Why Program?

Chapter 1

Python for Everybody

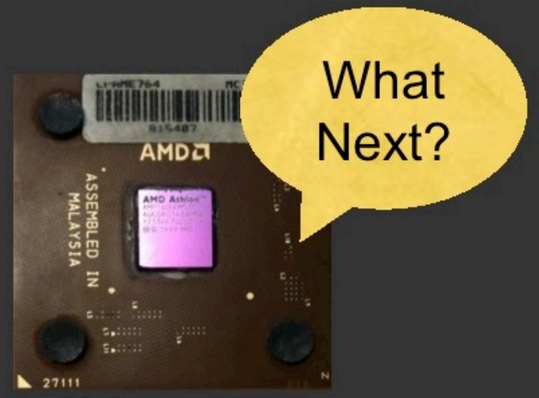
www.py4e.com

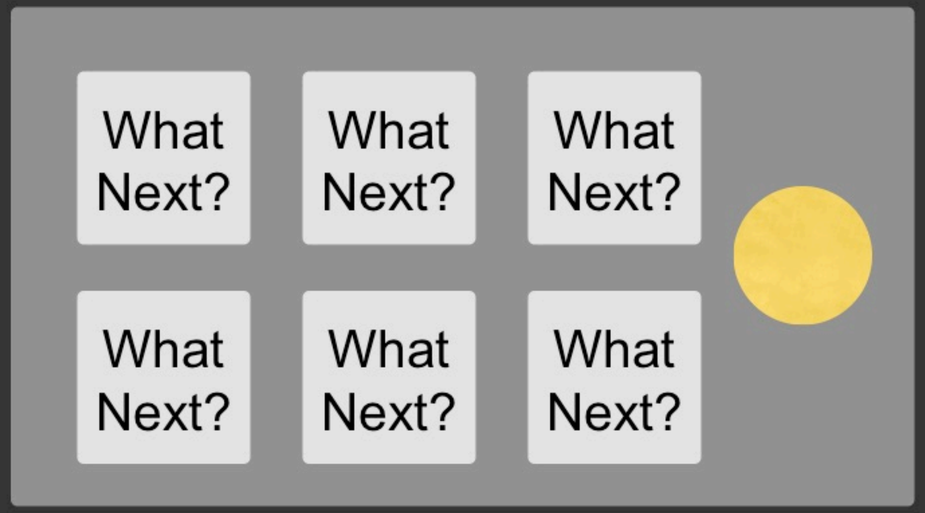
UMSI

CC BY

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 done
- Users have it easy - someone already put many different programs (instructions) into the computer and users just pick the ones they want to use





Programmers Anticipate Needs

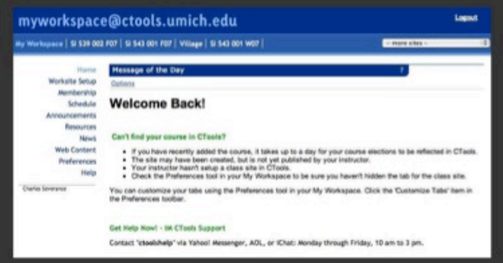
- iPhone Applications are a market
- iPhone Applications have over 3 Billion downloads
- Programmers have left their jobs to be full-time iPhone developers
- Programmers know the ways of the program

Users vs. Programmers

- Users see computers as a set of tools - word processor, spreadsheet, map, to-do list, etc.
- Programmers learn the computer “ways” and the computer language
- Programmers have some tools that allow them to build new tools
- Programmers sometimes write tools for lots of users and sometimes programmers write little “helpers” for themselves to automate a task

Why be a programmer?

- To get some task done - we are the user and programmer
  - Clean up survey data
- To produce something for others to use - a programming job
  - Fix a performance problem in the Sakai software
  - Add a guestbook to a web site



User

Computer
Hardware + Software

Programmer

Data

Information

....

Networks

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.

What is Code? Software? A Program?

- A sequence of stored instructions
  - It is a little piece of our intelligence in the computer
  - 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 Humans...

https://www.youtube.com/watch?v=gwWRjvwlLKg

Programs for Humans...

while music is playing:
    Left hand out and up
    Right hand out and up
    Flip Left hand
    Flip Right hand
    Left hand to right shoulder
    Right hand to left shoulder
    Left hand to back of head
    Right ham to back of head
    Left hand to right hit
    Right hand to left hit
    Left hand on left bottom
    Right hand on right bottom
    Wiggle
    Wiggle
    Jump

https://www.youtube.com/watch?v=gwWRjvwlLKg

Programs for Humans...

while music is playing:
    Left hand out and up
    Right hand out and up
    Flip Left hand
    Flip Right hand
    Left hand to right shoulder
    Right hand to left shoulder
    Left hand to back of head
    Right ham to back of head
    Left hand to right hit
    Right hand to left hit
    Left hand on left bottom
    Right hand on right bottom
    Wiggle
    Wiggle
    Jump

https://www.youtube.com/watch?v=gwWRjvwlLKg

Programs for Humans...

while music is playing:
    Left hand out and up
    Right hand out and up
    Flip Left hand
    Flip Right hand
    Left hand to right shoulder
    Right hand to left shoulder
    Left hand to back of head
    Right hand to back of head
    Left hand to right hip
    Right hand to left hip
    Left hand on left bottom
    Right hand on right bottom
    Wiggle
    Wiggle
    Jump

https://www.youtube.com/watch?v=gwWRjvwlLKg

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

Image: https://www.flickr.com/photos/allan_harris/4908070612/ Attribution-NoDerivs 2.0 Generic (CC BY-ND 2.0)

Programs for Python...

Image: https://www.flickr.com/photos/allan_harris/4908070612/ Attribution-NoDerivs 2.0 Generic (CC BY-ND 2.0)

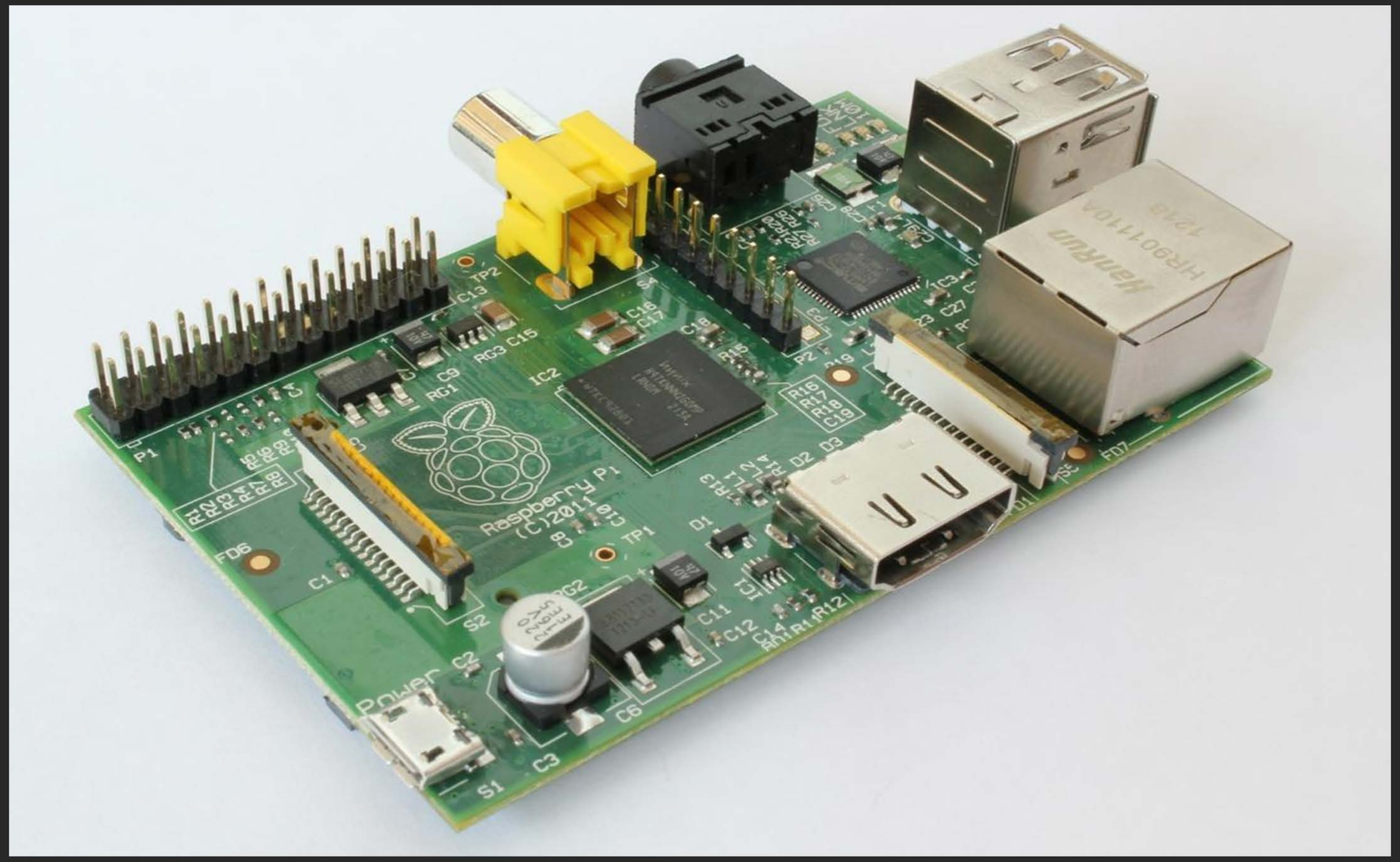
```
name = input('Enter file:')
handle = open(name)

counts = dict()
for line in handle:
    words = line.split()
    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: words.txt
to 16
```

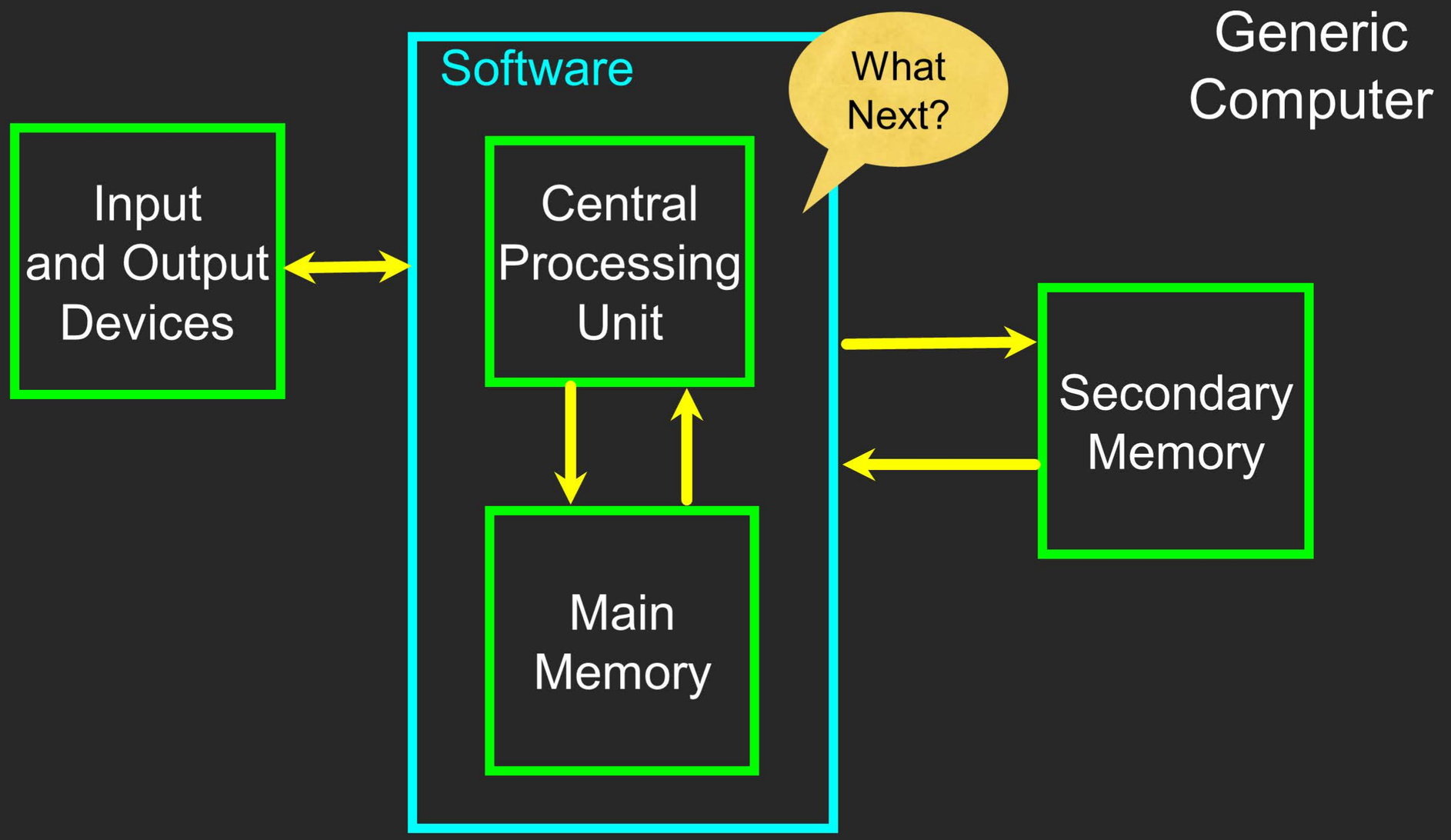
```
python words.py
Enter file: clown.txt
the 7
```

Hardware Architecture

Hardware Architecture

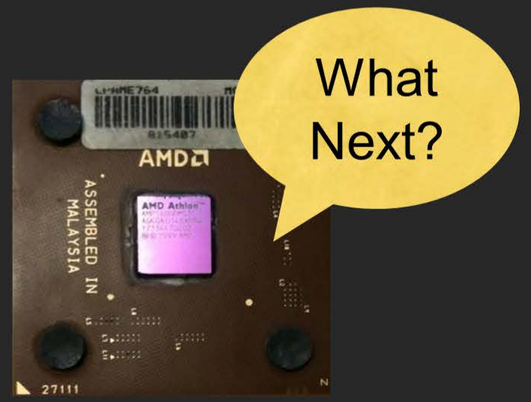


http://upload.wikimedia.org/wikipedia/commons/3/3d/RaspberryPi.jpg

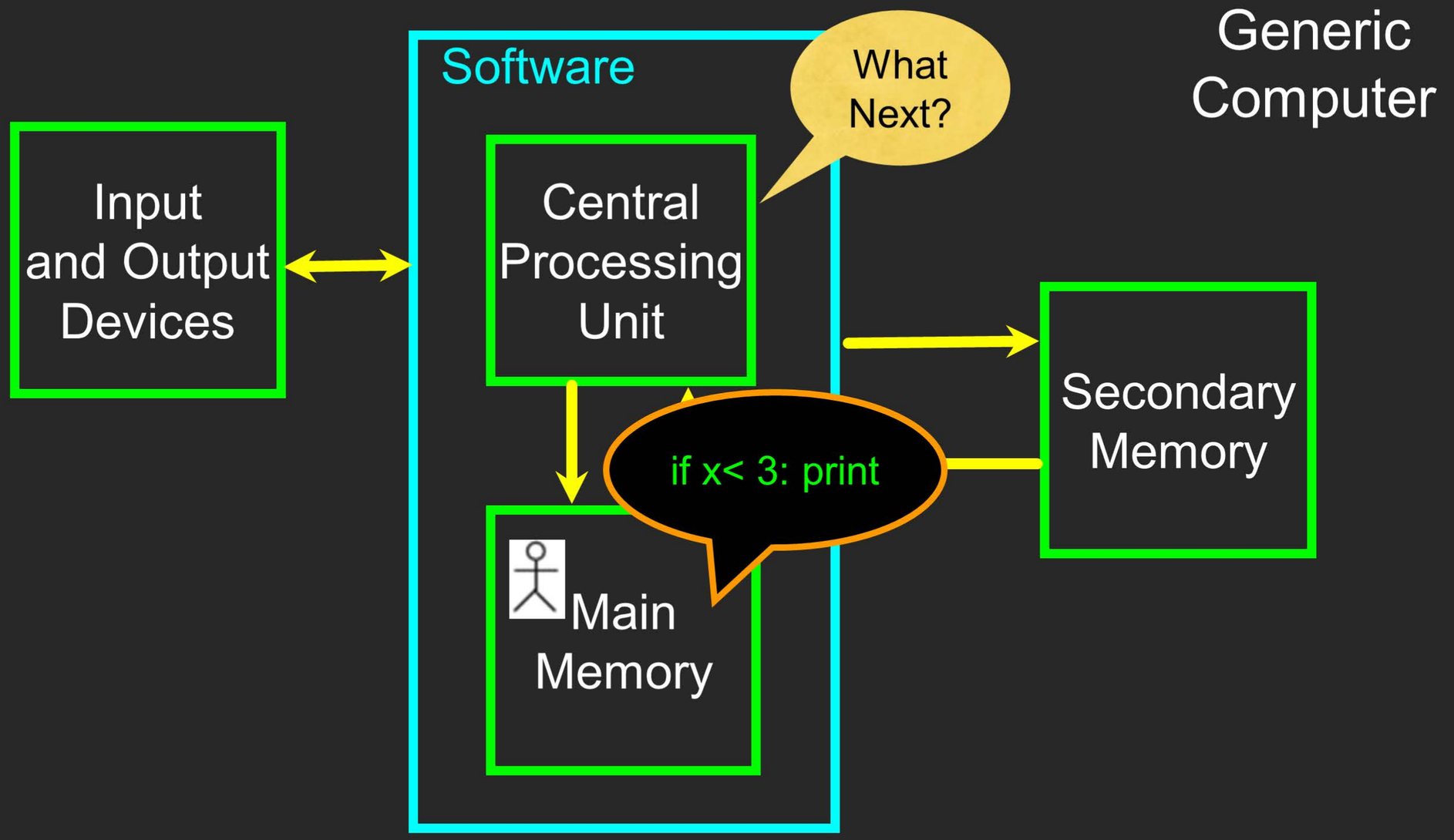

Generic Computer
Software
What Next?
Input and Output Devices
Central Processing Unit
Main Memory
Secondary Memory

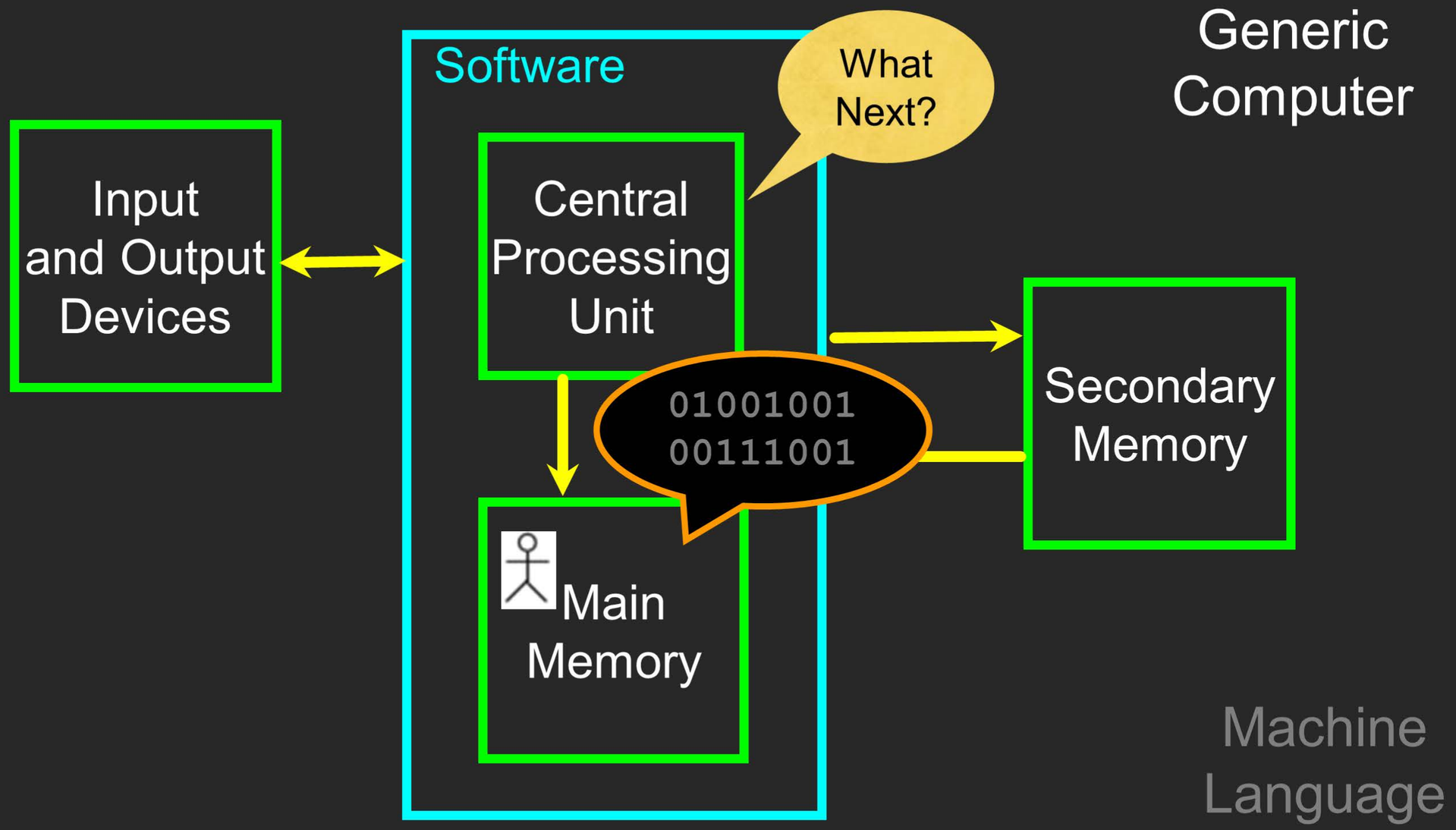
Definitions

- 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

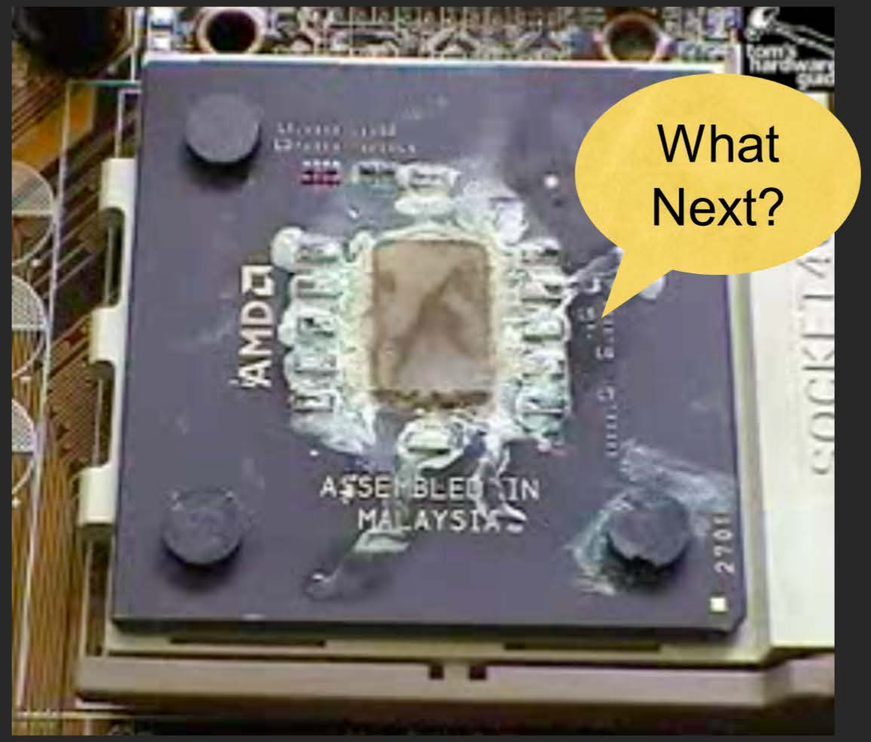


- Input Devices: Keyboard, Mouse, Touch Screen
- Output Devices: Screen, Speakers, Printer, DVD Burner
- Main Memory: Fast small temporary storage - lost on reboot - aka RAM
- Secondary Memory: Slower large permanent storage - lasts until deleted - disk drive / memory stick


Generic Computer
Software
What Next?
Input and Output Devices
Central Processing Unit
Secondary Memory
if x< 3: print
Main Memory

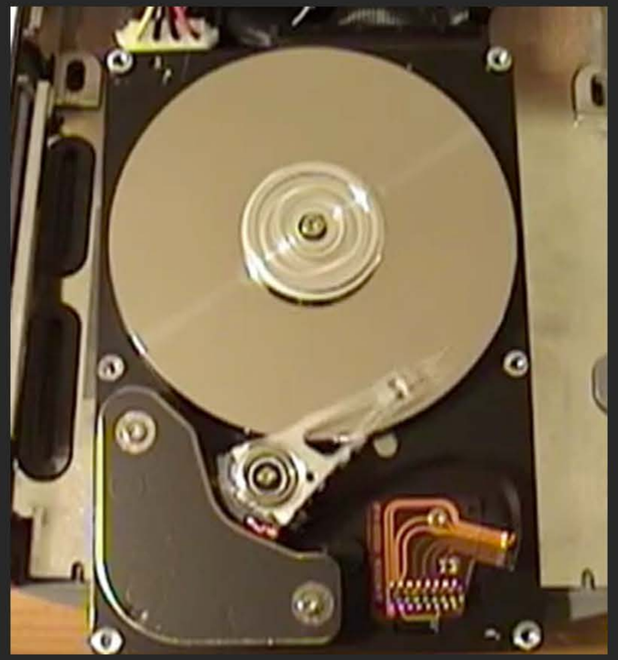

Generic Computer
Software
What Next?
Input and Output Devices
Central Processing Unit
01001001
00111001
Secondary Memory
Main Memory
Machine Language

Totally Hot CPU



http://www.youtube.com/watch?v=y39D4529FM4

Hard Disk in Action



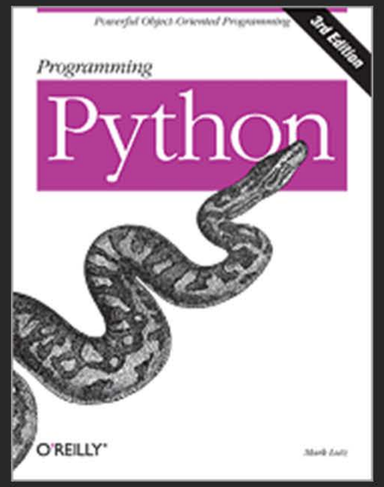
http://www.youtube.com/watch?v=9eMWG3fwiEU

Python as a Language

Parseltongue is the language of serpents and those who can converse with them. An individual who can speak Parseltongue is known as a Parselmouth. It is a very uncommon skill, and may be hereditary. Nearly all known Parselmouths are descended from Salazar Slytherin.

http://harrypotter.wikia.com/wiki/Parseltongue

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

- We need to learn the Python language so we can communicate our instructions to Python. In the beginning we will make lots of mistakes and speak gibberish like small children.

- When you make a mistake, the computer does not think you are “cute”. It says “syntax error” - given that it knows the language and you are just learning it. It seems like Python is cruel and unfeeling.

- You must remember that you are intelligent and can learn. The computer is simple and very fast, but cannot learn. So it is easier for you to learn Python than for the computer to learn English...

Talking to Python

Talking to Python

```
csev$ python3
Python 3.5.1 (v3.5.1:37a07cee5969, Dec  5 2015, 21:12:44)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwinType
"help", "copyright", "credits" or "license" for more information.
>>>
```

What next?

```
csev$ python3
Python 3.5.1 (v3.5.1:37a07cee5969, Dec  5 2015, 21:12:44)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwinType
"help", "copyright", "credits" or "license" for more information.
>>> x = 1
>>> print (x)
1
>>> x = x + 1
>>> print (x)
2
>>> exit()
```

This is a good test to make sure that you have Python correctly installed. Note that quit() also works to end the interactive session.

What Do We Say?

Elements of Python

- Vocabulary / Words - Variables and Reserved words (Chapter 2)
- Sentence structure - valid syntax patterns (Chapters 3-5)
- Story structure - constructing a program for a purpose

```
name = input('Enter file:')
handle = open(name)

counts = dict()
for line in handle:
    words = line.split()
    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, bigword)
```

A short “story” about how to count words in a file in Python

```
python words.py
Enter file: words.txt
to 16
```

Reserved Words

- You cannot use reserved words as variable names / identifiers

```
False   class   return  is      finally
None    if      for     lambda  continue
True    def     from    while   nonlocal
and     del     global  not     with
as      elif    try     or      yield
assert  else    import  pass
break   except  in      raise
```

Sentences or Lines

```
x = 2
x = x + 2
print(x)
```

← Assignment statement

← Assignment with expression

← Print function

Variable Operator Constant Function

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.

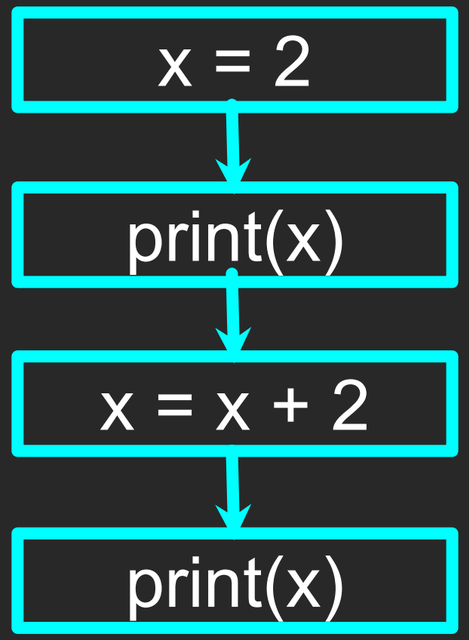
Interactive versus Script

- 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

- Like a recipe or installation instructions, 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 is to be repeated.
- Sometimes we store a set of steps to be used over and over as needed several places throughout the program (Chapter 4).

Sequential Steps



Program:

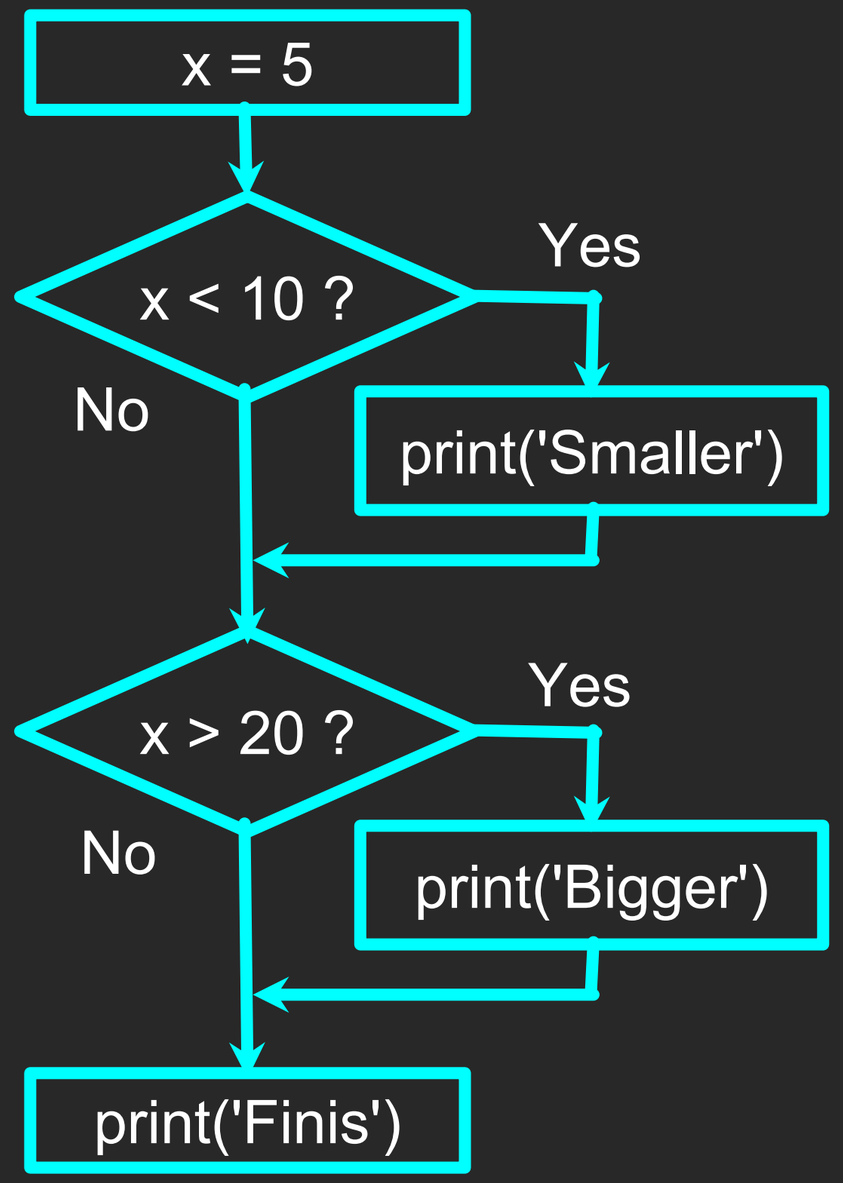
```
x = 2
print(x)
x = x + 2
print(x)
```

Output:

```
2
4
```

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

Conditional Steps



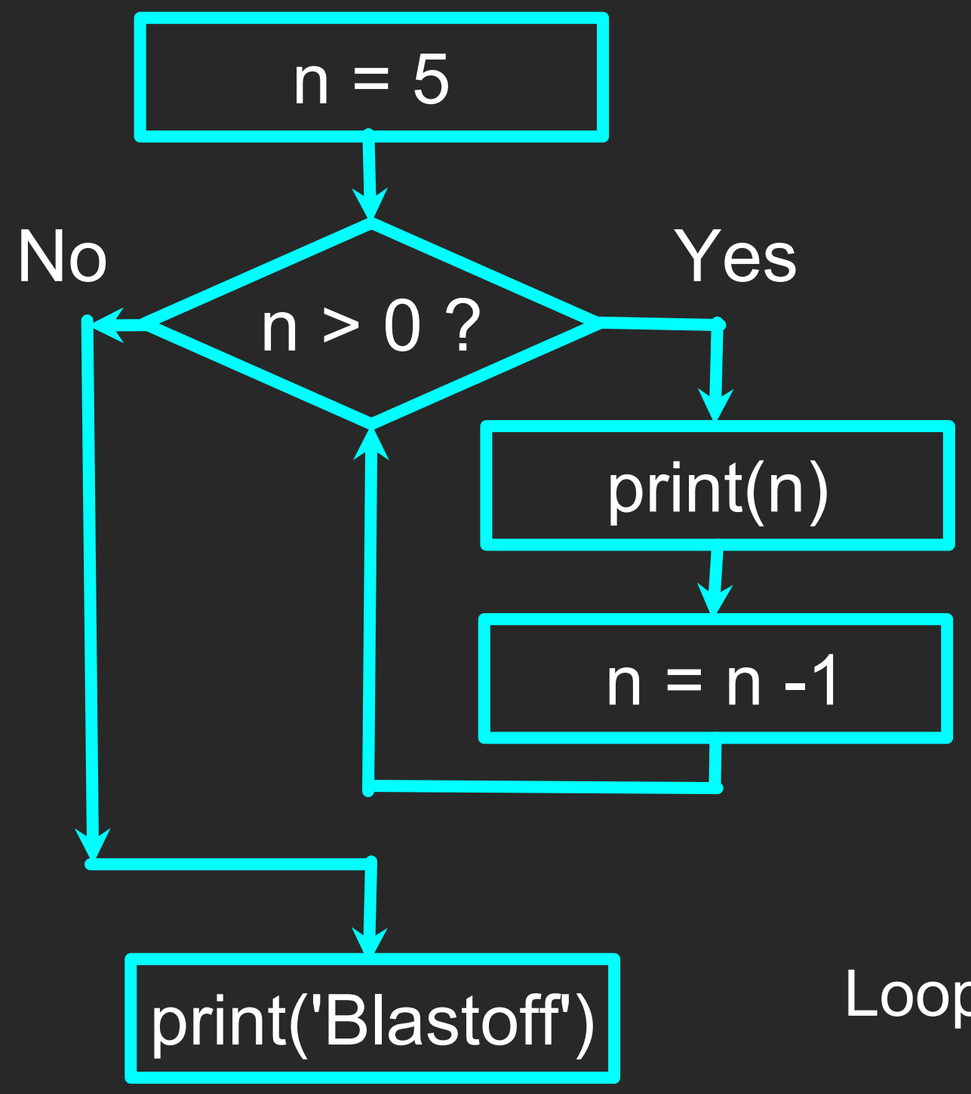
Program:

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

print('Finis')
```

Output:

```
Smaller
Finis
```

Repeated Steps



Program:

```
n = 5
while n > 0 :
    print(n)
    n = n – 1
print('Blastoff!')
```

Output:

5
4
3
2
1
Blastoff!

Loops (repeated steps) have iteration variables that change each time through a loop.

```
name = input('Enter file:')
handle = open(name)

counts = dict()
for line in handle:
    words = line.split()
    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)
```

Sequential

Repeated

Conditional

```
name = input('Enter file:')
handle = open(name, 'r')

counts = dict()
for line in handle:
    words = line.split()
    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)
```

A short Python "Story" about how to count words in a file

A word used to read data from a user

A sentence about updating one of the many counts

A paragraph about how to find the largest item in a list

Summary

- This is a quick overview of Chapter 1
- We will revisit these concepts throughout the course
- Focus on the big picture

Acknowledgements / Contributions

These slides are Copyright 2010- Charles R. Severance (www.dr-chuck.com) of the University of Michigan School of Information and made available under a Creative Commons Attribution 4.0 License. Please maintain this last slide in all copies of the document to comply with the attribution requirements of the license. If you make a change, feel free to add your name and organization to the list of contributors on this page as you republish the materials.

Initial Development: Charles Severance, University of Michigan School of Information

… Insert new Contributors and Translators here

Continue…

Variables, Expressions, and Statements

Chapter 2

Python for Everybody

www.py4e.com

UMSI

CC BY

Constants

- Fixed values such as numbers, letters, and strings, are called "constants" because their value does not change
- Numeric constants are as you expect
- String constants use single quotes (') or double quotes (")

```
>>> print(123)
123
>>> print(98.6)
98.6
>>> print('Hello world')
Hello world
```

Reserved Words

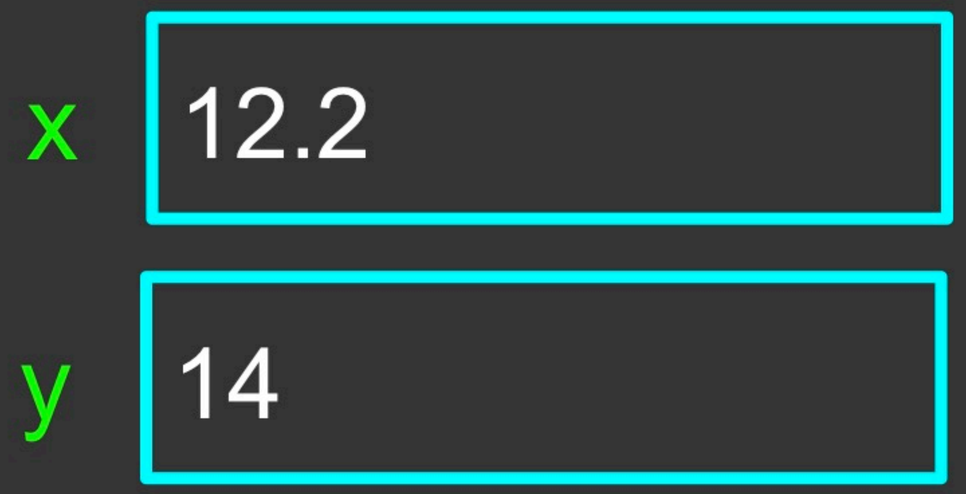
- You cannot use reserved words as variable names / identifiers

```
False   class   return  is      finally
None    if      for     lambda  continue
True    def     from    while   nonlocal
and     del     global  not     with
as      elif    try     or      yield
assert  else    import  pass
break   except  in      raise
```

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

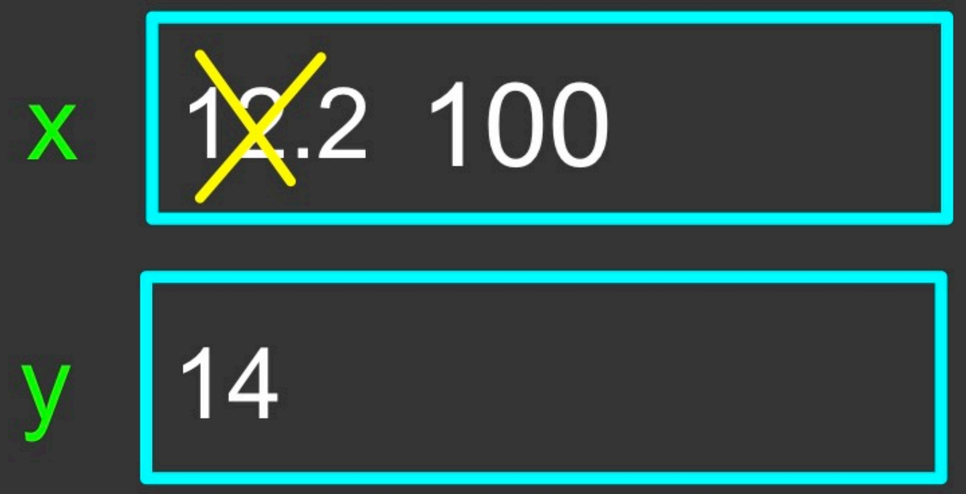
```
x = 12.2
y = 14
```



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

```
x = 12.2
y = 14
x = 100
```



Python Variable Name Rules

Must start with a letter or underscore _

Must consist of letters, numbers, and underscores

Case Sensitive

```
Good:       spam     eggs    spam23     _speed
Bad:        23spam      #sign   var.12
Different:     spam    Spam    SPAM
```

Sentences or Lines

```
x = 2
x = x + 2
print(x)
```

← Assignment statement

← Assignment with expression

← Print statement

Variable Operator Constant Function

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

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

```
x1q3z9ocd = 35.0
x1q3z9afd = 12.50
x1q3p9afd = x1q3z9ocd * x1q3z9afd
print(x1q3p9afd)
```

What is this bit of code doing?

```
x1q3z9ocd = 35.0
x1q3z9afd = 12.50
x1q3p9afd = x1q3z9ocd * x1q3z9afd
print(x1q3p9afd)
```

```
a = 35.0
b = 12.50
c = a * b
print(c)
```

What are these bits of code doing?

```
x1q3z9ocd = 35.0
x1q3z9afd = 12.50
x1q3p9afd = x1q3z9ocd * x1q3z9afd
print(x1q3p9afd)
```

```
a = 35.0
b = 12.50
c = a * b
print(c)
```

What are these bits of code doing?

```
hours = 35.0
rate = 12.50
pay = hours * rate
print(pay)
```

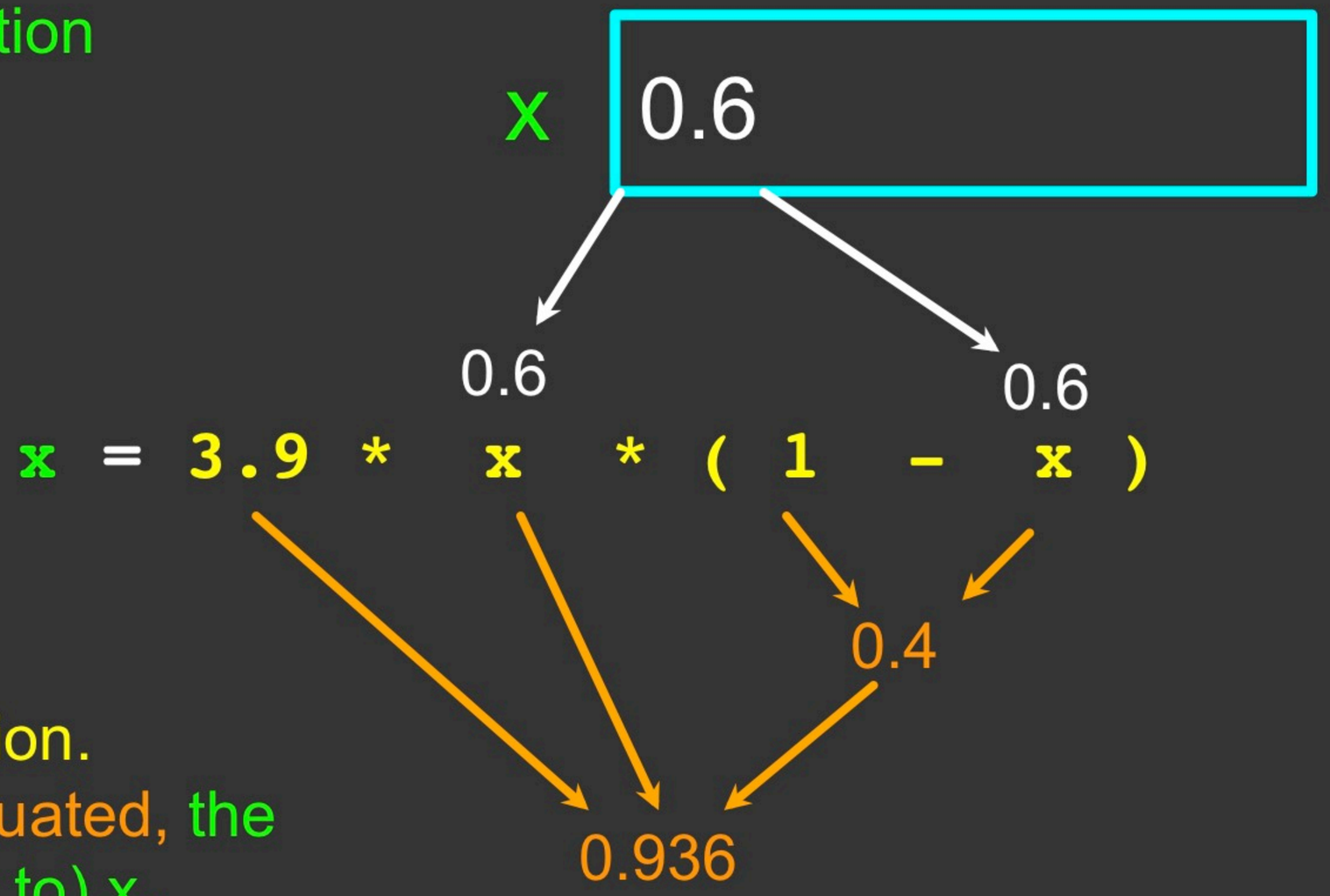
Assignment Statements

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

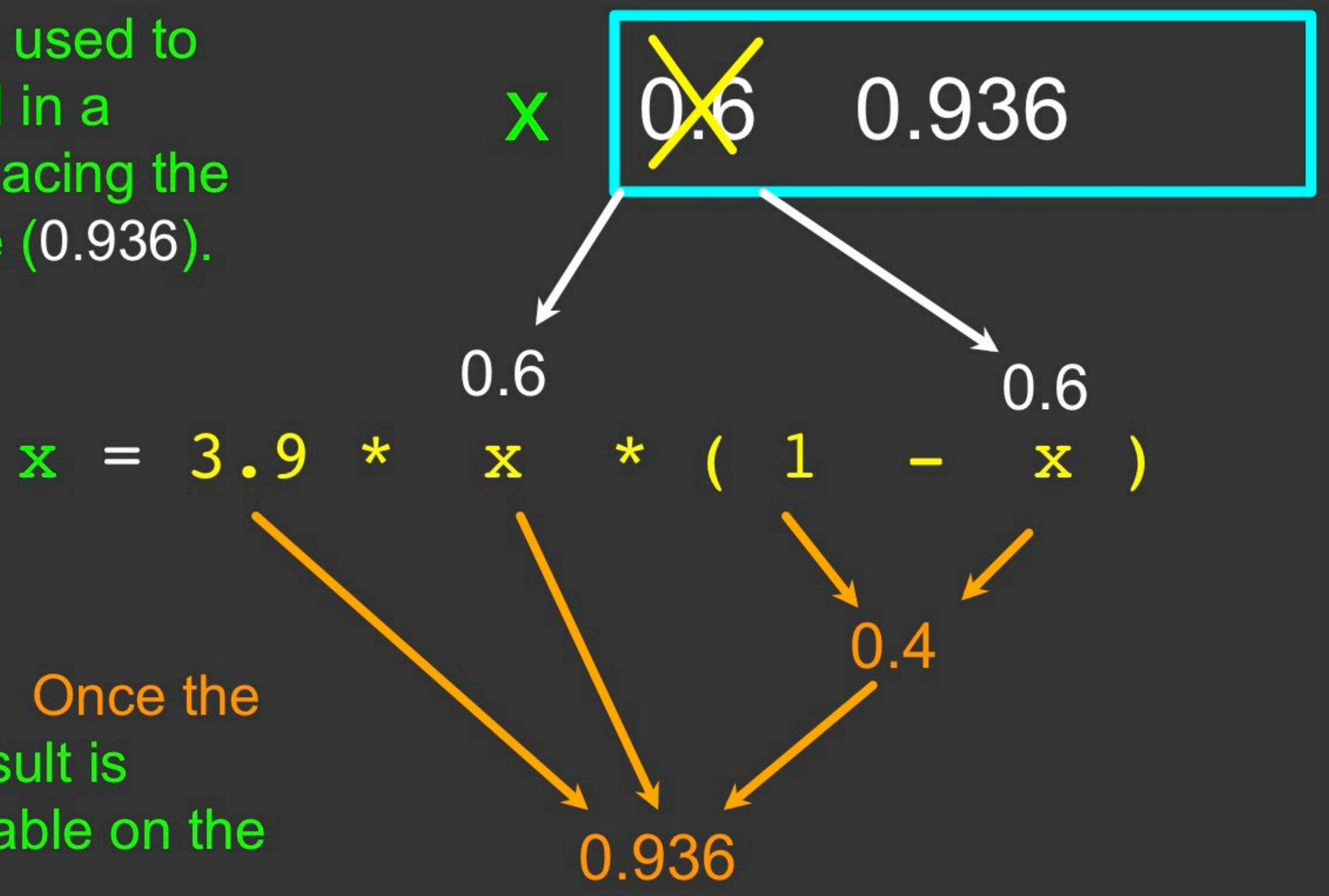
```
x = 3.9 * x * ( 1 - x )
```

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



The right side is an expression.
Once the expression is evaluated, the result is placed in (assigned to) x.

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 (0.936).



The right side is an expression. Once the expression is evaluated, the result is placed in (assigned to) the variable on the left side (i.e., x).

Expressions

Expressions

Numeric Expressions

- Because of the lack of mathematical symbols on computer keyboards - we use “computer-speak” to express the classic math operations
- Asterisk is multiplication
- Exponentiation (raise to a power) looks different than in math

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

Numeric Expressions

```
>>> xx = 2
>>> xx = xx + 2
>>> print(xx)
4
>>> yy = 440 * 12
>>> print(yy)
5280
>>> zz = yy / 1000
>>> print(zz)
5.28
```

```
>>> jj = 23
>>> kk = jj % 5
>>> print(kk)
3
>>> print(4 ** 3)
64
```

```
      4 R 3
5 ) 23
    20
    --
     3
```

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

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:

- Parentheses are always respected
- Exponentiation (raise to a power)
- Multiplication, Division, and Remainder
- Addition and Subtraction
- Left to right

Parenthesis
Power
Multiplication
Addition
Left to Right

```
>>> x = 1 + 2 ** 3 / 4 * 5
>>> print(x)
11.0
>>>
```

Parenthesis
Power
Multiplication
Addition
Left to Right

```
1 + 2 ** 3 / 4 * 5
1 + 8 / 4 * 5
1 + 2 * 5
1 + 10
11
```

Operator Precedence

Parenthesis
Power
Multiplication
Addition
Left to Right

- Remember the rules top to bottom
- When writing code - use parentheses
- 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

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
```

concatenate = put together

Type Matters

- Python knows what "type" everything is
- Some operations are prohibited
- You 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: Can't convert
'int' object to str implicitly
>>> type(eee)
<class'str'>
>>> type('hello')
<class'str'>
>>> type(1)
<class'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)
<class 'int'>
>>> temp = 98.6
>>> type(temp)
<class'float'>
>>> type(1)
<class 'int'>
>>> type(1.0)
<class'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)
199.0
>>> i = 42
>>> type(i)
<class'int'>
>>> f = float(i)
>>> print(f)
42.0
>>> type(f)
<class'float'>
>>>
```

Integer Division

- Integer division produces a floating point result

```
>>> 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
```

Integer division was different in Python 2.x

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)
<class 'str'>
>>> print(sval + 1)
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
TypeError: Can't convert 'int' object
to str implicitly
>>> ival = int(sval)
>>> type(ival)
<class '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()
with base 10: 'x'
```

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? Chuck
Welcome Chuck
```

Making a Program

Making a Program

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 = input('Enter file:')
handle = open(name, 'r')

# Count word frequency
counts = dict()
for line in handle:
    words = line.split()
    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)
```

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

```
# Convert elevator floors
inp = input('Europe floor?')
usf = int(inp) + 1
print('US floor', usf)
```

```
Europe floor? 0
US floor 1
```

Summary

- Type
- Reserved words
- Variables (mnemonic)
- Operators
- Operator precedence
- Integer Division
- Conversion between types
- User input
- Comments (#)

Exercise

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
```

Acknowledgements / Contributions

These slides are Copyright 2010- Charles R. Severance (www.dr-chuck.com) of the University of Michigan School of Information and made available under a Creative Commons Attribution 4.0 License. Please maintain this last slide in all copies of the document to comply with the attribution requirements of the license. If you make a change, feel free to add your name and organization to the list of contributors on this page as you republish the materials.

Initial Development: Charles Severance, University of Michigan School of Information

… Insert new Contributors and Translators here

...

Conditional Execution

Chapter 3

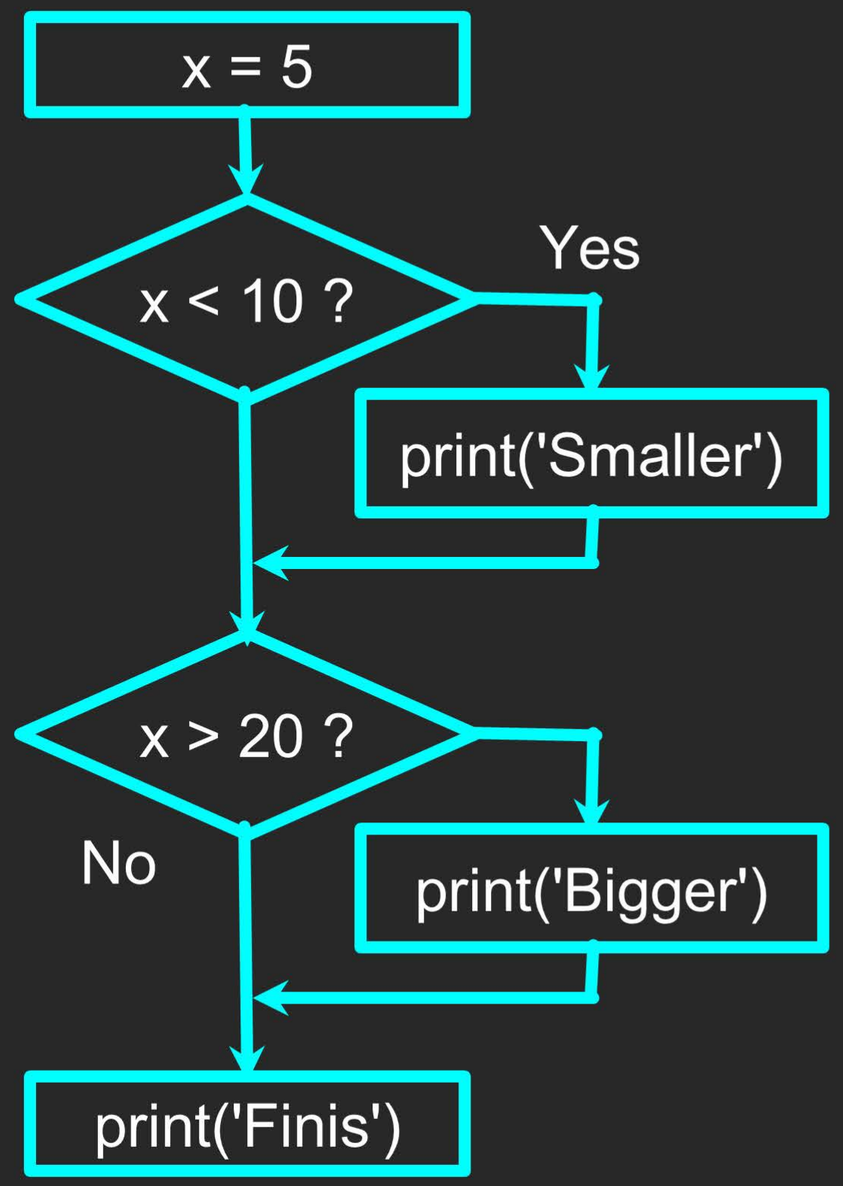
Python for Everybody

www.py4e.com

UMSI

CC BY

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 or Yes / No
- Comparison operators look at variables but do not change the variables

Python	Meaning
<	Less than
<=	Less than or Equal to
==	Equal to
>=	Greater than or Equal to
>	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 Equals 5')
if x < 6 : print('Less than 6')
if x <= 5 :
    print('Less than or Equals 5')
if x != 6 :
    print('Not equal 6')
```

Equals 5

Greater than 4

Greater than or Equals 5

Less than 6

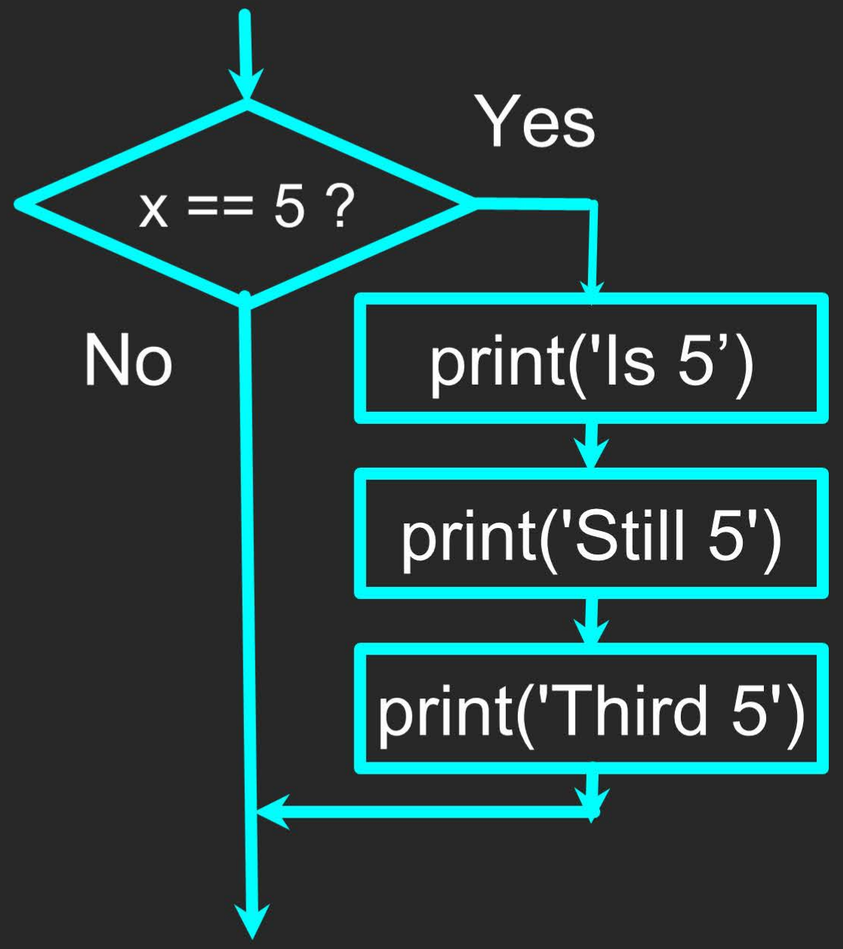
Less than or Equals 5

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
Is 6
Is Still 6
Third 6
Afterwards 6
```



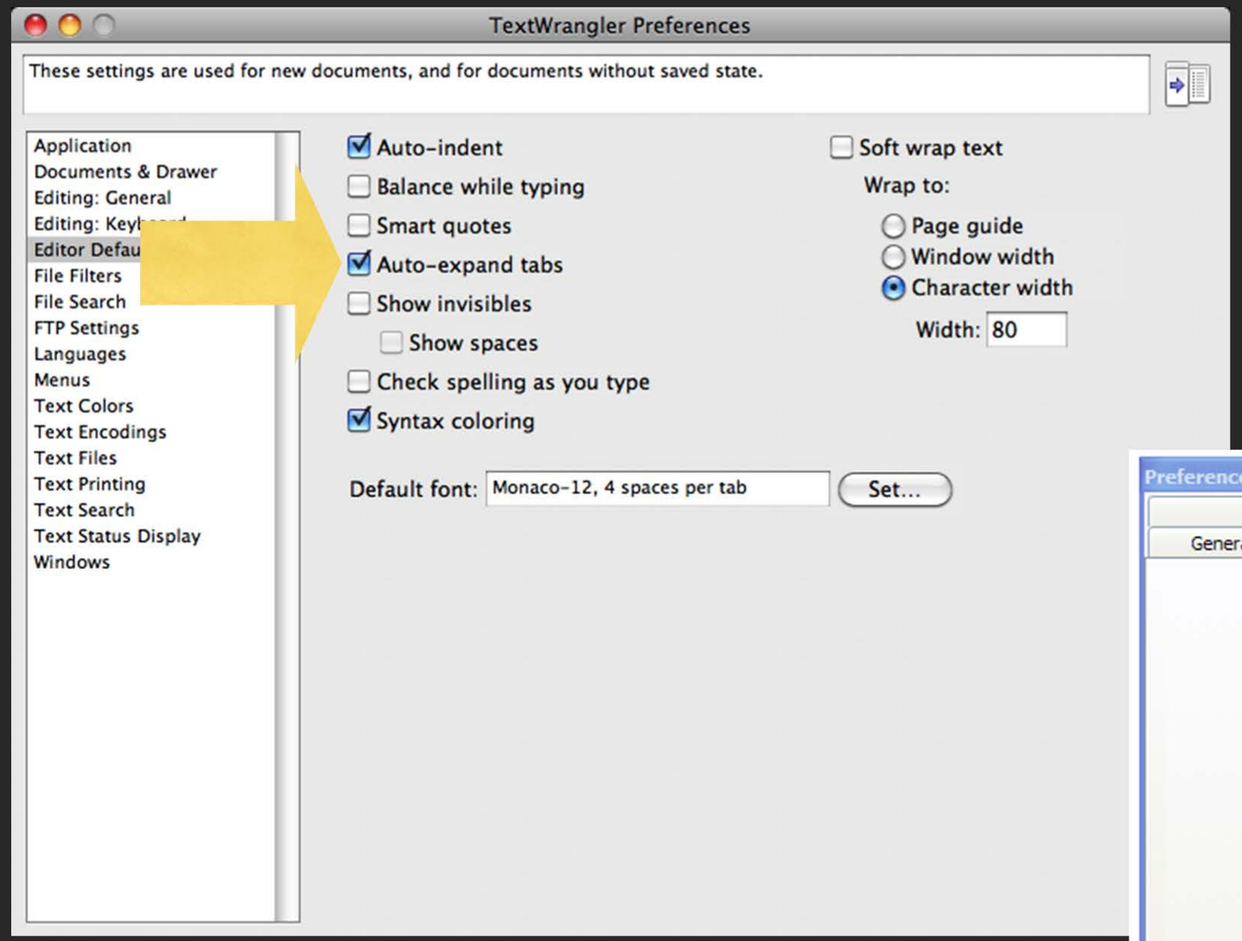
Indentation

- 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

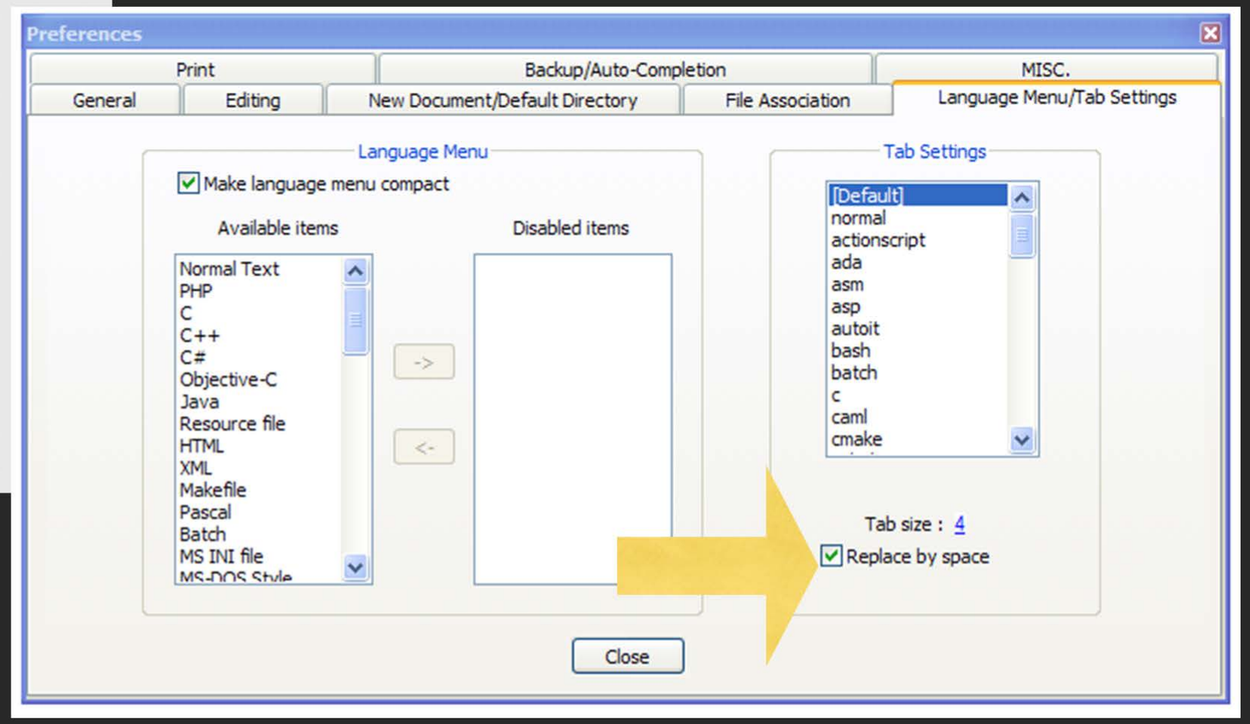
Warning: Turn Off Tabs!!

Atom automatically uses spaces for files with ".py" extension (nice!)

- 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 a line is indented. If you mix tabs and spaces, you may get “indentation errors” even if everything looks fine



This will save you much unnecessary pain.



increase / maintain after if or for

decrease to indicate end of block

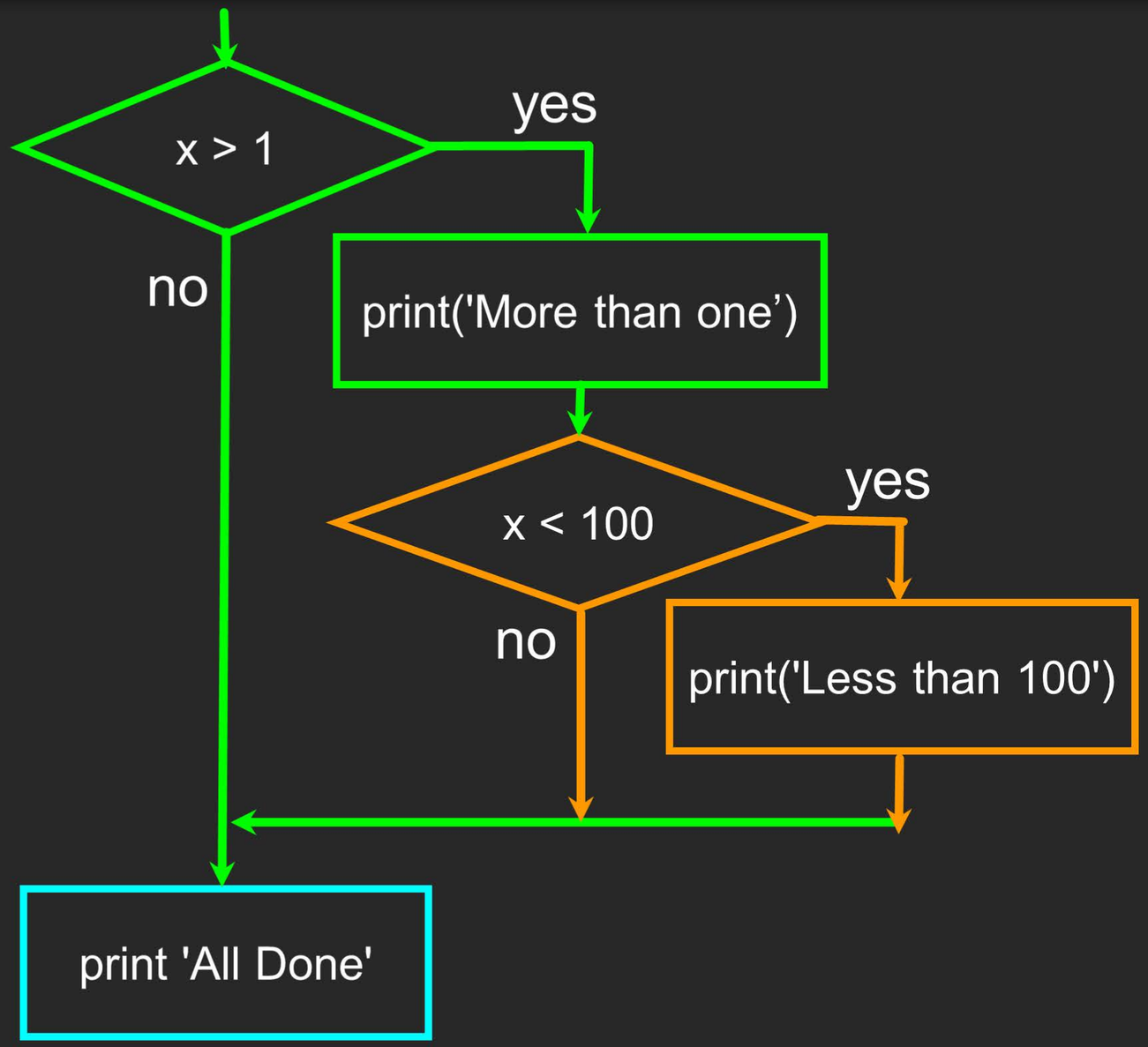
```
x = 5
if x > 2 :
    print('Bigger than 2')
    print('Still bigger')
print('Done with 2')

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

Think about begin/end blocks

```
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')
```

Nested Decisions

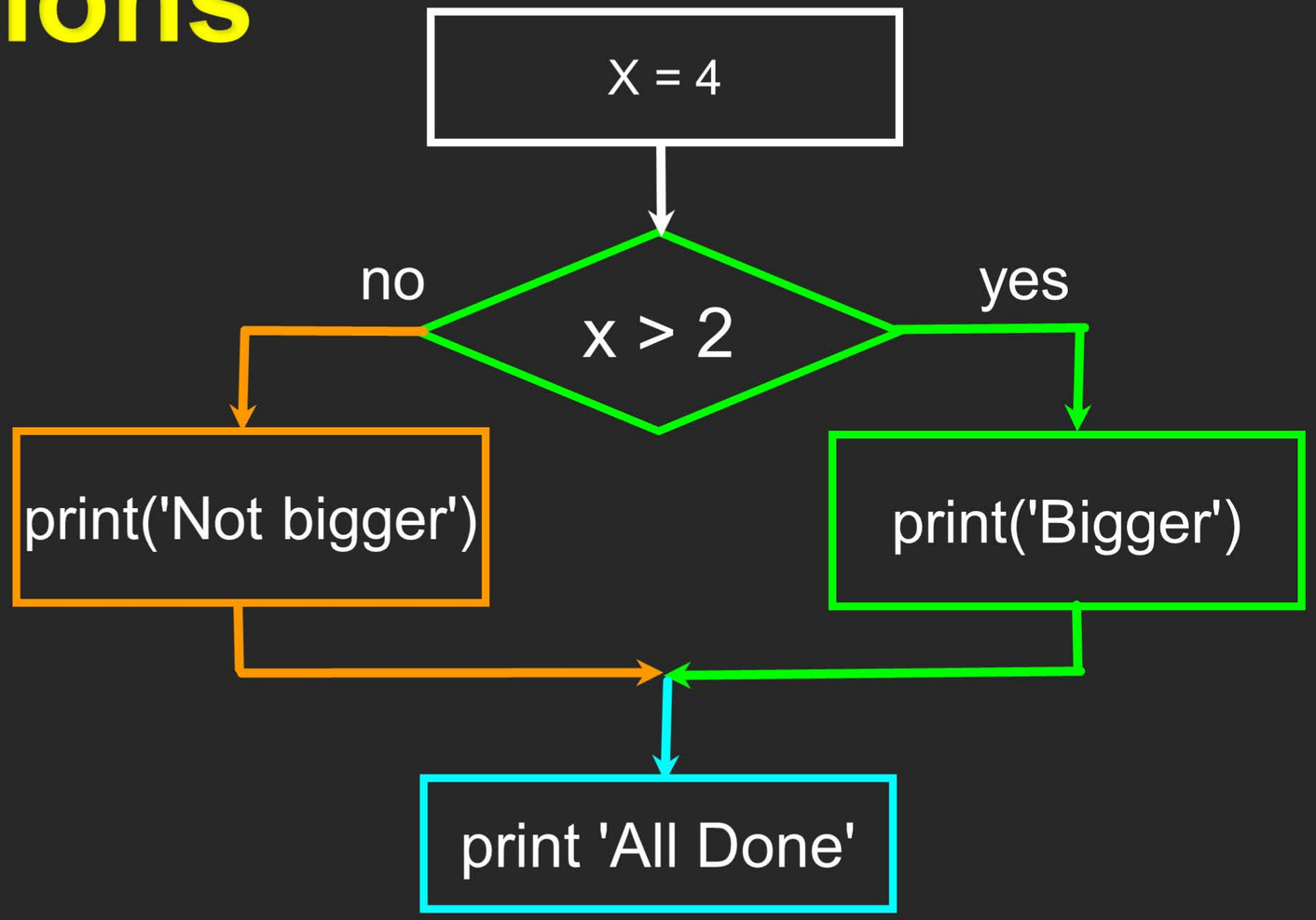
```
x = 42
if x > 1 :
    print('More than one')
    if x < 100 :
        print('Less than 100')
print('All done')
```



Two-way Decisions

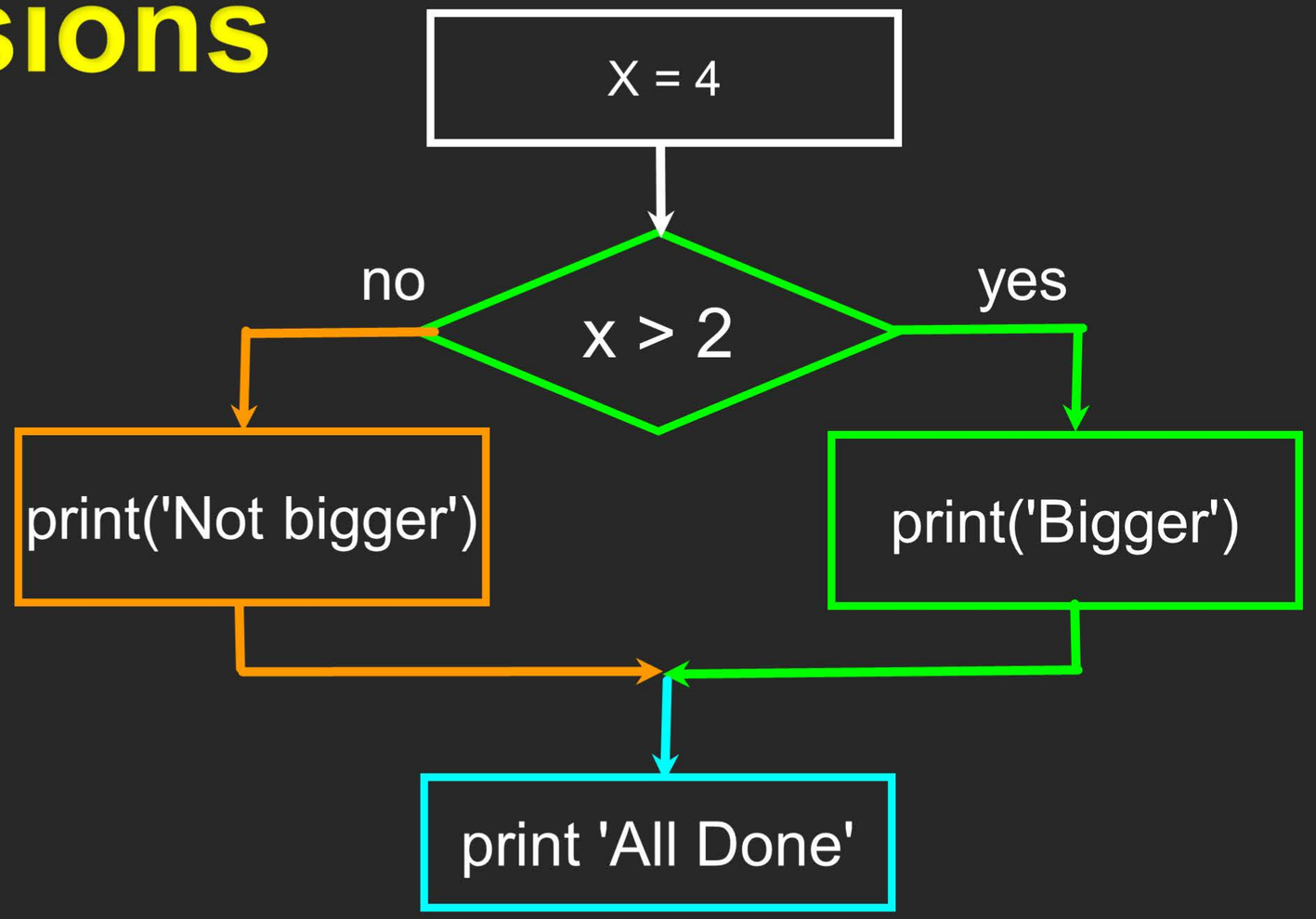
- Sometimes we want to do one thing if a logical expression is true and something else if the expression is false

- It is like a fork in the road - we must choose one or the other path but not both



Two-way Decisions with else:

```
x = 4

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

print 'All done'
```

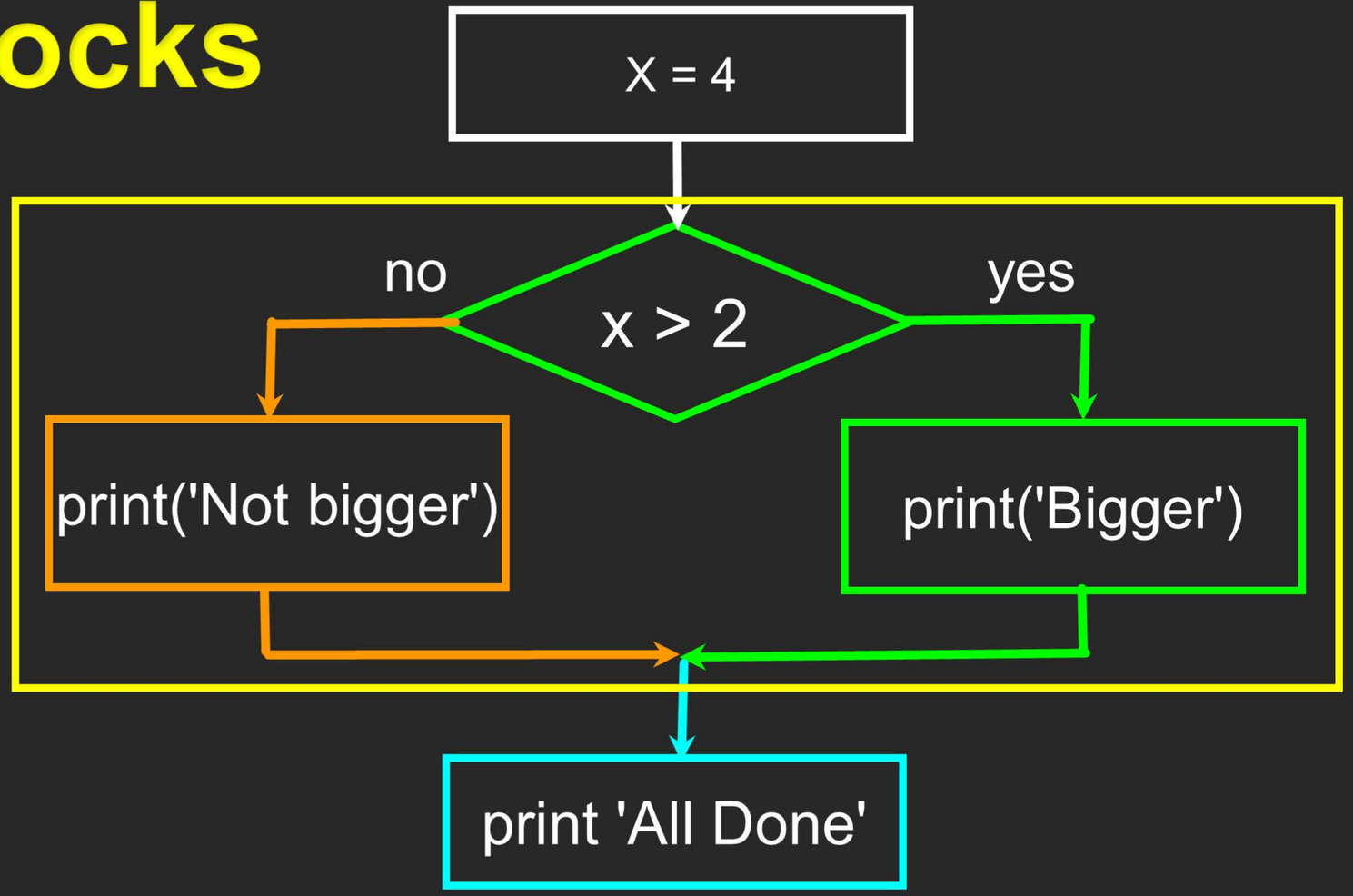


More Conditional Execution Patterns

More Conditional Execution Patterns

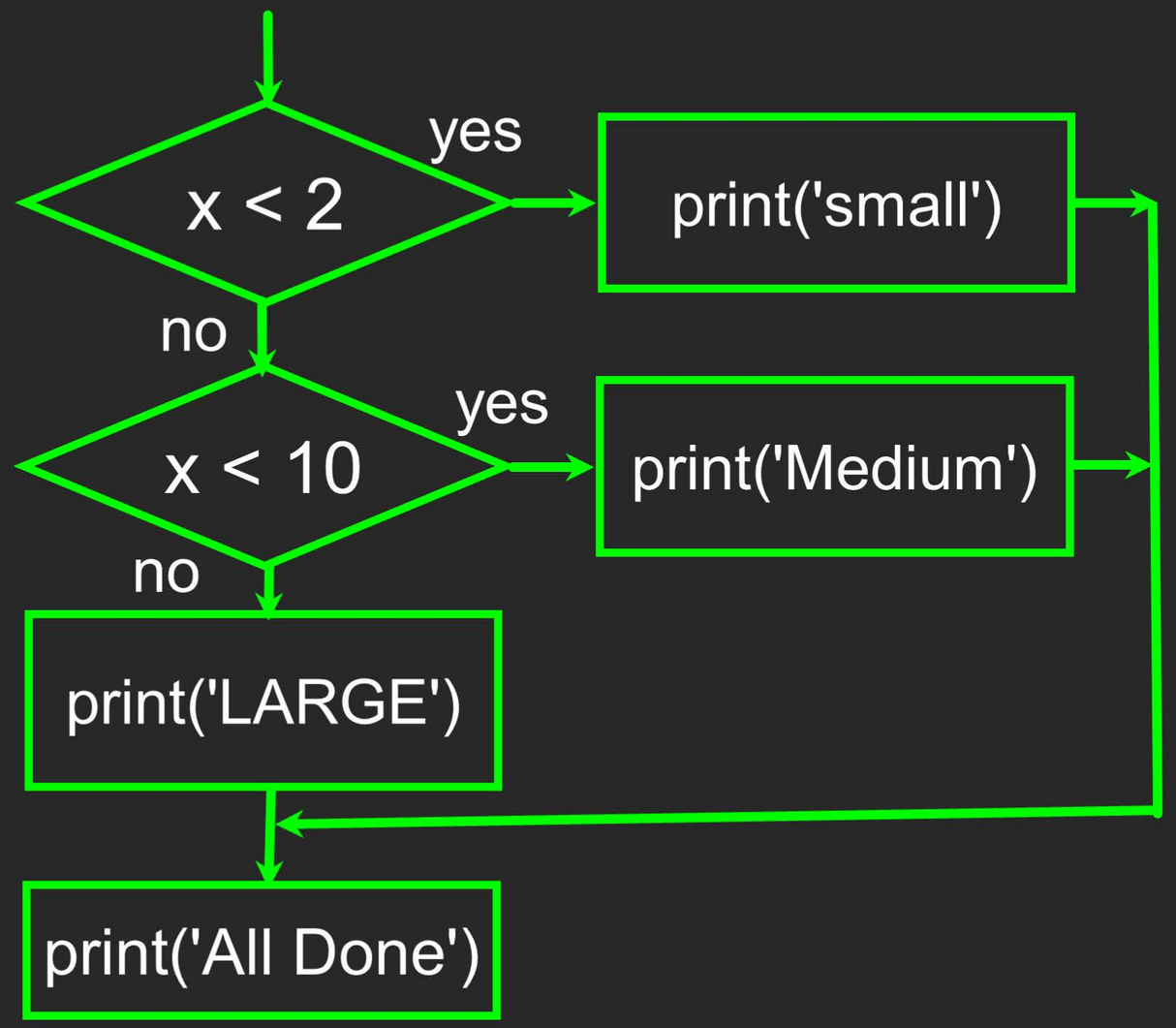
Visualize Blocks

```
x = 4

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

print 'All done'
```



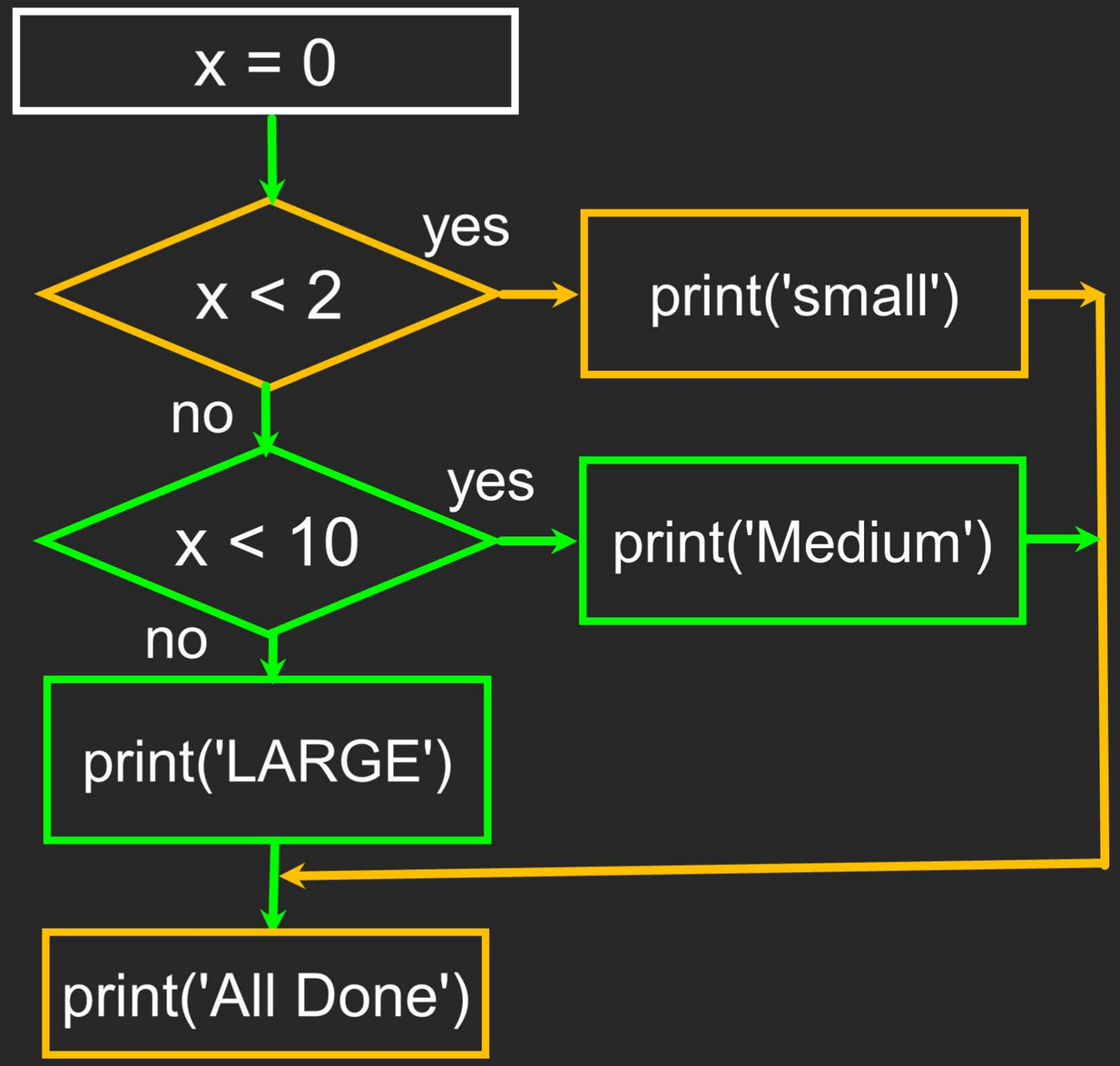
Multi-way

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



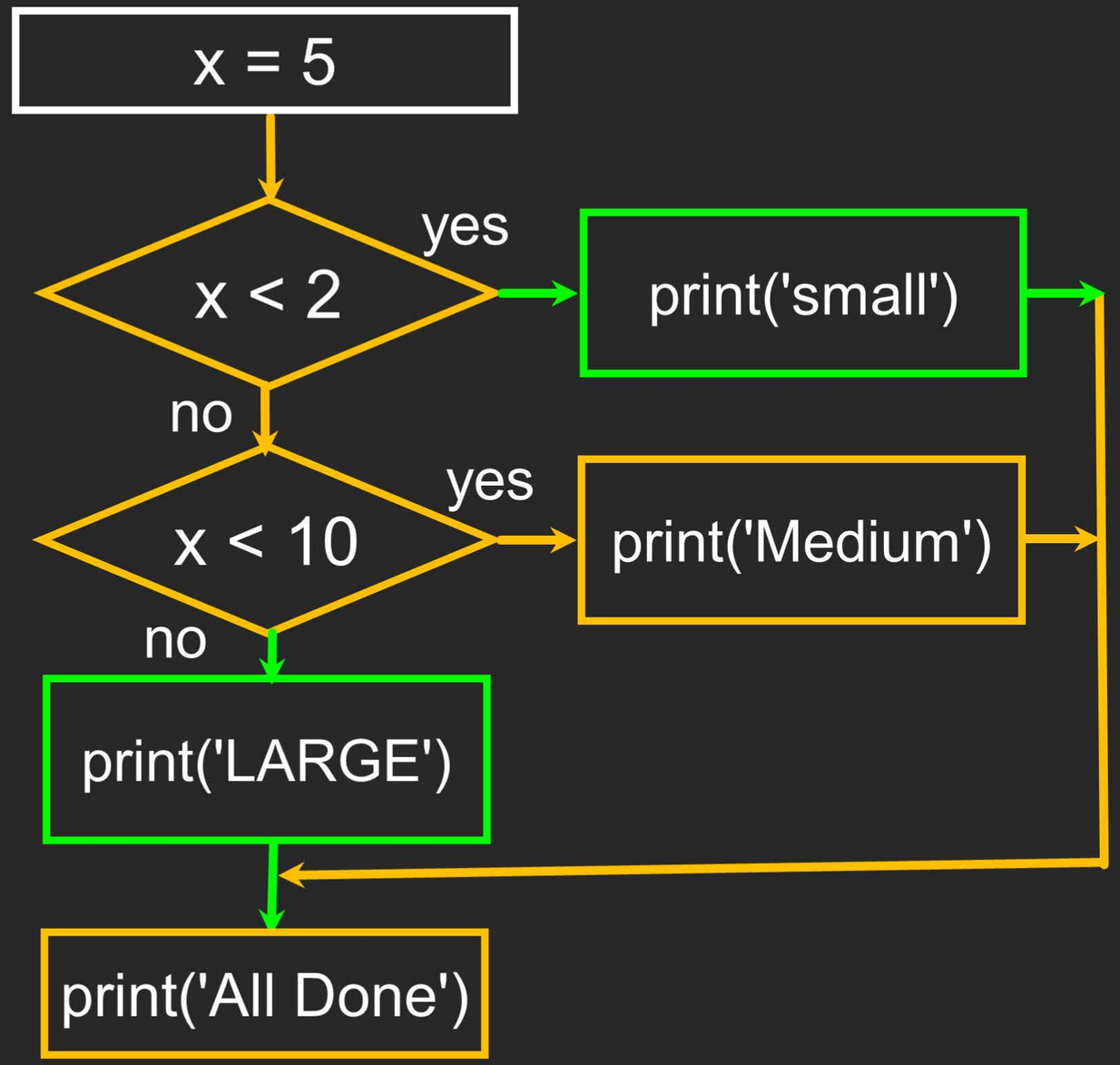
Multi-way

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



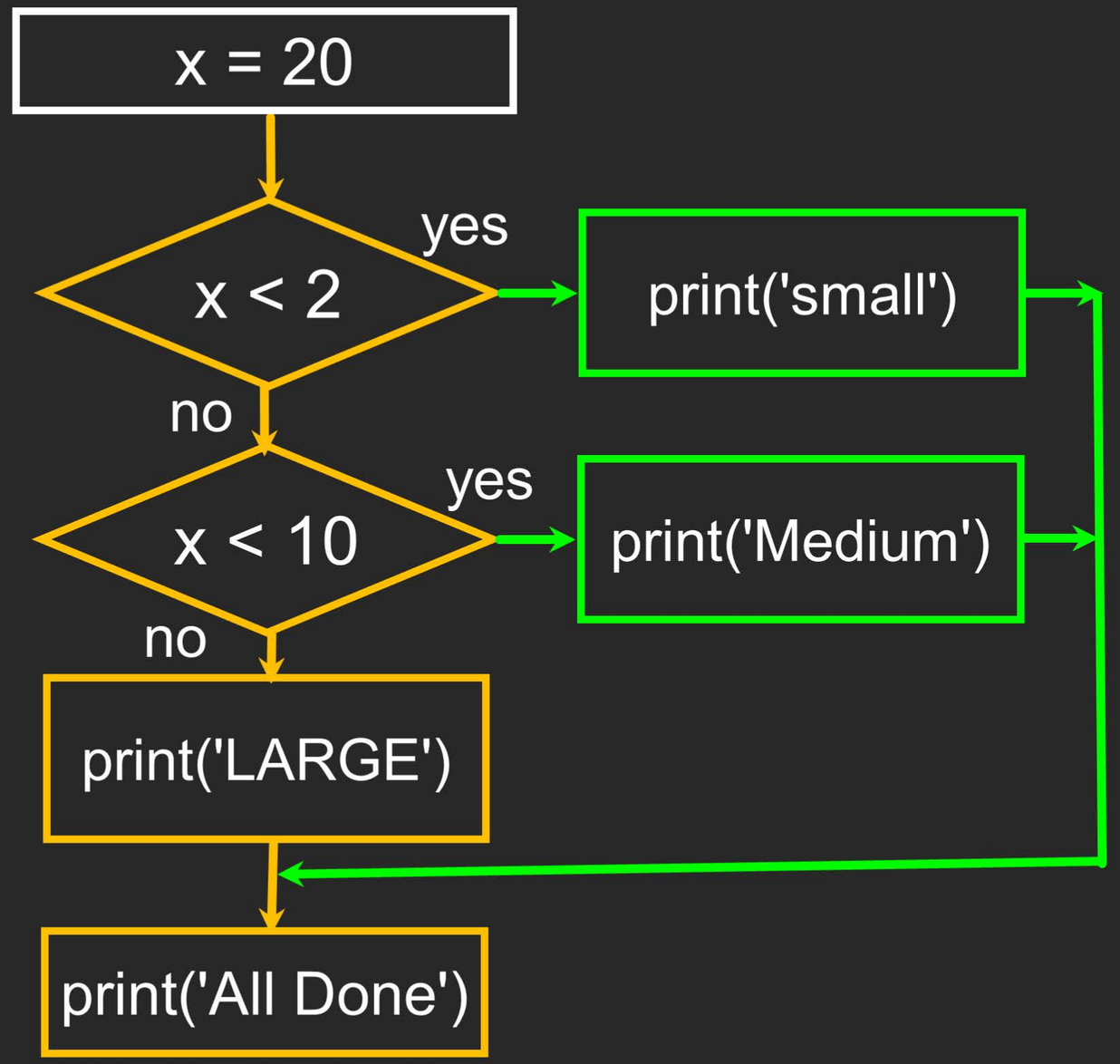
Multi-way

```
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 regardless of the value for x?

```
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 section5

```
$ python3 notry.py
Traceback (most recent call last):
File "notry.py", line 2, in <module>
istr = int(astr)ValueError: invalid literal
for int() with base 10: 'Hello Bob'
```

All Done

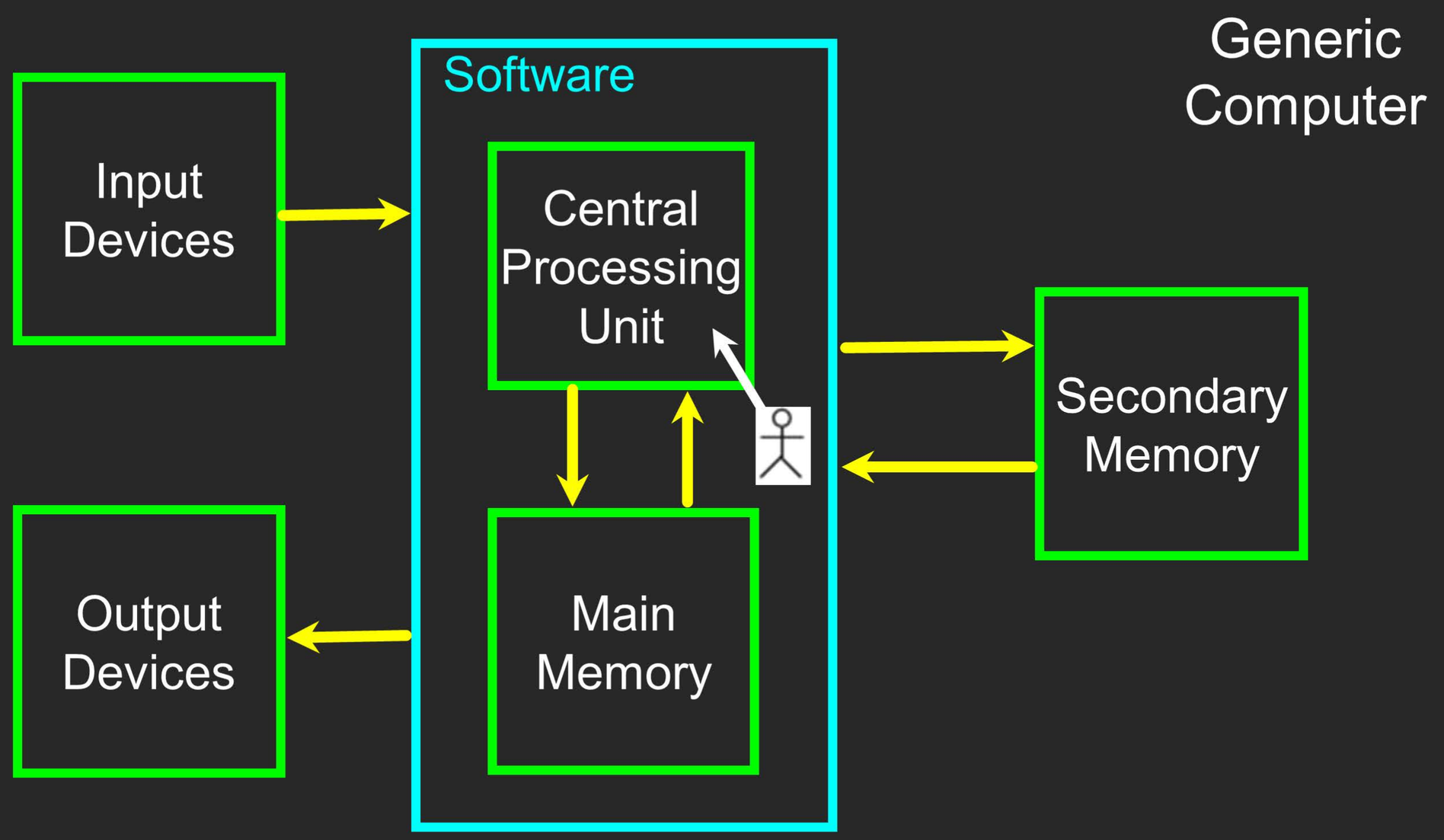
```
$ cat notry.py
astr = 'Hello Bob'
istr = int(astr)
print('First', istr)
astr = '123'
istr = int(astr)
print('Second', istr)
```

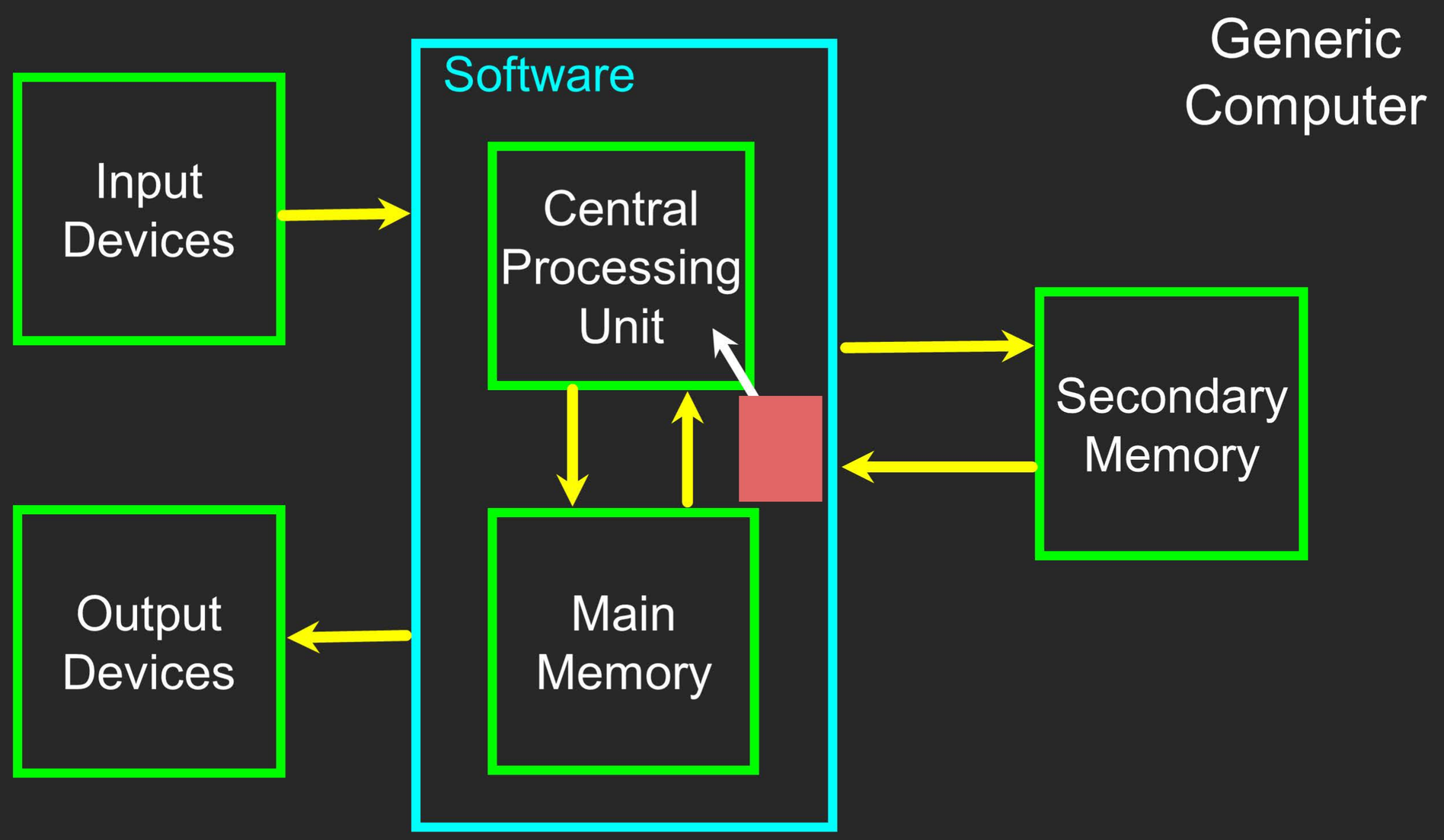
The program stops here

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

```
$ python3 notry.py
Traceback (most recent call last):
File "notry.py", line 2, in <module>
istr = int(astr)ValueError: invalid literal
for int() with base 10: 'Hello Bob'
```

All Done


Generic Computer
Software
Input Devices
Central Processing Unit
Secondary Memory
Output Devices
Main Memory


Generic Computer
Software
Input Devices
Central Processing Unit
Secondary Memory
Output Devices
Main Memory

```
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)
```

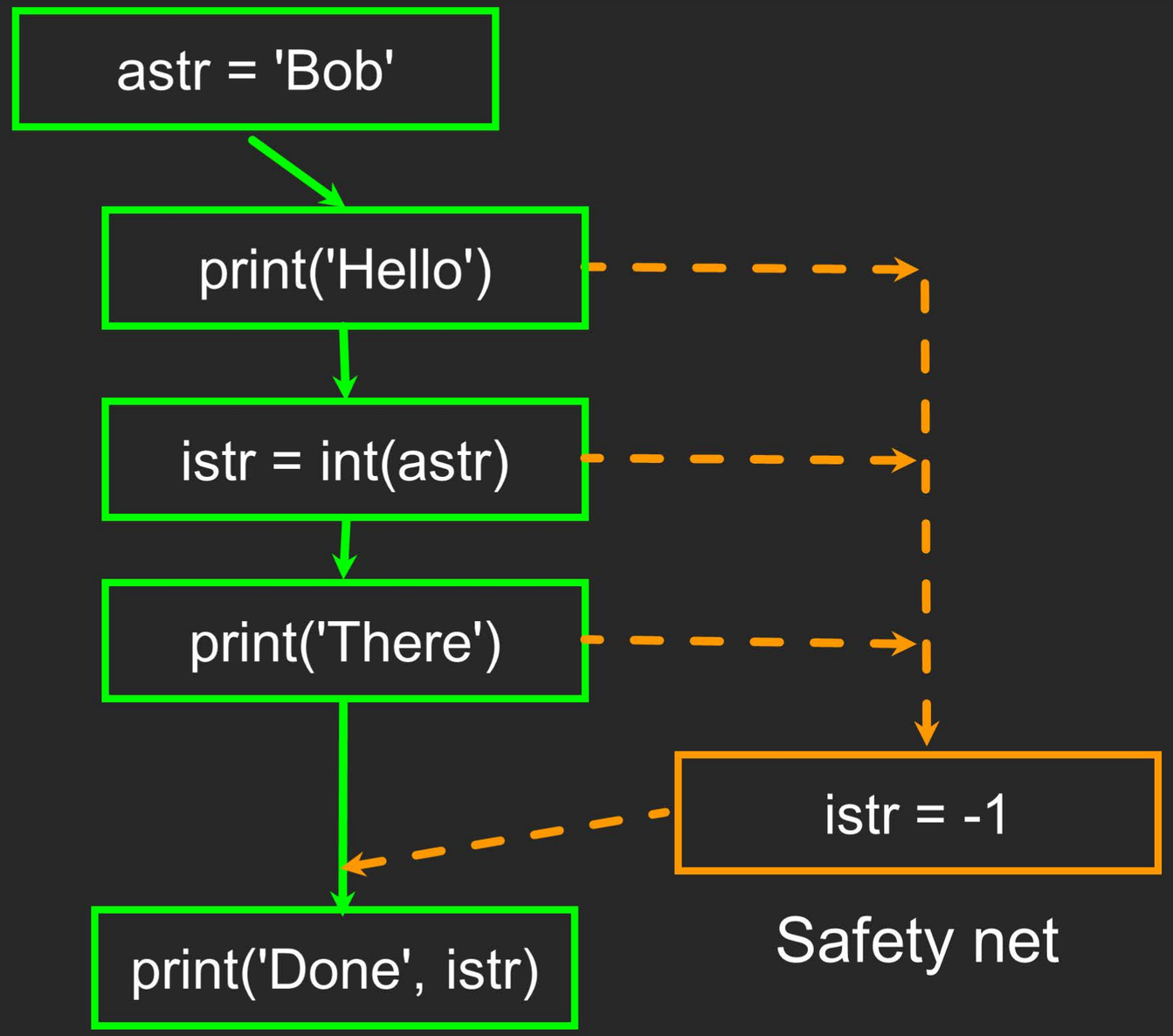
When the first conversion fails - it just drops into the except: clause and the program continues.

```
$ 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 = input('Enter a number:')
try:
    ival = int(rawstr)
except:
    ival = -1

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

```
$ python3 trynum.py
Enter a number:42
Nice work
$ python3 trynum.py
Enter a number:forty-two
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
  == <= >= > < ! =
- Indentation
- One-way Decisions
- Two-way decisions:
  if: and else:
- Nested Decisions
- Multi-way decisions using elif
- try / except to compensate for errors

Acknowledgements / Contributions

These slides are Copyright 2010- Charles R. Severance (www.dr-chuck.com) of the University of Michigan School of Information and made available under a Creative Commons Attribution 4.0 License. Please maintain this last slide in all copies of the document to comply with the attribution requirements of the license. If you make a change, feel free to add your name and organization to the list of contributors on this page as you republish the materials.

Initial Development: Charles Severance, University of Michigan School of Information

… Insert new Contributors and Translators here

...

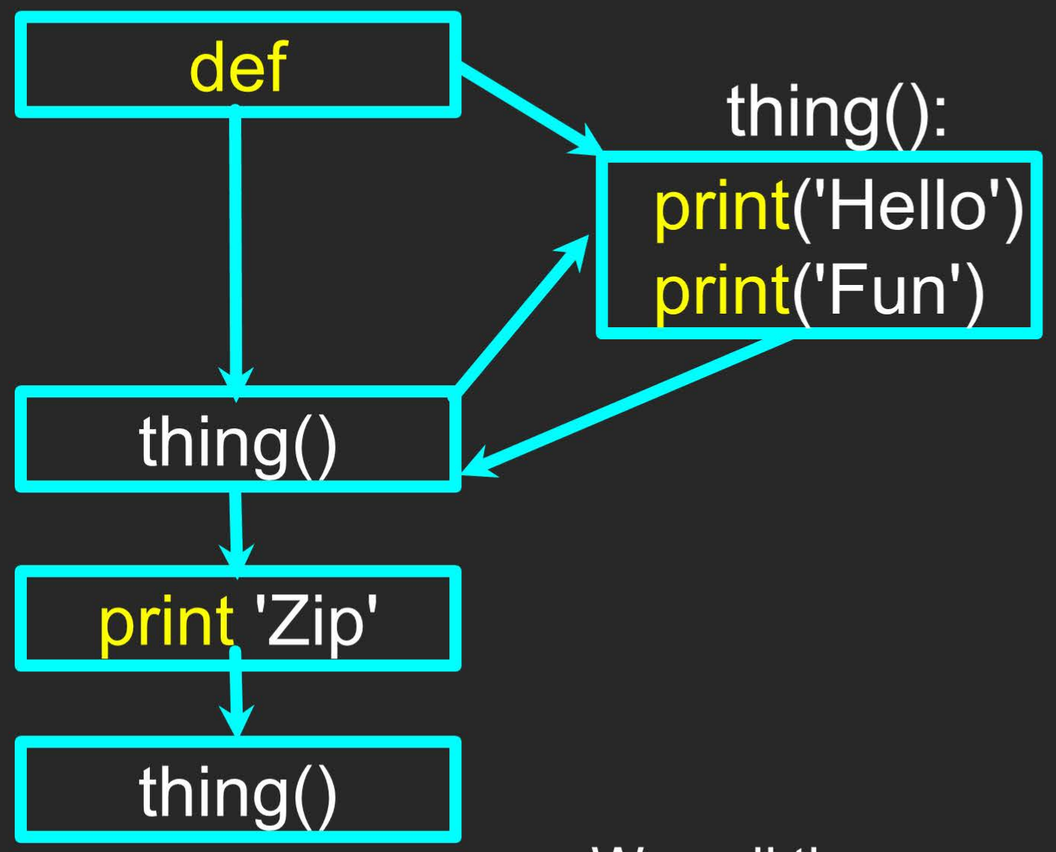
Functions

Chapter 4

Python for Everybody
www.py4e.com

UMSI
CC BY

Stored (and reused) Steps



Program:

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

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

Output:

```
Hello
Fun
Zip
Hello
Fun
```

We call these reusable pieces of code “functions”

Python Functions

- There are two kinds of functions in Python.
  - Built-in functions that are provided as part of Python - print(), 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

Argument

big = max('Hello world')

Assignment

'w'

Result

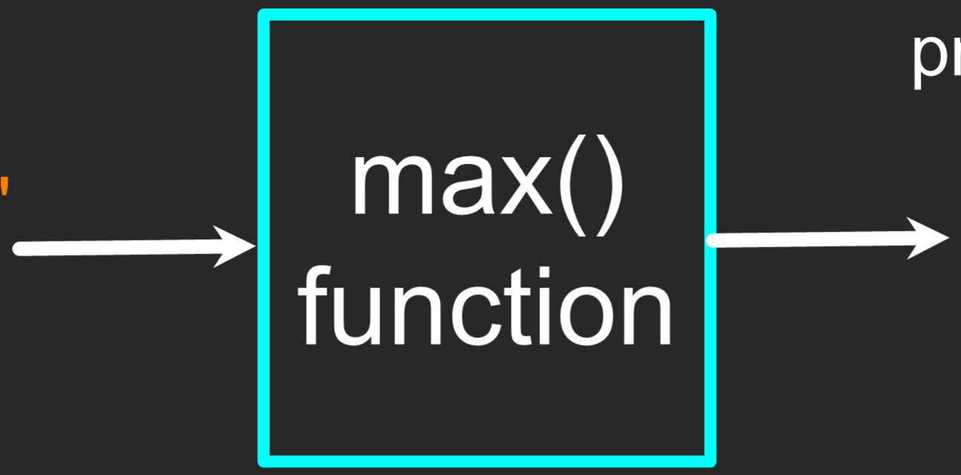
```
>>> big = max('Hello world')
>>> print(big)
w
>>> tiny = min('Hello world')
>>> print(tiny)

>>>
```

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.

'Hello world'
(a string)



'w'
(a string)

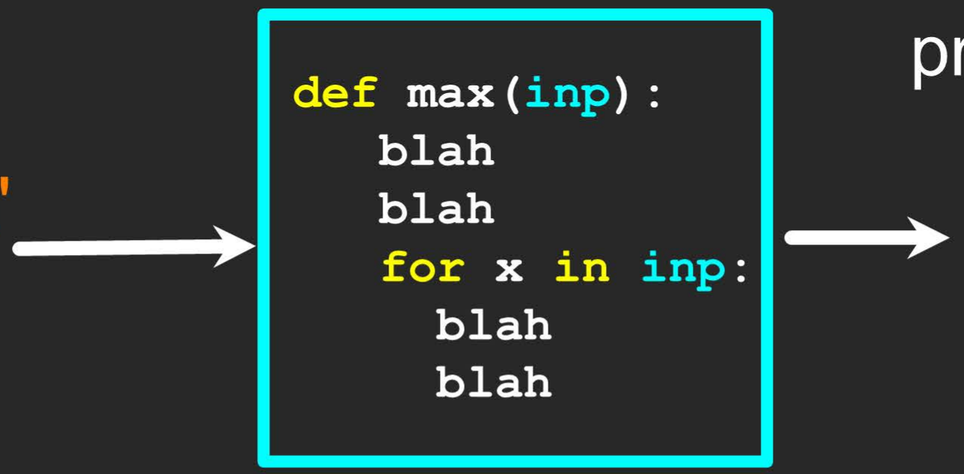
Guido wrote this code

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.

'Hello world'
(a string)



'w'
(a string)

Guido wrote this code

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)
<class 'int'>
>>> f = float(i)
>>> print(f)
42.0
>>> type(f)
<class '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)
<class '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)
<class '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()
```

A Function of Our Own

Acknowledgements / Contributions

These slides are Copyright 2010- Charles R. Severance (www.dr-chuck.com) of the University of Michigan School of Information and open.umich.edu and made available under a Creative Commons Attribution 4.0 License. Please maintain this last slide in all copies of the document to comply with the attribution requirements of the license. If you make a change, feel free to add your name and organization to the list of contributors on this page as you republish the materials.

Initial Development: Charles Severance, University of Michigan School of Information

… Insert new Contributors and Translators here

...

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.')
```

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)
```

```
Hello
Yo
7
```

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)
```

```
Hello
Yo
I'm a lumberjack, and I'm okay.
I sleep all night and I work all day.
7
```

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

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

Argument

Parameters

A parameter is a variable which we use in the function definition. It is a "handle" that allows the code in the function to access the arguments for a particular function invocation.

```
>>> def greet(lang):
...     if lang == 'es':
...         print('Hola')
...     elif lang == 'fr':
...         print('Bonjour')
...     else:
...         print('Hello')
...
>>> greet('en')
Hello
>>> greet('es')
Hola
>>> greet('fr')
Bonjour
>>>
```

Return Values

Often a function will take its arguments, do some computation, and return a value to be used as the value of the function call in the calling expression. The return keyword is used for this.

```
def greet():
    return "Hello"

print(greet(), "Glenn")
print(greet(), "Sally")
```

```
Hello Glenn
Hello Sally
```

Return Value

- A “fruitful” function is one that produces a result (or return value)
- The return statement ends the function execution and “sends back” the result of the function

```
>>> def greet(lang):
...     if lang == 'es':
...         return 'Hola'
...     elif lang == 'fr':
...         return 'Bonjour'
...     else:
...         return 'Hello'
...
>>> print(greet('en'),'Glenn')
Hello Glenn
>>> print(greet('es'),'Sally')
Hola Sally
>>> print(greet('fr'),'Michael')
Bonjour Michael
>>>
```

Arguments, Parameters, and Results

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

Parameter

```
def max(inp):
    blah
    blah
    for x in inp:
      blah
      blah
    return 'w'
```

'Hello world' → 'w'

Argument

Result

Multiple Parameters / Arguments

- We can define more than one parameter in the function definition
- We simply add more arguments when we call the function
- We match the number and order of arguments and parameters

```
def addtwo(a, b):
    added = a + b
    return added

x = addtwo(3, 5)
print(x)
```

```
8
```

Void (non-fruitful) Functions

- When a function does not return a value, we call it a “void” function
- Functions that return values are “fruitful” functions
- Void functions are “not fruitful”

To function or not to function...

- Organize your code into “paragraphs” - capture a complete thought and “name it”
- Don’t repeat yourself - make it work once and then reuse it
- If something gets too long or complex, break it up into logical chunks and put those chunks in functions
- Make a library of common stuff that you do over and over - perhaps share this with your friends...

Summary

- Arguments
- Results (fruitful functions)
- Void (non-fruitful) functions
- Why use functions?

- Functions
- Built-In Functions
  - Type conversion (int, float)
  - String conversions
- Parameters

Exercise

Rewrite your pay computation with time-and-a-half for overtime and create a function called computepay which takes two parameters (hours and rate).

Enter Hours: 45
Enter Rate: 10

Pay: 475.0

475 = 40 * 10 + 5 * 15

Acknowledgements / Contributions

These slides are Copyright 2010- Charles R. Severance (www.dr-chuck.com) of the University of Michigan School of Information and open.umich.edu and made available under a Creative Commons Attribution 4.0 License. Please maintain this last slide in all copies of the document to comply with the attribution requirements of the license. If you make a change, feel free to add your name and organization to the list of contributors on this page as you republish the materials.

Initial Development: Charles Severance, University of Michigan School of Information

… Insert new Contributors and Translators here

...

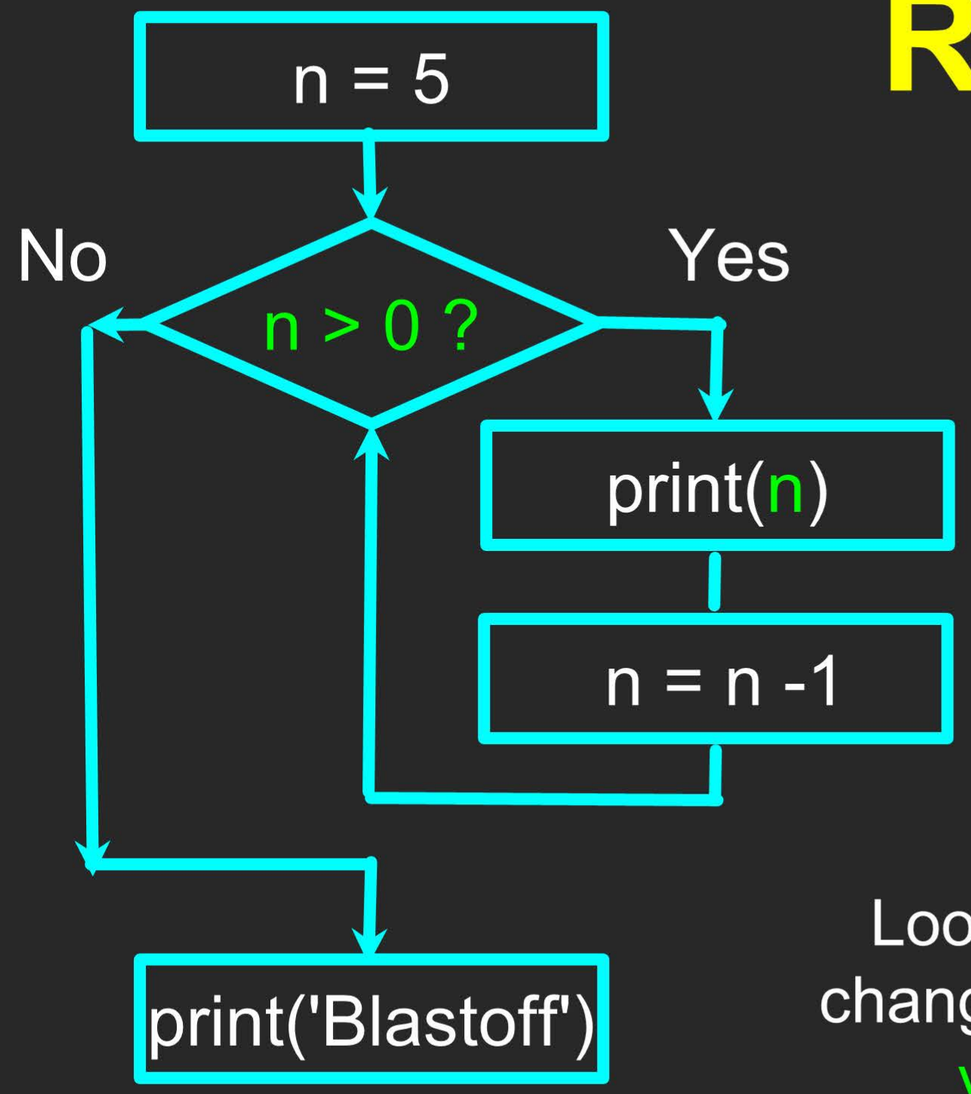
Loops and Iteration

Chapter 5

Python for Everybody
www.py4e.com

UMSI
CC BY

Repeated Steps



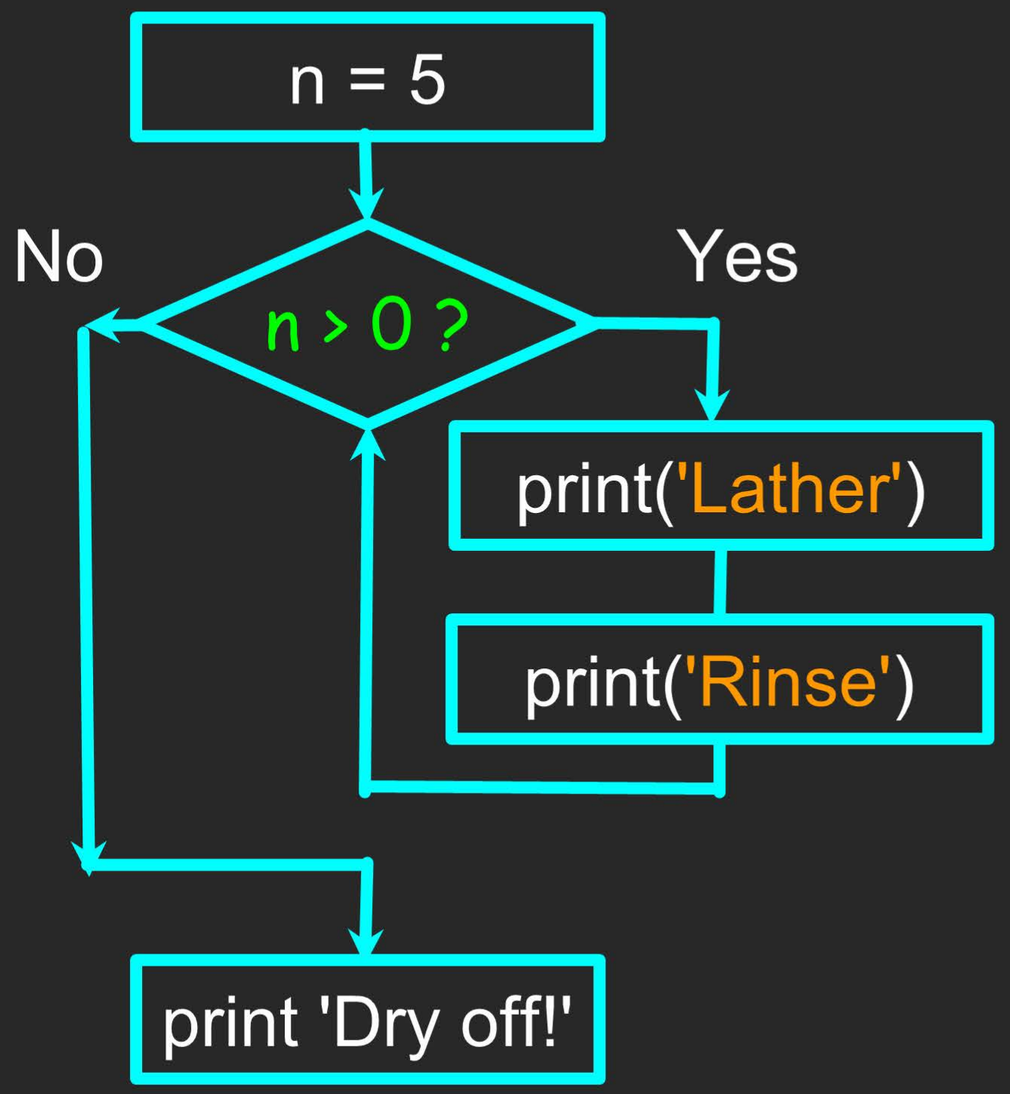
Program:

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

Output:

```
5
4
3
2
1
Blastoff!
0
```

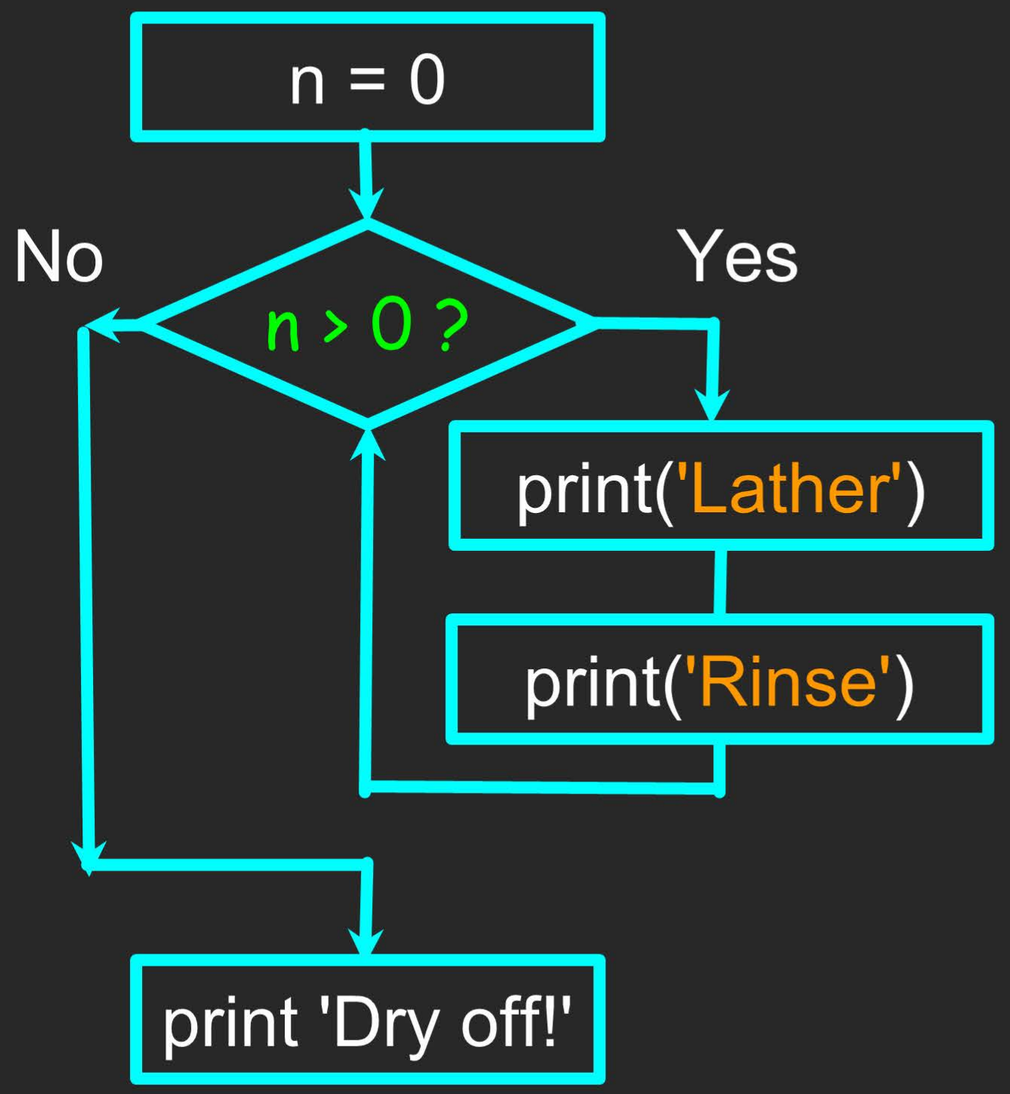
Loops (repeated steps) have iteration variables that change each time through a loop. Often these iteration variables go through a sequence of numbers.



An Infinite Loop

```
n = 5
while n > 0 :
    print('Lather')
    print('Rinse')
print('Dry off!')
```

What is wrong with this loop?



Another Loop

```
n = 0
while n > 0 :
    print('Lather')
    print('Rinse')
print('Dry off!')
```

What is this loop doing?

Breaking Out of a Loop

- The break statement ends the current loop and jumps to the statement immediately following the loop
- It is like a loop test that can happen anywhere in the body of the loop

```
while True:
    line = input('> ')
    if line == 'done' :
        break
    print(line)
print('Done!')
```

```
> hello there
hello there
> finished
finished
> done
Done!
```

Breaking Out of a Loop

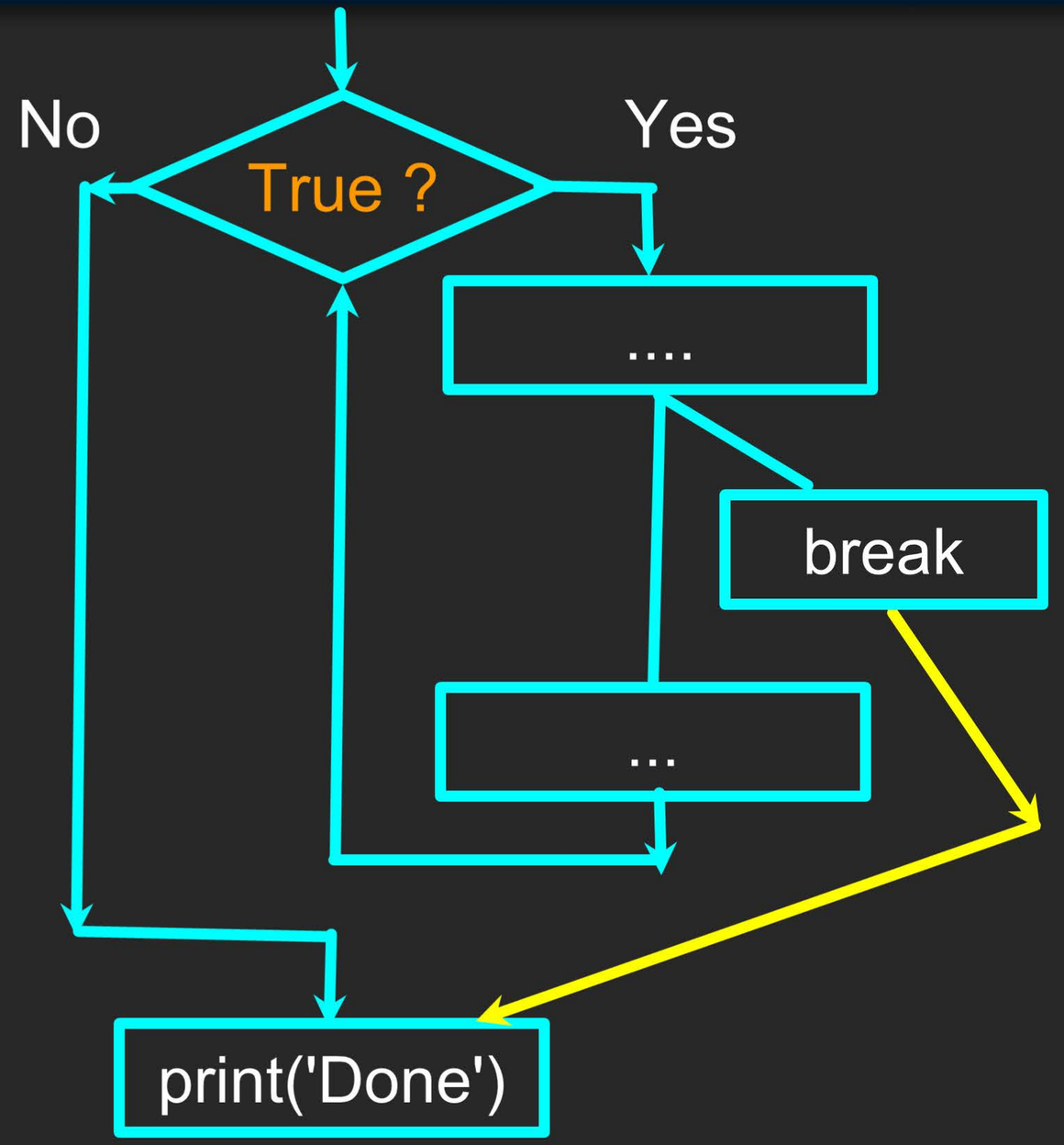
- The break statement ends the current loop and jumps to the statement immediately following the loop
- It is like a loop test that can happen anywhere in the body of the loop

```
while True:
    line = input('> ')
    if line == 'done' :
        break
    print(line)
print('Done!')
```

```
> hello there
hello there
> finished
finished
> done
Done!
```

```
while True:
    line = input('> ')
    if line == 'done' :
        break
    print(line)
print('Done!')
```

http://en.wikipedia.org/wiki/Transporter_(Star_Trek)



Finishing an Iteration with Continue

The continue statement ends the current iteration and jumps to the top of the loop and starts the next iteration

```
while True:
    line = input('> ')
    if line[0] == '#' :
        continue
    if line == 'done' :
        break
    print(line)
print('Done!')
```

```
> hello there
hello there
> # don't print this
> print this!
print this!
> done
Done!
```

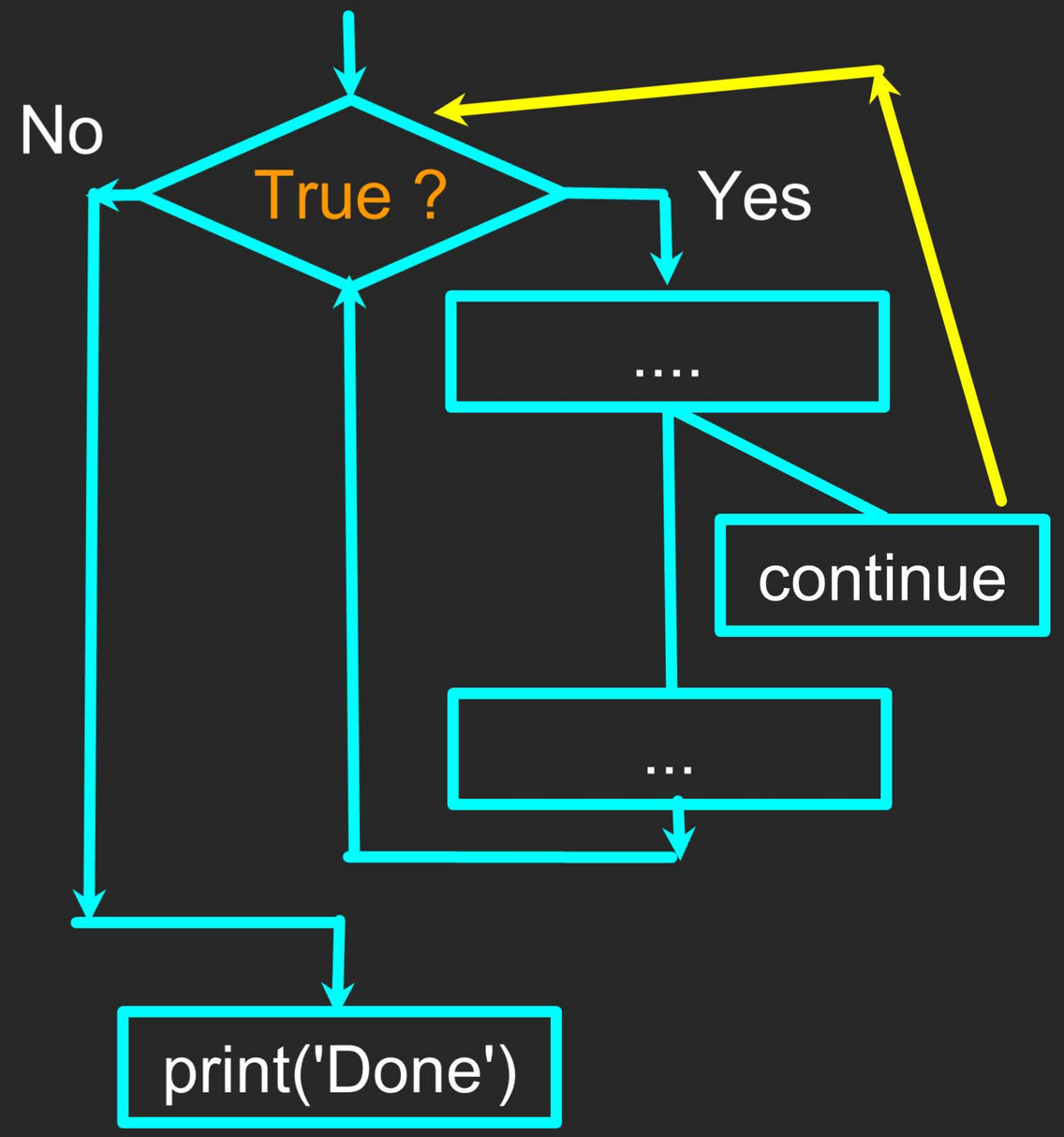
Finishing an Iteration with Continue

The continue statement ends the current iteration and jumps to the top of the loop and starts the next iteration

```
while True:
    line = input('> ')
    if line[0] == '#' :
        continue
    if line == 'done' :
        break
    print(line)
print('Done!')
```

```
> hello there
hello there
> # don't print this
> print this!
print this!
> done
Done!
```

```
while True:
    line = raw_input('> ')
    if line[0] == '#' :
        continue
    if line == 'done' :
        break
    print(line)
print('Done!')
```



Indefinite Loops

- While loops are called "indefinite loops" because they keep going until a logical condition becomes False
- The loops we have seen so far are pretty easy to examine to see if they will terminate or if they will be "infinite loops"
- Sometimes it is a little harder to be sure if a loop will terminate

Definite Loops

Acknowledgements / Contributions

These slides are Copyright 2010- Charles R. Severance (www.dr-chuck.com) of the University of Michigan School of Information and open.umich.edu and made available under a Creative Commons Attribution 4.0 License. Please maintain this last slide in all copies of the document to comply with the attribution requirements of the license. If you make a change, feel free to add your name and organization to the list of contributors on this page as you republish the materials.

Initial Development: Charles Severance, University of Michigan School of Information

… Insert new Contributors and Translators here

...

Definite Loops

- Quite often we have a list of items of the lines in a file - effectively a finite set of things
- We can write a loop to run the loop once for each of the items in a set using the Python for construct
- These loops are called “definite loops” because they execute an exact number of times
- We say that “definite loops iterate through the members of a set”

A Simple Definite Loop

```
for i in [5, 4, 3, 2, 1] :
    print(i)
print('Blastoff!')
```

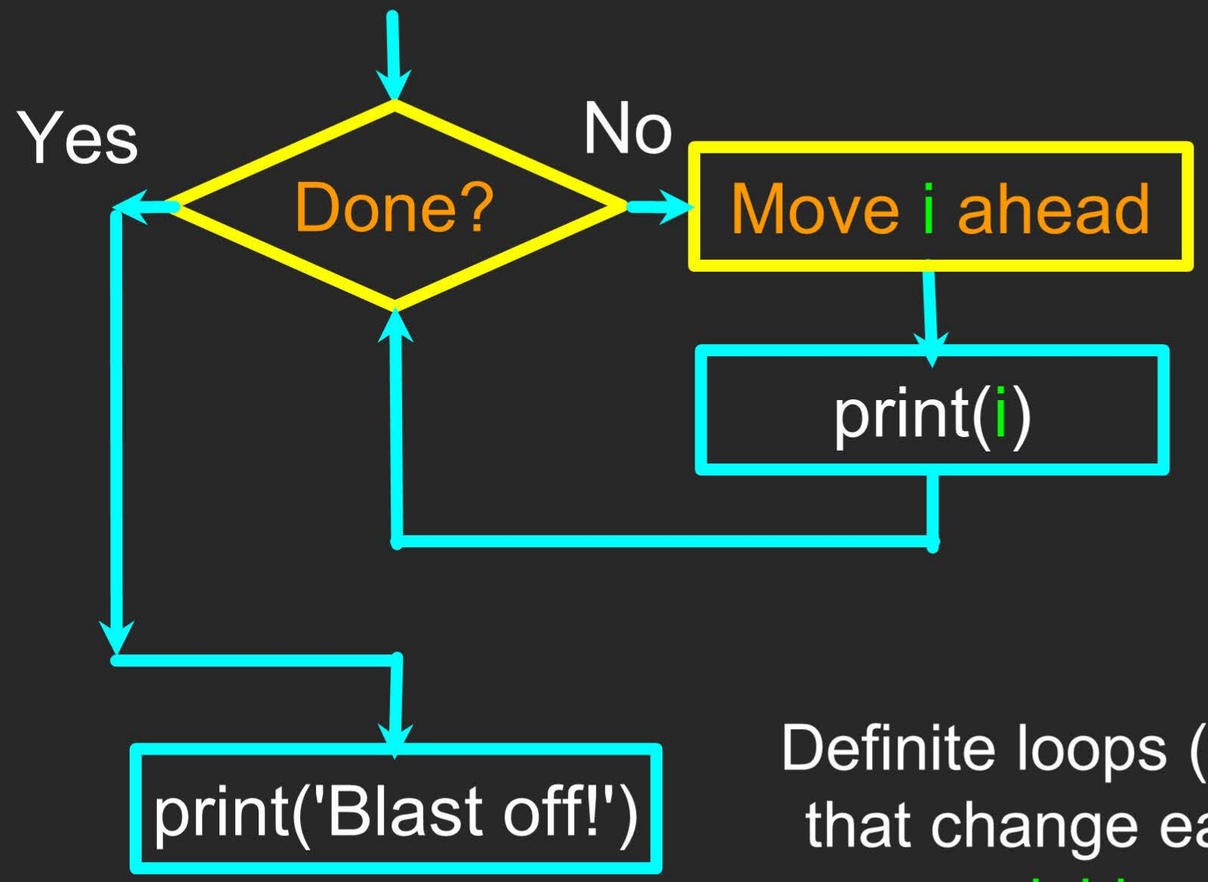
```
5
4
3
2
1
Blastoff!
```

A Definite Loop with Strings

```
friends = ['Joseph', 'Glenn', 'Sally']
for friend in friends :
    print('Happy New Year:', friend)
print('Done!')
```

```
Happy New Year: Joseph
Happy New Year: Glenn
Happy New Year: Sally

Done!
```

A Simple Definite Loop




```
for i in [5, 4, 3, 2, 1] :
    print(i)
print('Blastoff!')
```

```
5
4
3
2
1
Blastoff!
```

Definite loops (for loops) have explicit iteration variables that change each time through a loop. These iteration variables move through the sequence or set.

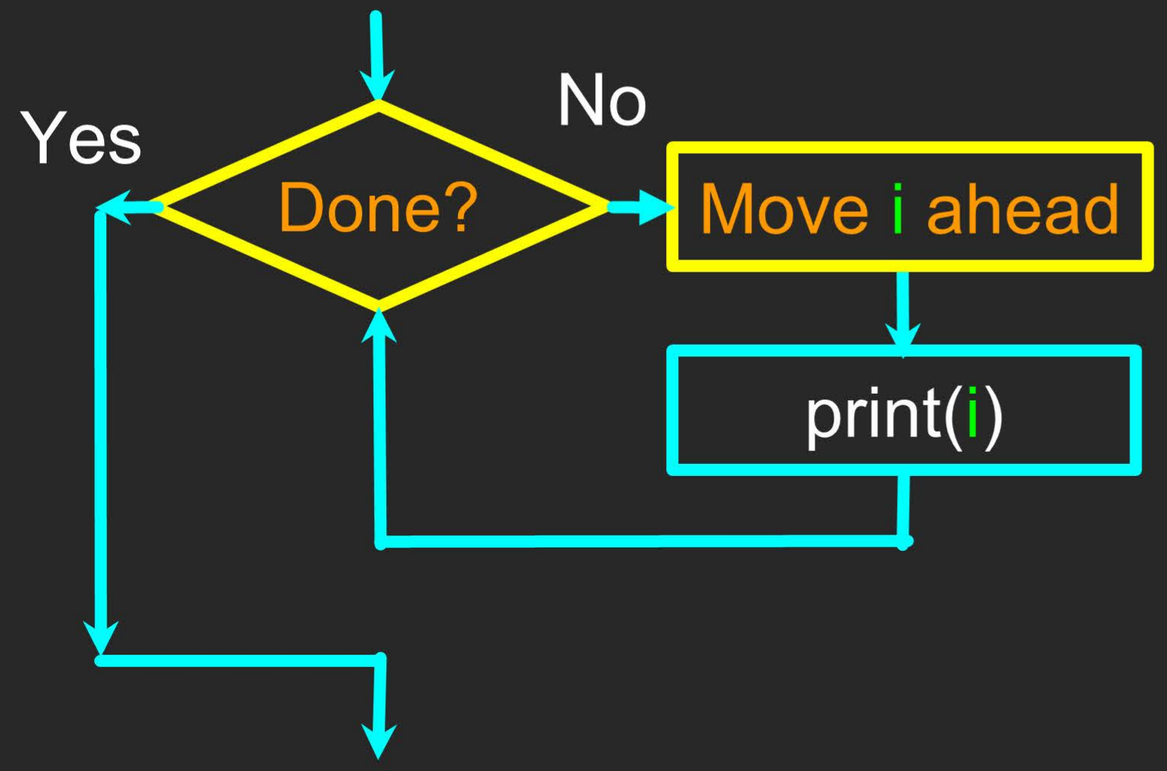
Looking at In...

- The iteration variable "iterates" through the sequence (ordered set)
- The block (body) of code is executed once for each value in the sequence
- The iteration variable moves through all of the values in the sequence

Iteration variable

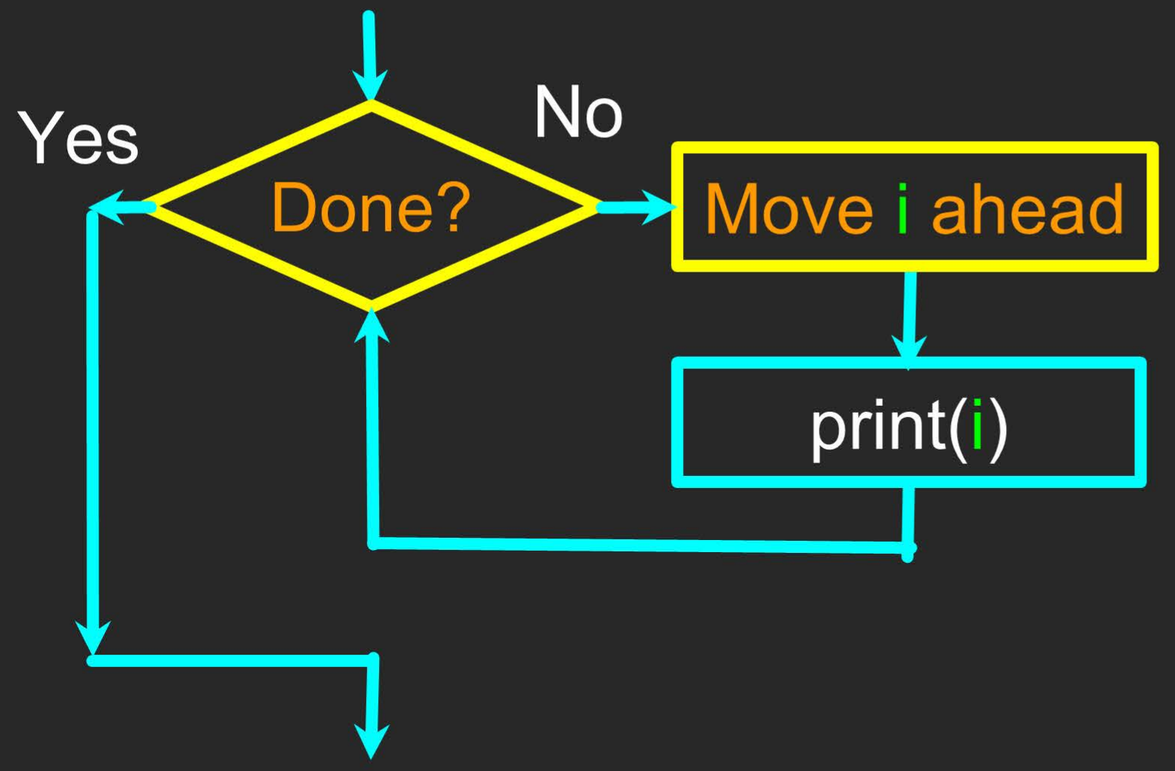
Five-element sequence

```
for i in [5, 4, 3, 2, 1] :
    print(i)
```

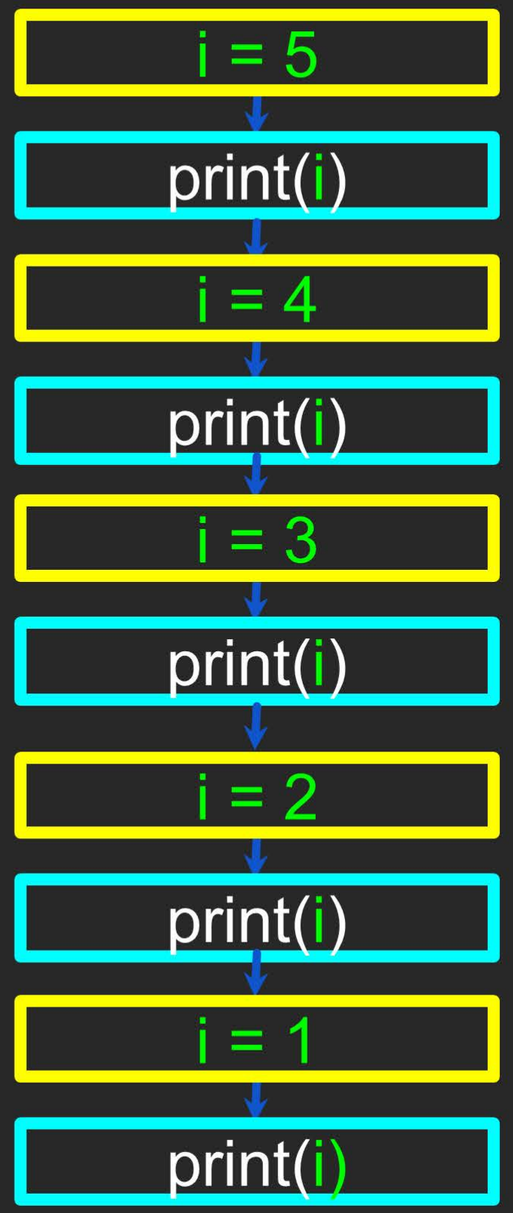



```
for i in [5, 4, 3, 2, 1] :
    print(i)
```

- The iteration variable "iterates" through the sequence (ordered set)
- The block (body) of code is executed once for each value in the sequence
- The iteration variable moves through all of the values in the sequence




```
for i in [5, 4, 3, 2, 1] :
    print(i)
```



Definite Loops

- Quite often we have a list of items of the lines in a file - effectively a finite set of things
- We can write a loop to run the loop once for each of the items in a set using the Python for construct
- These loops are called "definite loops" because they execute an exact number of times
- We say that "definite loops iterate through the members of a set"

Loop Idioms

Acknowledgements / Contributions

These slides are Copyright 2010- Charles R. Severance (www.dr-chuck.com) of the University of Michigan School of Information and open.umich.edu and made available under a Creative Commons Attribution 4.0 License. Please maintain this last slide in all copies of the document to comply with the attribution requirements of the license. If you make a change, feel free to add your name and organization to the list of contributors on this page as you republish the materials.

Initial Development: Charles Severance, University of Michigan School of Information

… Insert new Contributors and Translators here

...

Loop Idioms: What We Do in Loops

Note: Even though these examples are simple, the patterns apply to all kinds of loops

Making “smart” Loops

The trick is “knowing” something about the whole loop when you are stuck writing code that only sees one entry at a time

Set some variables to initial values

for thing in data:

Look for something or do something to each entry separately, updating a variable

Look at the variables

Looping Through a Set

```
print('Before')
for thing in [9, 41, 12, 3, 74, 15] :
     print(thing)
print('After')
```

```
$ python basicloop.py
Before
9
41
12
3
74
15
After
```

What is the Largest Number?

What is the Largest Number?

3

What is the Largest Number?

41

What is the Largest Number?

12

What is the Largest Number?

9

What is the Largest Number?

74

What is the Largest Number?

15

What is the Largest Number?

What is the Largest Number?

3 41 12 9 74 15

What is the Largest Number?

largest_so_far -1

What is the Largest Number?

3

largest_so_far 3

What is the Largest Number?

41

largest_so_far 41

What is the Largest Number?

12

largest_so_far 41

What is the Largest Number?

9

largest_so_far 41

What is the Largest Number?

74

largest_so_far 74

What is the Largest Number?

15

74

What is the Largest Number?

3 41 12 9 74 15

74

Finding the Largest Value

```
largest_so_far = -1
print('Before', largest_so_far)
for the_num in [9, 41, 12, 3, 74, 15] :
    if the_num > largest_so_far :
        largest_so_far = the_num
    print(largest_so_far, the_num)

print('After', largest_so_far)
```

```
$ python largest.py
Before -1
9  9
41  41
41 12
41  3
74  74
74  15
After 74
```

We make a variable that contains the largest value we have seen so far. If the current number we are looking at is larger, it is the new largest value we have seen so far.

More Loop Idioms

Acknowledgements / Contributions

These slides are Copyright 2010- Charles R. Severance (www.dr-chuck.com) of the University of Michigan School of Information and open.umich.edu and made available under a Creative Commons Attribution 4.0 License. Please maintain this last slide in all copies of the document to comply with the attribution requirements of the license. If you make a change, feel free to add your name and organization to the list of contributors on this page as you republish the materials.

Initial Development: Charles Severance, University of Michigan School of Information

… Insert new Contributors and Translators here

...

Counting in a Loop

```
zork = 0
print('Before', zork)
for thing in [9, 41, 12, 3, 74, 15] :
    zork = zork + 1
    print(zork, thing)
print('After', zork)
```

```
$ python countloop.py
Before 0
1 9
2 41
3 12
4 3
5 74
6 15
After 6
```

To count how many times we execute a loop, we introduce a counter variable that starts at 0 and we add one to it each time through the loop.

Summing in a Loop

```
zork = 0
print('Before', zork)
for thing in [9, 41, 12, 3, 74, 15] :
    zork = zork + thing
    print(zork, thing)
print('After', zork)
```

```
$ python countloop.py
Before 0
9 9
50 41
62 12
65 3
139 74
154 15
After 154
```

To add up a value we encounter in a loop, we introduce a sum variable that starts at 0 and we add the value to the sum each time through the loop.

Finding the Average in a Loop

```
count = 0
sum = 0
print('Before', count, sum)
for value in [9, 41, 12, 3, 74, 15] :
    count = count + 1
    sum = sum + value
    print(count, sum, value)
print('After', count, sum, sum / count)
```

```
$ python averageloop.py
Before 0 0
1 9 9
2 50 41
3 62 12
4 65 3
5 139 74
6 154 15
After 6 154 25
```

An average just combines the counting and sum patterns and divides when the loop is done.

Filtering in a Loop

```
print('Before')
for value in [9, 41, 12, 3, 74, 15] :
    if value > 20:
        print 'Large number',value
print('After')
```

```
$ python search1.py
Before
Large number 41
Large number 74
After
```

We use an if statement in the loop to catch / filter the values we are looking for.

Search Using a Boolean Variable

```
found = False
print('Before', found)
for value in [9, 41, 12, 3, 74, 15] :
    if value == 3 :
        found = True
    print(found, value)
print('After', found)
```

```
$ python search1.py
Before False
False 9
False 41
False 12
True 3
True 74
True 15
After True
```

If we just want to search and know if a value was found, we use a variable that starts at False and is set to True as soon as we find what we are looking for.

How to Find the Smallest Value

```
largest_so_far = -1
print('Before', largest_so_far)
for the_num in [9, 41, 12, 3, 74, 15] :
    if the_num > largest_so_far :
        largest_so_far = the_num
    print(largest_so_far, the_num)

print('After', largest_so_far)
```

```
$ python largest.py
Before -1
9  9
41  41
41 12
41  3
74  74
74  15
After 74
```

How would we change this to make it find the smallest value in the list?

Finding the Smallest Value

```
smallest_so_far = -1
print('Before', smallest_so_far)
for the_num in [9, 41, 12, 3, 74, 15] :
    if the_num < smallest_so_far :
        smallest_so_far = the_num
    print(smallest_so_far, the_num)

print('After', smallest_so_far)
```

We switched the variable name to smallest_so_far and switch the > to <

Finding the Smallest Value

```
smallest_so_far = -1
print('Before', smallest_so_far)
for the_num in [9, 41, 12, 3, 74, 15] :
    if the_num < smallest_so_far :
        smallest_so_far = the_num
    print(smallest_so_far, the_num)

print('After', smallest_so_far)
```

```
$ python smallbad.py
Before -1
-1  9
-1  41
-1 12
-1  3
-1  74
-1  15
After -1
```

We switched the variable name to smallest_so_far and switch the > to <

Finding the Smallest Value

```
smallest = None
print('Before')
for value in [9, 41, 12, 3, 74, 15] :
    if smallest is None :
        smallest = value
    elif value < smallest :
        smallest = value
    print(smallest, value)
print('After', smallest)
```

```
$ python smallest.py
Before
9 9
9 41
9 12
3 3
3 74
3 15
After 3
```

We still have a variable that is the smallest so far. The first time through the loop smallest is None, so we take the first value to be the smallest.

The "is" and "is not" Operators

```
smallest = None
print('Before')
for value in [3, 41, 12, 9, 74, 15] :
    if smallest is None :
        smallest = value
    elif value < smallest :
        smallest = value
    print smallest, value

print('After', smallest)
```

- Python has an is operator that can be used in logical expressions
- Implies "is the same as"
- Similar to, but stronger than ==
- is not also is a logical operator

Summary

- While loops (indefinite)
- Infinite loops
- Using break
- Using continue
- For loops (definite)
- Iteration variables
- Loop idioms
- Largest or smallest

Acknowledgements / Contributions

These slides are Copyright 2010- Charles R. Severance (www.dr-chuck.com) of the University of Michigan School of Information and open.umich.edu and made available under a Creative Commons Attribution 4.0 License. Please maintain this last slide in all copies of the document to comply with the attribution requirements of the license. If you make a change, feel free to add your name and organization to the list of contributors on this page as you republish the materials.

Initial Development: Charles Severance, University of Michigan School of Information

… Insert new Contributors and Translators here

...