

- Variables and Assignment
 - Loops, including the Accumulation Pattern
 - Types
-
- Robotics: motion commands as an example of the *input-compute-output* pattern

Please sit with a NEW partner (not your robot partner)

Check your IDLE Configuration

- Verify IDLE shortcut:
 - ▣ Launch IDLE (Start → All Programs → Python 2.6 → IDLE)
 - ▣ In IDLE, choose File → Open...
 - ▣ What directory does the “Open Dialog” start in?
 - CSSE 120 or another good place? Excellent, help a neighbor in need
 - Python26? Follow step 4 (and ONLY step 4) at
 - <http://www.rose-hulman.edu/class/csse/resources/Python/installation.htm>
- Set IDLE to always do a Save when you choose *Run*
 - ▣ In IDLE, choose the *Options* menu, then *Configure IDLE...*
 - ▣ Select the *General* tab
 - ▣ Select the “No prompt” radio button under *Autosave Preferences*

Review: The Python Interpreter

- What it does:

- ▣ Takes in Python commands
- ▣ Converts them to 0s and 1s for the “CPU”
- ▣ Gets answer back from “CPU”


- How we'll use it:

- ▣ IDLE's Python *shell*—lets us “talk with” the interpreter
 - `>>>` is the Python *prompt*
- ▣ Saving and running a module in a file

Review: Saving Programs

- ❑ IDLE's interactive Program Shell is good for trying out snippets of code
 - ❑ But it is annoying to keep retyping, so ...
- ❑ Can save definitions in separate files
 - ❑ Called *modules* or *scripts*
 - ❑ In IDLE, use *File* → *New Window*, include the *.py* when saving
- ❑ Can edit in any text editor (like Notepad++), or ...
- ❑ Can use an *integrated development environment(IDE)*
 - ❑ Recognizes what you type
 - ❑ Tries to help
 - ❑ Examples: IDLE, Eclipse

Review: Running Programs

- Like typing in all the lines, but easier
- One way: Open file in IDLE and run it
 - ▣ *File* → *Open...*, then select the file
 - ▣ *Run* → *Run Module* (or simply **F5**)
 - Output appears in the interactive Program Shell
- Another way: type `import <module>` at prompt
 - ▣ Replace <module> with name of module, don't type the ".py"
 - ▣ Example: `import chaos` 
 - ▣ Runs the code in the imported file
 - ▣ Note the `.pyc` files in your Python folder
 - A partially translated version of your file, called *byte code*
 - Interpreter saves this to make loading faster next time
- Yet another way: `double-click the .py or .pyc file`

This example assumes that you have a file named *chaos.py* in your default Python folder

Outline of today's session:

chapter 2, some of chapter 3

- Identifiers, Expressions
- *Syntax* (form) versus *Semantics* (meaning)
- *print* statements
- Variables and assignments
- Lists and the *range* function
- Definite loops, counting loops, the *accumulator* pattern
- Basic types: numbers (*int* and *float*)
- Math library
- The *accumulator* pattern
- Robots: motion commands as an example of the *input-compute-output* pattern

Identifiers: Names in Programs

- Uses of *identifiers* so far...
 - Modules
 - Functions
 - Variables
 - Classes
 - Rules for identifiers in Python
 - Start with a letter or `_` (the “underscore character”)
 - Followed by any sequence of letters, numbers, or `_`
 - Case matters! `spam` \neq `Spam` \neq `sPam` \neq `SPAM`
 - Choose descriptive names!
- Q3a-f, Q4

Reserved Words

- Built-in names
- Can't use as regular identifiers
- Python reserved words:

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

Be careful not to redefine function names accidentally

□ Examples:

- *len* – used to find the number of items in a sequence
- *max*
- *min*
- *float* – used to convert a number to a floating point number
- *sleep* – pauses the program for a specified length of time (in the *time* module)
- *sqrt* – square roots (in the *math* module)

Expressions

- Fragments of code that produce or calculate new data values
- Examples
 - ▣ *Literals*: indicate a specific value
 - ▣ *Identifiers*: evaluate to their assigned value
 - ▣ *Compound* expressions using *operators* like:
+ - * / ** %
- Can use parentheses to group

Syntax versus Semantics

print (output) statements

- Programming languages have precise rules for:
 - ▣ *Syntax* (form)
 - ▣ *Semantics* (meaning)
- Computer scientists use *meta-languages* to describe these rules

- Example:

- ▣ Syntax:

- `print`
 - `print <expr>`
 - `print <expr>, <expr>, ..., <expr>`
 - `print <expr>, <expr>, ..., <expr>,`

A “slot” to be filled with any expression

Repeat indefinitely

Note: trailing comma

- ▣ Semantics?

- Is the following allowed? `print "The answer is:", 7 * 3 * 2`

Variables and Assignments

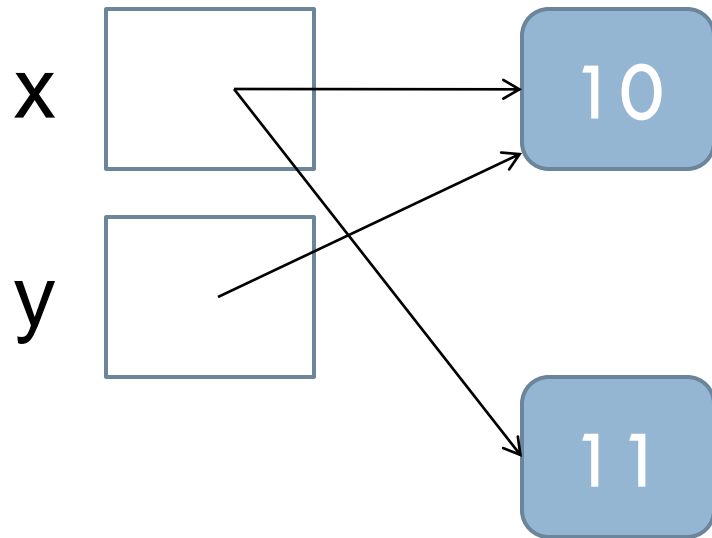
□ Variable

- ▣ Identifier that stores a value
- ▣ A value must be *assigned* to the variable
- ▣ $\langle \text{variable} \rangle = \langle \text{expr} \rangle$ (syntax)

□ Assignment

- ▣ Process of giving a value to a variable
- ▣ Python uses $=$ (equal sign, read as “gets” or “becomes”)
 - ▣ $x = 0.25$
 - ▣ $x = 3.9 * x * (1 - x)$

Variables as sticky notes



$x = 10$

$y = x$

$x = x + 1$

Assignment Statements

1. Simple assignments

- ▣ `<variable> = <expr>`

2. Input assignments

- ▣ `<variable> = input(<prompt>)`

- `temp = input("Enter high temperature for today")`

3. Compound assignments

- ▣ `<var>op=<expr>` means `<var> = <var> op <expr>`

where `op` is one of: `+` `-` `*` `/` `%`

- Example: `total += 5` is the same as `total = total + 5`

4. Simultaneous assignments

- ▣ `<var>, <var>, ..., <var> = <expr>, <expr>, ..., <expr>`

- `sum, diff = x + y, x - y`

- `x, y = y, x`

Sequences

- Python has two kinds of sequences:
 - ▣ *Lists*, for example:
 - [2, 3, 5, 7]
 - ["My", "dog", "has", "fleas"]
 - ▣ *Tuples*, for example
 - (3, -8)
 - ("month", 10)
- We will focus on lists, which can be generated by the *range* function:
 - ▣ range(<expr>)
 - ▣ range(<expr>, <expr>)
 - ▣ range(<expr>, <expr>, <expr>)

Definite loops

- Definition

- **Loop**: a **control structure** for executing a portion of a program multiple times
- **Definite**: Python **knows** how many times to **iterate** the body of the loop

- Syntax for a definite (*for*) loop:

```
for <var> in <sequence> :  
    <body>
```

- Semantics for a definite (*for*) loop:

- **Executes** <body> **once for every element of** <sequence>, **with** <var> **set to that element**

Examples using loops

Loop index

```
>>> for i in [0, 1, 2, 3, 4, 5]:  
    print 2**i
```

Loop sequence

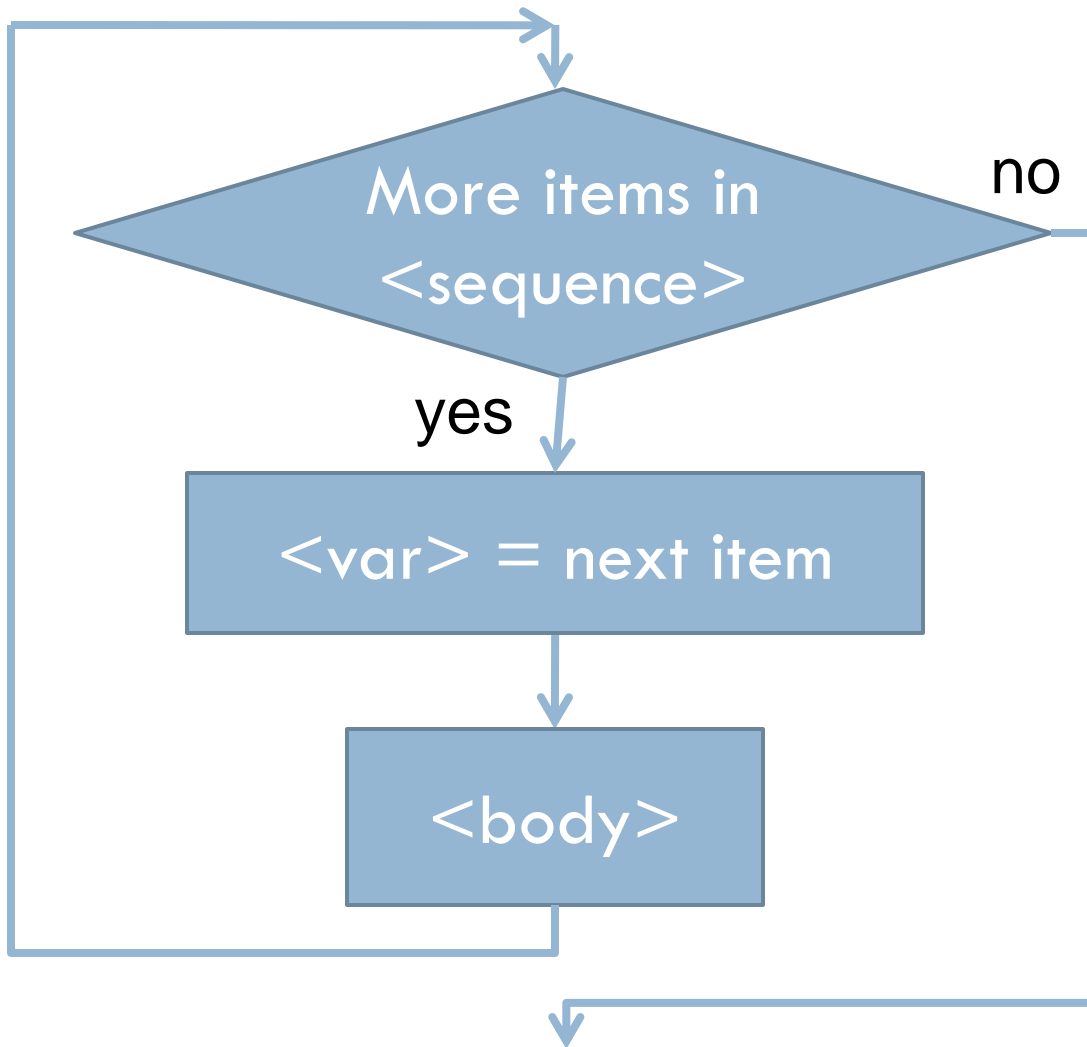
```
>>> for k in range(6):  
    print k, 2**k
```

Loop body

```
>>> for b in ["John", "Paul", "George", "Ringo"]:  
    print b, " was a Beatle"
```

```
>>> for i in range(15, 2, -1):  
    print i,  
    print
```

Flowchart for a for loop



Trace this by hand:

```
sum = 0
for k in range(4):
    sum = sum + k

print a
```

An ***accumulator*** combines parts of a list using looping.

We'll use the above *accumulator* pattern often this term!

Another loop with an accumulator

- Find the sum of the positive odd numbers that are ≤ 13
- Do it together as a class, in IDLE

Data types

□ *Data*

- ▣ Information stored and manipulated on a computer
- ▣ Different kinds of data will be stored and manipulated in different ways

□ *Data type*

- ▣ A particular way of interpreting bits
- ▣ Determines the possible values an item can have
- ▣ Determines the operations supported on items

Numeric data types

```
print "Please enter the count of the given kind of  
    coin."  
  
quarters = input("Quarters: ")  
dimes = input("Dimes: ")  
nickels = input("Nickels: ")  
pennies = input("Pennies: ")  
  
total = quarters * 0.25 + dimes * 0.10 + nickels *  
    .05 + pennies * .01  
  
print "The total value of your change is $", total  
  
print "The total value of your change is $%0.2f" % total
```

Finding the Type of Data

- Built-in function `type(<expr>)` returns the data type of any value
- Find the types of: 3, 3.0, -32, 64.0, “Shrubbery”, [2, 3]
- Why do we need different numerical types?
 - ▣ Operations on *int* are more efficient
 - Compute algorithms for operations on *int* are simple and fast
 - ▣ Counting requires *int*
 - ▣ *Floats* provide approximate values when we need real numbers

Built-in Help

- `dir()`
- `dir(<identifier>)`
- `help(<identifier>)`

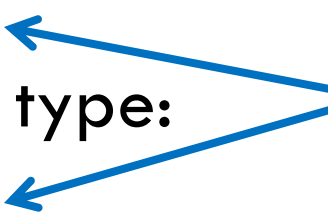
- To see which functions are built-in, type:

```
dir(__builtins__)
```

- To see how to use them, type:

```
help(__builtins__)
```

There are TWO
underscores before and
after the word *builtins*



Some Numeric Operations

Operator	Operation
+	Addition
-	Subtraction
*	Multiplication
/	Division
**	Exponentiation
%	Remainder
//	Do integer division (even on floats)

Function	Operation
abs(x)	Absolute value of x
round(x, y)	Round x to y decimal places
int(x)	Convert x to the int data type
float(x)	Convert x to the float data type

Math library functions

Quadratic formula to find real roots for quadratic equations of the form $ax^2 + bx + c = 0$

□ Solution:

$$x = \frac{-b + \sqrt{b^2 - 4ac}}{2a} \quad x = \frac{-b - \sqrt{b^2 - 4ac}}{2a}$$

- Write out the Python expression for the first formula.
- If you have time, test it IDLE

More math library components

Python	Mathematics	English
pi	π	Approximation of pi
e	e	Approximation of e
sin(x)	$\sin x$	The sine of x
cos(x)	$\cos x$	The cosine of x
tan(x)	$\tan x$	The tangent of x
atan2(y,x)	$\tan^{-1}(y,x)$	Arc tangent (inverse tangent) computes the angle formed by the positive x-axis and the ray from (0,0) to (x,y)
log(x)	$\ln x$	The natural (base e) log of x
log10(x)	$\log_{10} x$	The base 10 log of x
exp(x)	e^x	The exponential of x

EXPLORING WITH PYTHON



Pair Programming

- Working in pairs on a single computer
 - ▣ One person, the *driver*, uses the keyboard
 - ▣ The other person, the *navigator*, watches, thinks, and takes notes
- For hard (or new) problems, this technique
 - ▣ Reduces number of errors
 - ▣ Saves time in the long run
- Works best when partners have similar skill level
- If not, then student with most experience should navigate, while the other student drives.

Food tasting

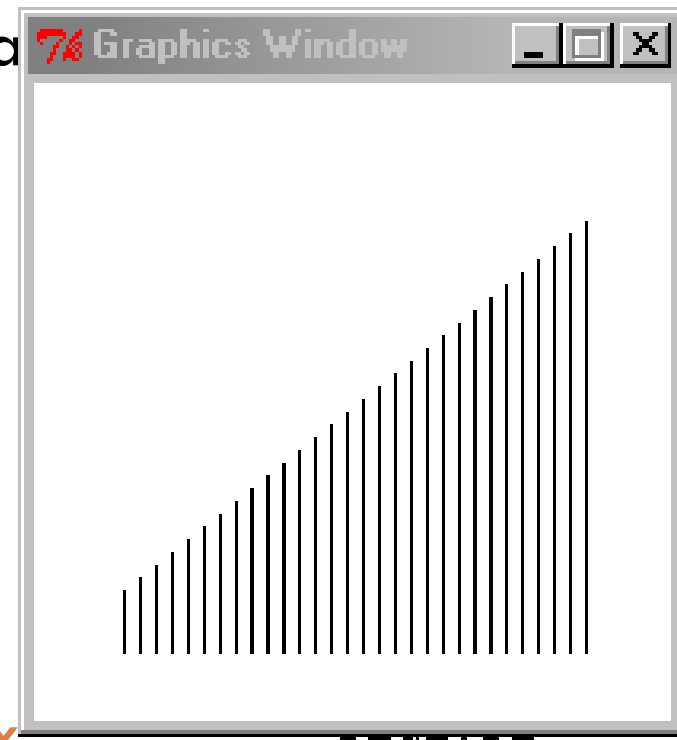
- Suppose you are at food tasting show and are tasting 5 different dishes
- Sampling the dishes in different orders may affect how good they taste
- If you want to try out every possible ordering, how many different orders would there be?
 - ▣ That number is the factorial of 5
 - ▣ $n! = n (n - 1) (n - 2) \dots (1)$
- What type of problem is this?

Accumulating results: factorial

- Work with a partner (pick a driver and navigator)
- Write a Python program called **factorial.py** that
 - ▣ Prompts the user for an integer
 - ▣ Calculates the factorial of the integer
 - $n! = n (n - 1) (n - 2) \dots (1)$
 - ▣ Does **not** use the built-in `math.factorial` function
 - ▣ Outputs the result to the screen
- Driver: email the code to your partner (so each has the program for the open-computer parts of exams)
- Submit one copy of program with both student's names in a program comment.
- Submit it in ANGEL to the **Lessons > Homework > Homework 3 > Factorial Drop Box**

Graphics Exercise with loops

- Trade roles with partner—new driver, new navigator
- Write a program called **barChart.py** that draws a figure like this where the lengths of the lines increase by a constant amount
- Use your previous graphics program as a template: import graphics functions, create a window, etc.
- You may want to use variables to hold current x-coordinate and current line length, and change the values of those variables each time through the loop



□ Homework 3 > Bar Chart Drop Box