
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.

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

Output:

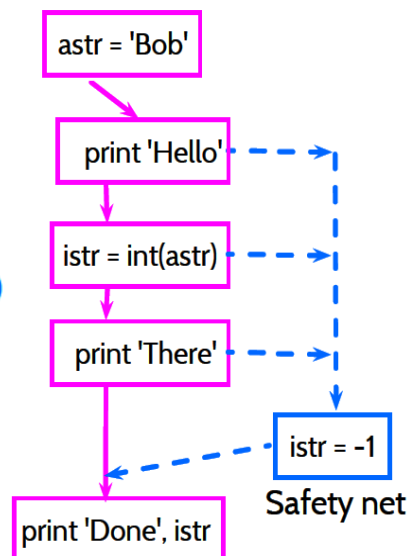
First -1

Second 123

try / except

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

print 'Done', istr
```



Sample try / except

```
num = input('Enter a number: ')
```

```

try:
    ival = int(num)
except:
    ival = -1
if ival > 0 :
    print ('Nice work')
else:
    print ('Not a number')

```

Output 1:

Enter a number: 42

Nice work

Output 2:

Enter a number: forty-two

Not a number

Exercise 1

Write a program to prompt the user for hours and rate per hour to compute gross pay.

Enter Hours: 35

Enter Rate: 2.75

Pay: 96.25

Exercise 2

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 3

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

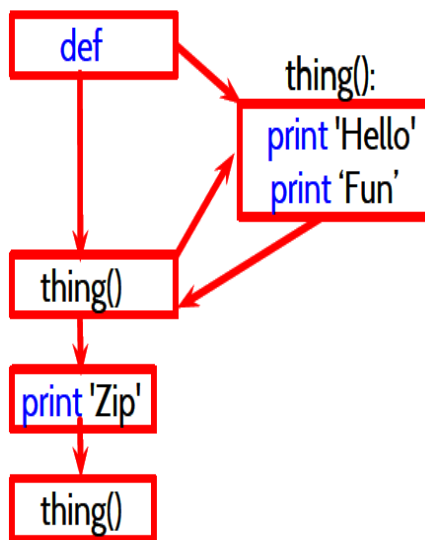
Enter Hours: forty

Error, please enter numeric input

Functions

Stored (and reused) Steps

We call these reusable pieces of code “functions”



Program:

```
def thing():
    print 'Hello'
    print 'Fun'
```

```
thing()
print 'Zip'
thing()
```

Output:

```
Hello
Fun
Zip
Hello
Fun
```

Python Functions

- There are two kinds of functions in Python.
 - > Built-in functions that are provided as part of Python -input(), type(), float(), int() ...
 - > Functions that we define ourselves and then use
- We treat the built-in function names as “new” reserved words (i.e., we avoid them as variable names)

Function Definition

In Python a function is some reusable code that takes arguments(s) as input, does some computation, and then returns a result or results

We define a function using the def reserved word

We call/invoke the function by using the function name, parentheses, and arguments in an expression

Max Function

```
>>> big = max('Hello world')
```

```
>>> print big
```

```
w
```

A function is some stored code that we use. A function takes some input and produces an output

Guido wrote this code

Building our Own Functions

We create a new function using the def keyword followed by optional parameters in parentheses

We indent the body of the function

This defines the function but *does not execute the body of the function*

```
def print_lyrics():
    print ("I'm a lumberjack, and I'm okay.")
    print ('I sleep all night and I work all day.')
```

```
x = 5
```

```
print ('Hello')
```

```
def print_lyrics():
```

```
    print ("I'm a lumberjack, and I'm okay.")
    print ('I sleep all night and I work all day.')
print ('Yo')
x = x + 2
print x
```

Definitions and Uses

Once we have defined a function, we can call (or invoke) it as many times as we like

This is the store and reuse pattern

```
x = 5
print 'Hello'
def print_lyrics():
    print ("I'm a lumberjack, and I'm okay.")
    print ('I sleep all night and I work all day.')
print 'Yo'
print_lyrics()
x = x + 2
print x
```

Arguments

An argument is a value we pass into the function as its input when we call the function

We use arguments so we can direct the function to do different kinds of work when we call it at different times

We put the arguments in parentheses after the name of the function