Introduction to Computer Programming in Python

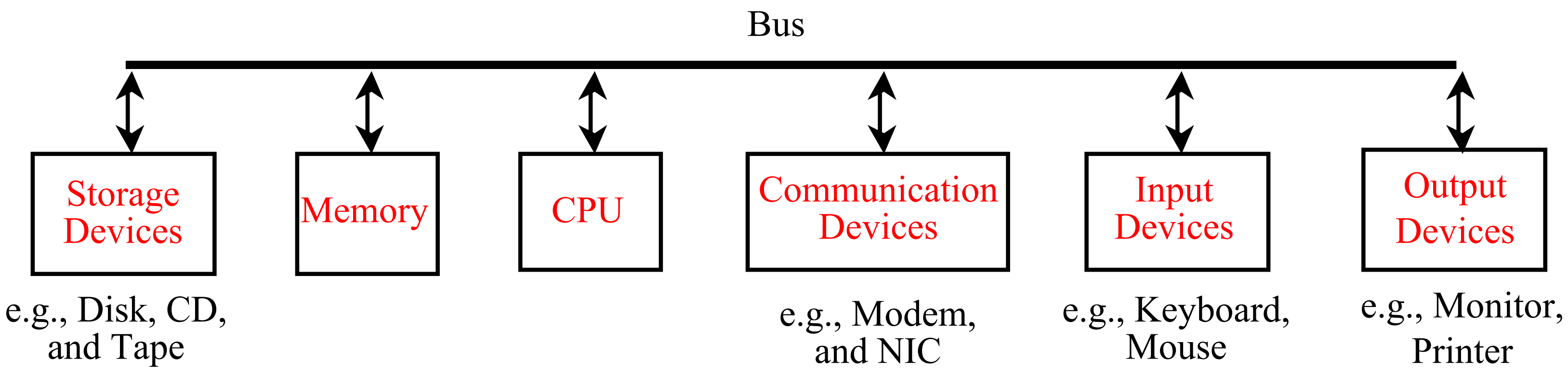
Dr. William C. Bulko

What is a Computer?

What is a Computer?

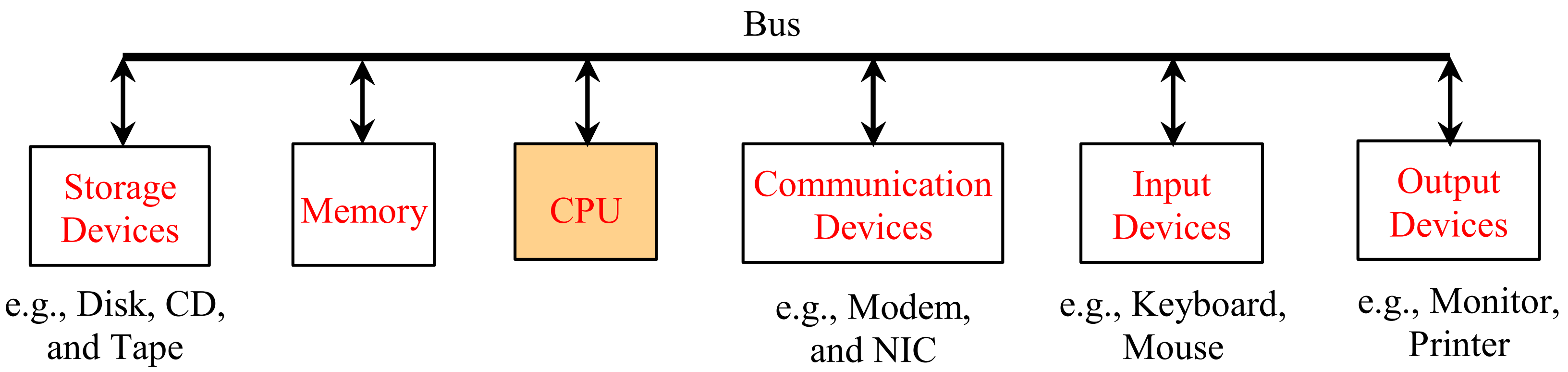
A typical computer consists of:

- a CPU
- memory
- a hard disk
- a monitor
- a keyboard and maybe a mouse
- and one or more communication devices.



CPU (Central Processing Unit)

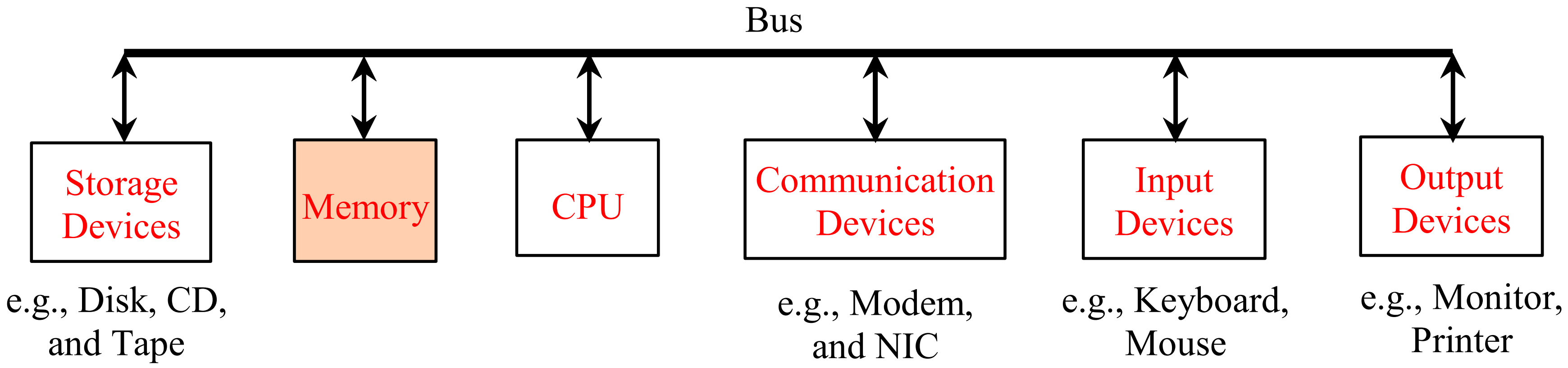
- the “brain” of a computer: It retrieves instructions from memory and executes them.
- CPU speed originally measured in megahertz (MHz): 1 MHz = 1,000,000 cycles per second.
- CPU speed has been improved continuously. Intel® Core™2 Quad Processor Q9550: 2.83 GHz. (1 gigahertz = 1000 megahertz).



Memory

Memory is a place to store data and program instructions for the CPU to execute. It is not the same thing as secondary storage.

- A program and its data must be brought to memory before they can be executed.
- A memory unit is an ordered sequence of *bytes*, each or which holds eight *bits*.
- Memory is never empty, but its initial content may be meaningless to your program. The current content of a memory byte is lost whenever new information is placed in it. (Think of the whiteboard in this room.)

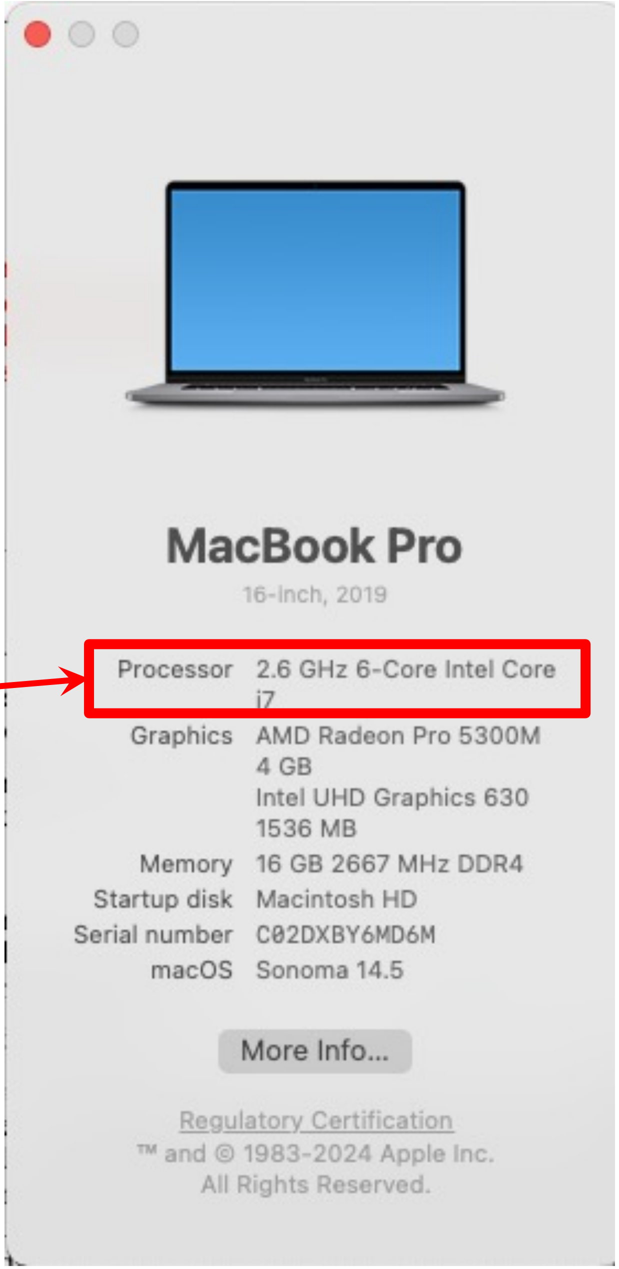


The “Fetch-Execute Cycle”

The Central Processing Unit (CPU) in a computer executes a “fetch-execute cycle” over and over.

- Fetch an instruction
- Execute the instruction
- Repeat

The speed of this cycle is defined by the speed of the processor chip.



MacBook Pro

16-inch, 2019

Processor	2.6 GHz 6-Core Intel Core i7
Graphics	AMD Radeon Pro 5300M 4 GB Intel UHD Graphics 630 1536 MB
Memory	16 GB 2667 MHz DDR4
Startup disk	Macintosh HD
Serial number	C02DXBY6MD6M
macOS	Sonoma 14.5

Regulatory Certification

™ and © 1983-2024 Apple Inc.
All Rights Reserved.

Programs and Programming Languages

Computer *programs*, collectively known as *software*, are simply sets of instructions to the computer.

- A programmer tells a computer what to do through programs. Without software, a computer is an empty machine.
- Computers do not understand human languages, so you need to use computer languages to communicate with them. Programs are written using various different *programming languages*.

Machine Language

- *Machine language* is a set of primitive instructions built into every computer. It is the language that that computer's CPU chip "speaks".
- The instructions are in the form of binary code, so you have to enter binary codes for various instructions.
- Programming in a native machine language is a very tedious process. Moreover, the programs are highly difficult to read and modify. For example: to add two numbers, you might write an instruction in binary that looks like this:

```
1101000100100011
```

High-Level Programming Languages

- *High-level programming languages* are even more English-like and easy to learn and program.
- The following is an example of a high-level language statement that computes the area of a circle with radius 5:

```
area = 5 * 5 * 3.1416
```

- This statement would actually be translated into multiple machine language instructions.

Popular High-Level Languages

COBOL (COmmon Business Oriented Language)

FORTRAN (FORmula TRANslation)

BASIC (Beginner’s All-purpose Symbolic Instructional Code)

Pascal (named for Blaise Pascal)

Ada (named for Ada Lovelace)

C (strongly associated with the UNIX operating system)

Visual Basic (Basic-like visual language developed by Microsoft)

Haskell (language for functional programming)

C++ (an object-oriented language, based on C)

Java

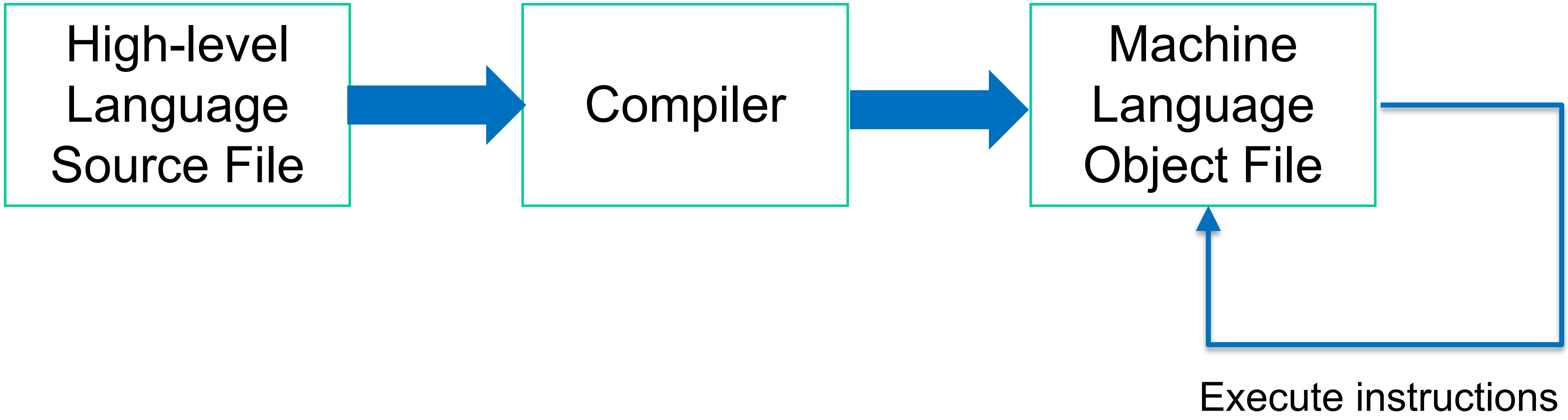
Swift

Python

Compilers vs. Interpreters

Compilers are programs that take a *source file* written in a high-level language, translate them, and produce an *object file* written in machine language.

The object file can then be executed by the computer.

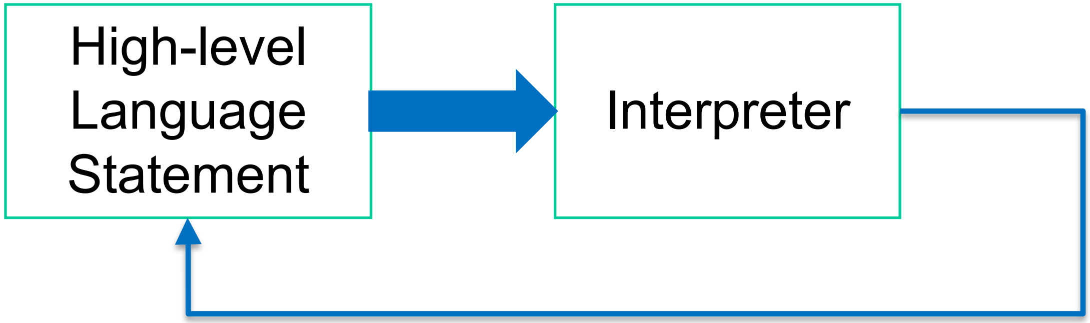


C, C++, and Java are typically compiled languages.

Compilers vs. Interpreters

Interpreters are programs that take individual instructions written in a high-level language, translate them, and execute them.

You can also accumulate instructions in a file and have the interpreter automatically run them one at a time.



Execute a single instruction

Python, Ruby, and Swift are typically interpreted languages.

Python

Some thoughts about programming

"The only way to learn a new programming language is by writing programs in it.”

-- B. Kernighan and D. Ritchie

"Computers are good at following instructions, but not at reading your mind."

-- D. Knuth

Program:

n. A magic spell cast over a computer allowing it to turn one's input into error messages.

tr. v. To engage in a pastime similar to banging one's head against a wall, but with fewer opportunities for reward.

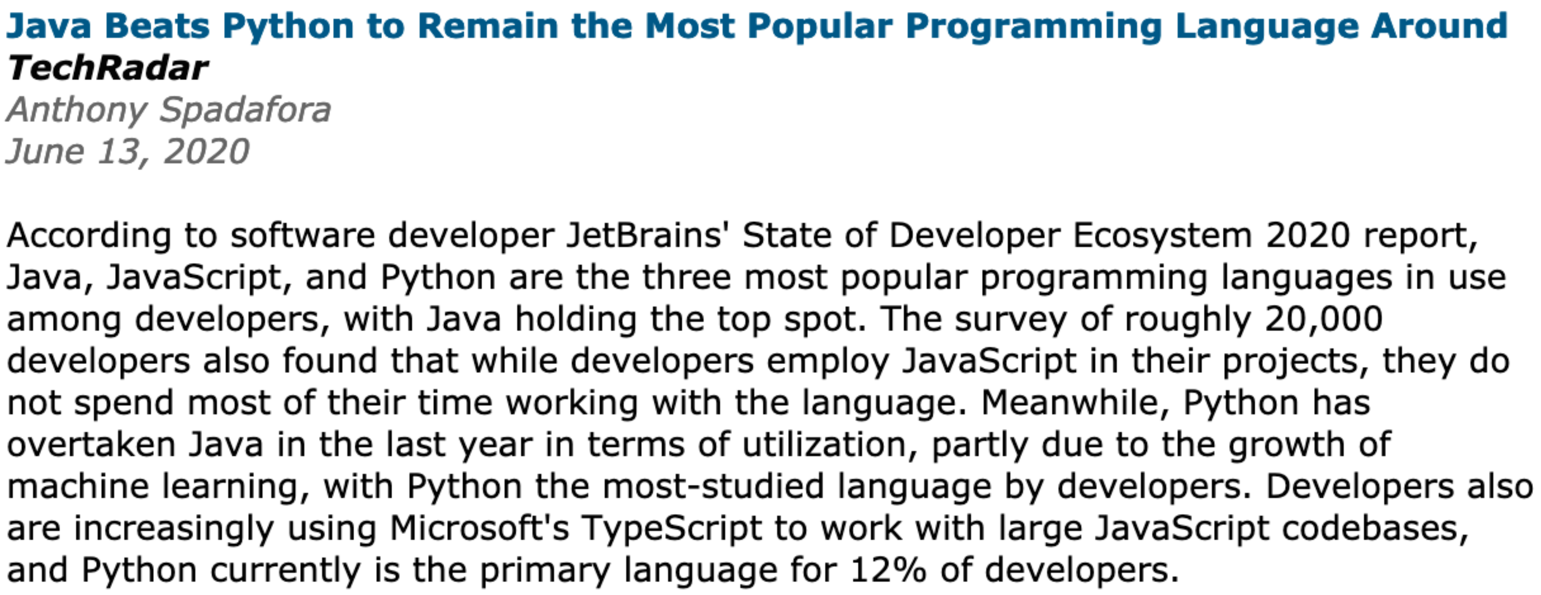
What is Python?

A high-level language developed by Guido van Rossum in the Netherlands in the late 1980s. Released in 1991.

Named after Monty Python

Clean, concise syntax with compact design

- Easy to learn and remember
- Can create powerful programs quickly
- Language constructs are simple



Java Beats Python to Remain the Most Popular Programming Language Around
TechRadar
Anthony Spadafora
June 13, 2020

According to software developer JetBrains' State of Developer Ecosystem 2020 report, Java, JavaScript, and Python are the three most popular programming languages in use among developers, with Java holding the top spot. The survey of roughly 20,000 developers also found that while developers employ JavaScript in their projects, they do not spend most of their time working with the language. Meanwhile, Python has overtaken Java in the last year in terms of utilization, partly due to the growth of machine learning, with Python the most-studied language by developers. Developers also are increasingly using Microsoft's TypeScript to work with large JavaScript codebases, and Python currently is the primary language for 12% of developers.

What is Python?

- Python is a general purpose programming language. That means you can use Python to write code for any programming tasks.
- Python was used to write
    - the Google search engine
    - mission critical projects in NASA
    - programs for processing financial transactions at the NY Stock Exchange.
- Python is interpreted, which means that Python code is translated and executed by an interpreter one statement at a time.
- Python is an object-oriented programming language. Object-oriented programming is a powerful tool for developing reusable software.

Getting Python

To install it on your personal computer / laptop, you can download it for free at:

www.python.org/downloads

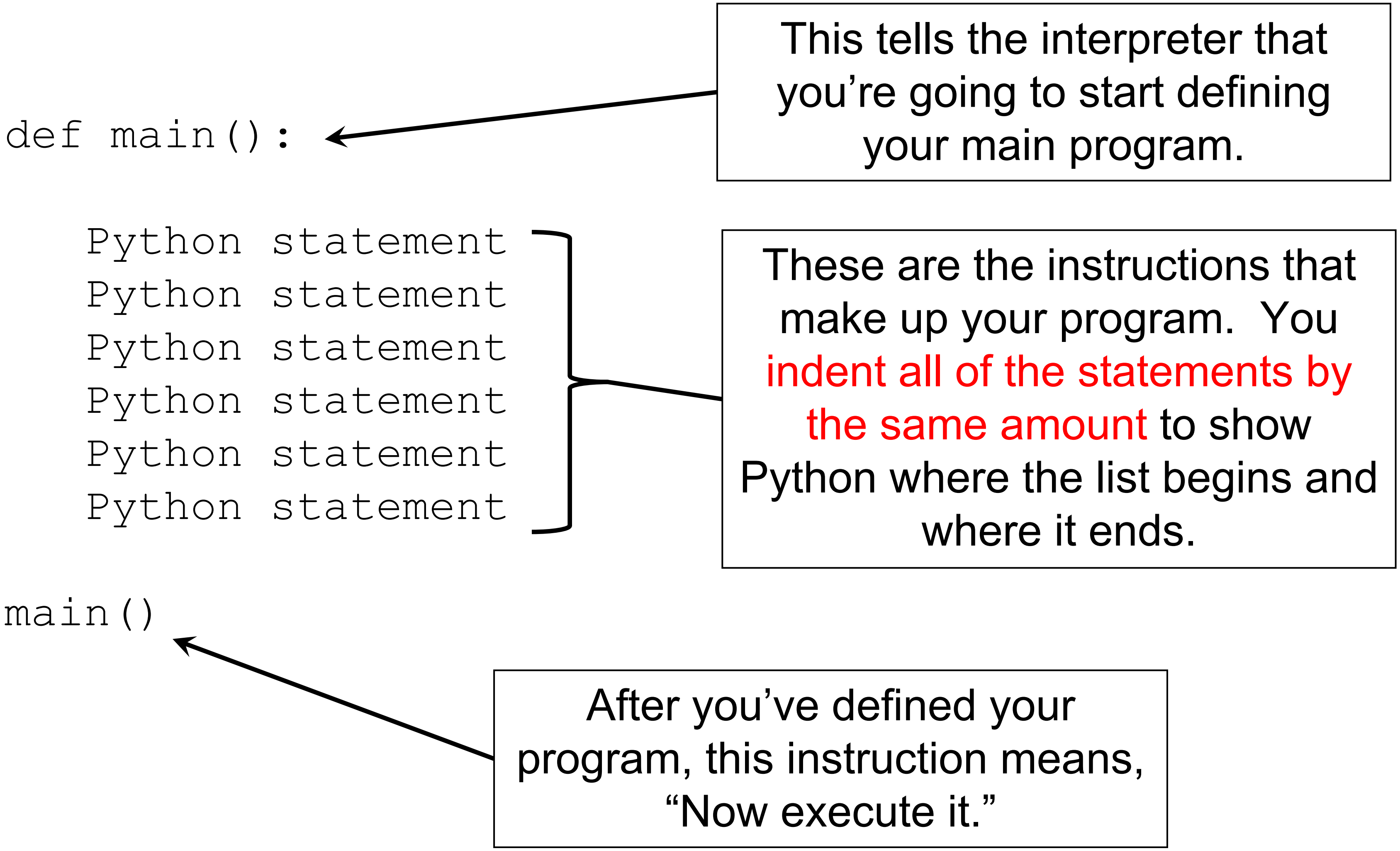
- It’s available for both Windows and Mac OS.
- If you have a Mac, you may already have it preinstalled.
- It comes with an editor and user interface called IDLE.
- There are two major versions: Python 2 and Python 3. Python 3 is newer, and it is not backward compatible, so if you write a program in one version, it may not work properly (or at all) in the other interpreter.

It has already been installed on the workstations in this classroom.

A simple Python program

```
def main():

   # Display two messages
   print("Welcome to Python!")
   print("Go Horns Go")

   # Evaluate an arithmetic expression
   print((10.5 + 2 * 3) / (45 - 3.5))

main()
```

The framework of a simple Python program



```
def main():

    Python statement
    Python statement
    Python statement
    Python statement
    Python statement
    Python statement

main()
```

This tells the interpreter that you’re going to start defining your main program.

These are the instructions that make up your program. You indent all of the statements by the same amount to show Python where the list begins and where it ends.

After you’ve defined your program, this instruction means, “Now execute it.”

Program Documentation

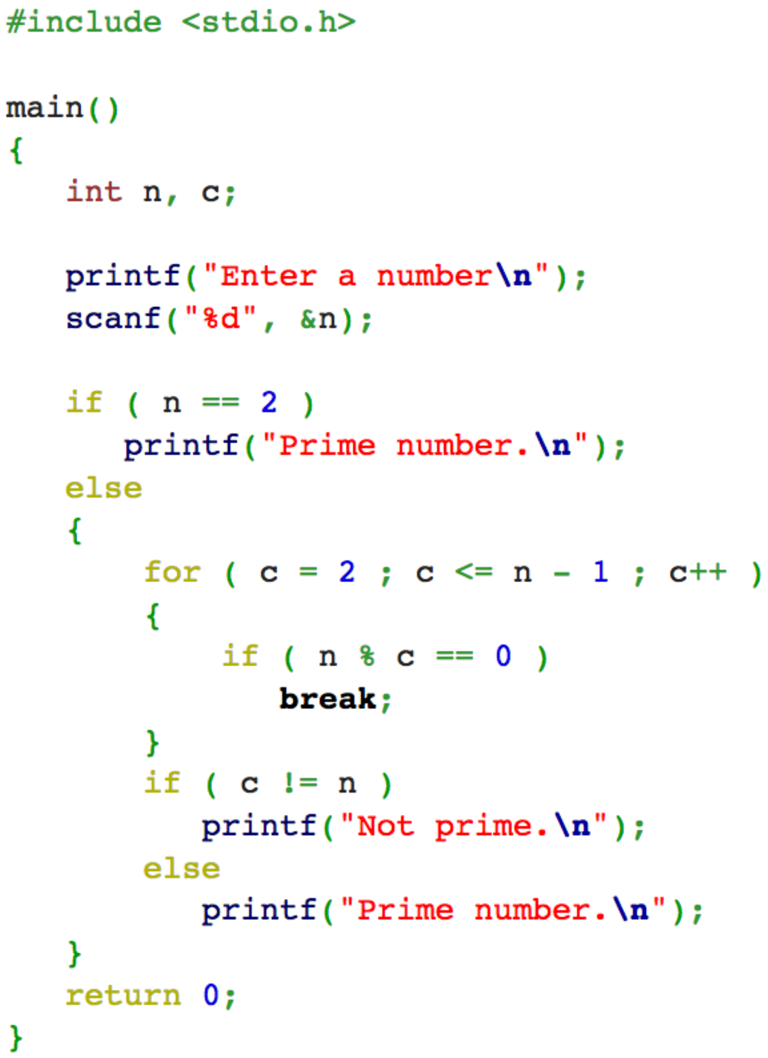
Documentation refers to comments included within a source code file that explains what the code does.

- Programmers typically include a *file header*: a summary at the beginning of each file explaining what the file contains, what the code does, and what key features or techniques appear.
- From a classroom perspective: you should always include your name, class section, instructor, date, and a brief description at the beginning of the program.
- Comments also appear interspersed within the file:
  - Before each function or class definition (i.e., program subdivision)
  - Before each major block of code that performs a significant task
  - Before or next to any line of code that may not be easy to understand

Programming Style

Every programming language has its own unique *style*: conventions that all good programmers follow to make programs clear and easy to read, understand, debug, and maintain.

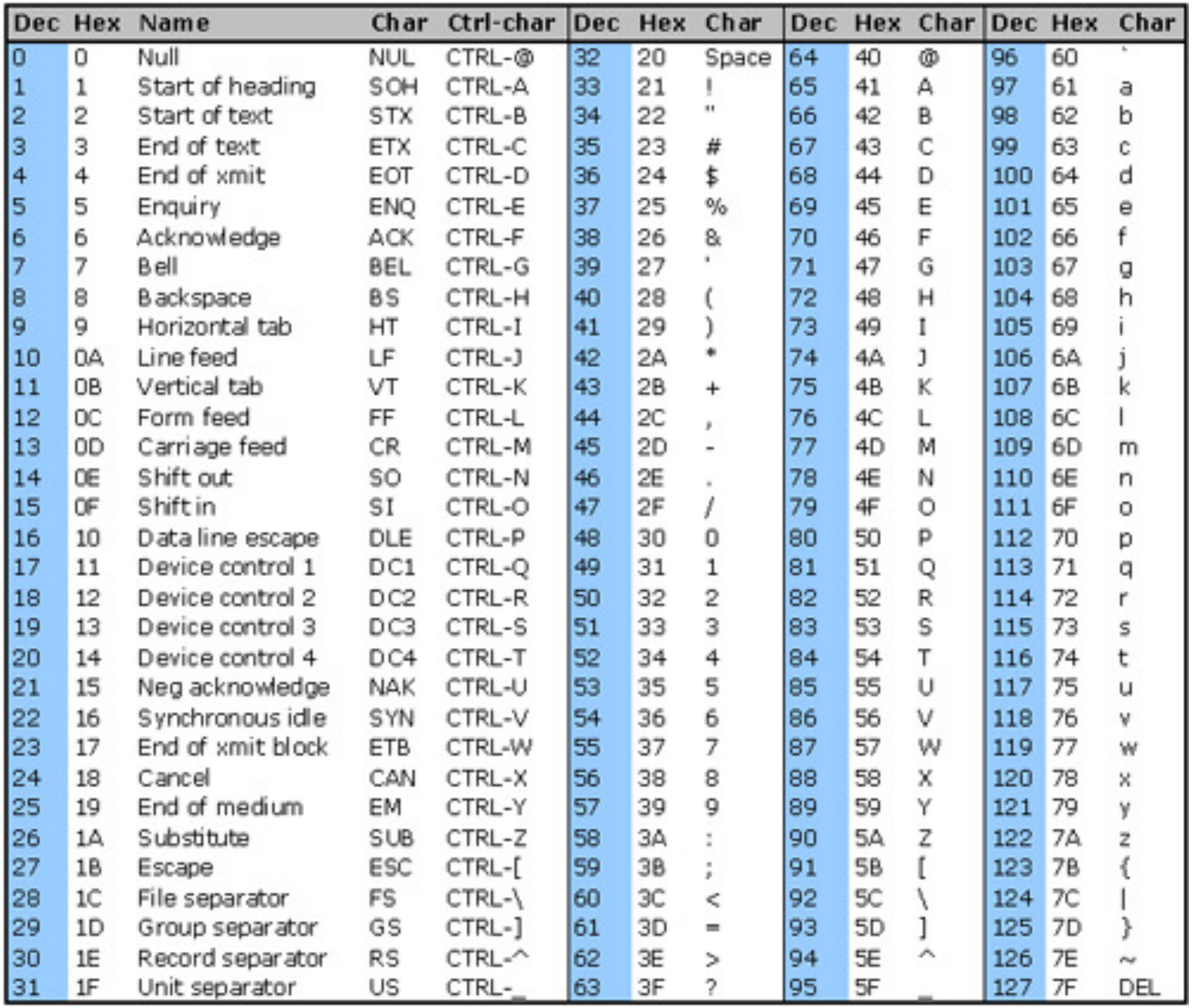
- In Python, each level of indentation is 4 spaces.
- Use blank lines to separate segments of the code.
- We will learn more elements of style as we go along.



```
#include <stdio.h>

main()
{
   int n, c;

   printf("Enter a number\n");
   scanf("%d", &n);

   if ( n == 2 )
      printf("Prime number.\n");
   else
   {
       for ( c = 2 ; c <= n - 1 ; c++ )
       {
           if ( n % c == 0 )
              break;
       }
       if ( c != n )
          printf("Not prime.\n");
       else
          printf("Prime number.\n");
   }
   return 0;
}
```

Example of a C program

ASCII table



Dec	Hex	Name	Char	Ctrl-char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	0	Null	NUL	CTRL-@	32	20	Space	64	40	@	96	60	`
1	1	Start of heading	SOH	CTRL-A	33	21	!	65	41	A	97	61	a
2	2	Start of text	STX	CTRL-B	34	22	"	66	42	B	98	62	b
3	3	End of text	ETX	CTRL-C	35	23	#	67	43	C	99	63	c
4	4	End of xmit	EOT	CTRL-D	36	24	$	68	44	D	100	64	d
5	5	Enquiry	ENQ	CTRL-E	37	25	%	69	45	E	101	65	e
6	6	Acknowledge	ACK	CTRL-F	38	26	&	70	46	F	102	66	f
7	7	Bell	BEL	CTRL-G	39	27	'	71	47	G	103	67	g
8	8	Backspace	BS	CTRL-H	40	28	(	72	48	H	104	68	h
9	9	Horizontal tab	HT	CTRL-I	41	29	)	73	49	I	105	69	i
10	0A	Line feed	LF	CTRL-J	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	VT	CTRL-K	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	FF	CTRL-L	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage feed	CR	CTRL-M	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	SO	CTRL-N	46	2E	.	78	4E	N	110	6E	n
15	0F	Shift in	SI	CTRL-O	47	2F	/	79	4F	O	111	6F	o
16	10	Data line escape	DLE	CTRL-P	48	30	0	80	50	P	112	70	p
17	11	Device control 1	DC1	CTRL-Q	49	31	1	81	51	Q	113	71	q
18	12	Device control 2	DC2	CTRL-R	50	32	2	82	52	R	114	72	r
19	13	Device control 3	DC3	CTRL-S	51	33	3	83	53	S	115	73	s
20	14	Device control 4	DC4	CTRL-T	52	34	4	84	54	T	116	74	t
21	15	Neg acknowledge	NAK	CTRL-U	53	35	5	85	55	U	117	75	u
22	16	Synchronous idle	SYN	CTRL-V	54	36	6	86	56	V	118	76	v
23	17	End of xmit block	ETB	CTRL-W	55	37	7	87	57	W	119	77	w
24	18	Cancel	CAN	CTRL-X	56	38	8	88	58	X	120	78	x
25	19	End of medium	EM	CTRL-Y	57	39	9	89	59	Y	121	79	y
26	1A	Substitute	SUB	CTRL-Z	58	3A	:	90	5A	Z	122	7A	z
27	1B	Escape	ESC	CTRL-[	59	3B	;	91	5B	[	123	7B	{
28	1C	File separator	FS	CTRL-\	60	3C	<	92	5C	\	124	7C	|
29	1D	Group separator	GS	CTRL-]	61	3D	=	93	5D	]	125	7D	}
30	1E	Record separator	RS	CTRL-^	62	3E	>	94	5E	^	126	7E	~
31	1F	Unit separator	US	CTRL-_	63	3F	?	95	5F	_	127	7F	DEL

Special characters

`\n` is actually a symbol for the ASCII character “line feed” <LF>.

The backslash is used to indicate an “escape sequence”, identifying the next character as something special.

`\n`: the “new line” escape sequence (like hitting return)

`\t`: the tab escape sequence (like hitting tab)

`\"`: double quote (print a real double quote instead of beginning or ending a string)

`\'`: single quote (print a real apostrophe instead of beginning or ending a string)

`\\`: a real backslash, if you ever have the need to print one

Variables and Assignment Statements

Variables: What can we name them?

- Variable names must start with a letter or the underscore (“`_`”) character.
- After that, it can be followed by any number of letters, underscores, or digits.
- Variable names are case-sensitive, so “`score`” is different from “`Score`”.
- You must avoid using reserved words as variable names: these are words that have a special meaning in a programming language such as Python.
    - For example: `def`, `str`, `print`, etc.
    - IDLE displays reserved words in color to help you recognize them, which is useful since most people don’t know all of them.

Variables: What *should* we name them?

In addition to the hard-and-fast rules on the previous chart, there are also naming conventions that all (good) programmers obey:

- You should choose meaningful names that describe what the purpose of the variable is. This helps people reading the program (including you) understand what the code is doing.

  Use `max` rather than `m`

  Use `item` rather than `c`

- Variable names should begin with a <u>lowercase</u> letter.

- It is common to combine multiple words (such as `avgHeight`) into a variable name in order to be descriptive. When you do this, improve readibilty by using lowercase for the first "word" and uppercase for subsequent words. (This is called "camelCase".)

A program segment using assignment statements

Execute each of these statements in sequence (on paper, not using IDLE) and show what each one does. (Beware of tricks!)

```
print (“Start here”)
firstNum = 3 + (16 - 4) / 3
print (“The first number is: ”)
secondNum = 13 % 2 – 1
print (“The second number is: secondNum”)
thirdNum = secondNum + 5
secondNum = secondNum + 3
print (“The numbers are: ”, firstNum,
          secondNum, thirdNum)
```

Exercise

A postage stamp vending machine only accepts dollar bills.

A customer inserts dollar bills into the vending machine and then pushes the "purchase" button. The vending machine gives the customer as many 68-cent first-class stamps as possible, and returns the change in the form of 1-cent stamps.

For example:

- if the customer inserts one dollar, the machine gives out one 68-cent stamp and 32 1-cent stamps.
- If the customer enters two dollars, the machine gives out two 68-cent stamps and 64 1-cent stamps.
- If the customer enters three dollars, the machine gives out four 68-cent stamps and 28 1-cent stamps.

Write a program that calculates the number of 68-cent stamps and 1-cent stamps given out for any positive number of dollars.

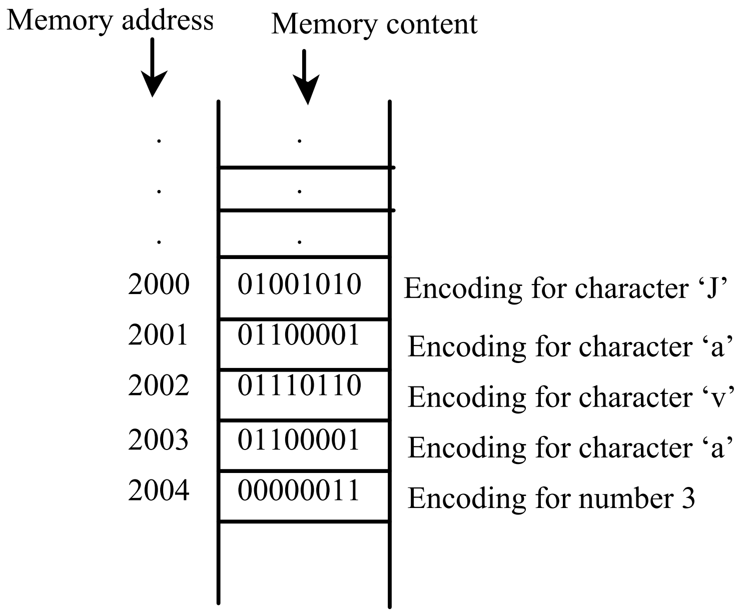
Introduction to Computer Programming in Python

Dr. William C. Bulko

Data Types

How is data stored?

- Data of various kinds, such as numbers, characters, and strings, are encoded as a series of *bits* (zeros and ones).
- Computers use zeros and ones because digital devices have two stable states, which are referred to as *zero* and *one* by convention.



Memory address	Memory content	
.	.	
.	.	
.	.	
2000	01001010	Encoding for character ‘J’
2001	01100001	Encoding for character ‘a’
2002	01110110	Encoding for character ‘v’
2003	01100001	Encoding for character ‘a’
2004	00000011	Encoding for number 3

- A *byte* is the smallest unit of storage a programmer can access.
- Characters such as “J” or small numbers such as 3 can be stored in a single byte. If a program needs to store a large number that cannot fit into a single byte, it may be split across a number of adjacent bytes in memory.
- The way data is encoded within a byte varies depending on whether the data represents a number, a character, or something else. Programmers need not be concerned about the encoding and decoding of data, which is performed automatically by the system based on the encoding scheme.

What is a data type?

A *data type* is the kind of value represented by a constant or stored by a variable.

So far, you have seen the following three data types:

- `str`: represents text (a string)
  - Currently, we use it for input and output
  - You’ll see more uses for it later if we have time
- `int`: whole numbers
  - Computations are exact
- `float`: real numbers (numbers with decimal points)
  - Large range, but fixed precision
  - Computations are <u>not</u> exact

    Example: `1.0 / 3.0 = .3333333333333333`

They are important because they are represented within the computer in different ways.

What is a data type?

It would be nice if you looked at the character string “25” and could do arithmetic with it.

However, the `int` 25 (a number) is represented in binary in the computer by:

```
0000 0000 0001 1001
```

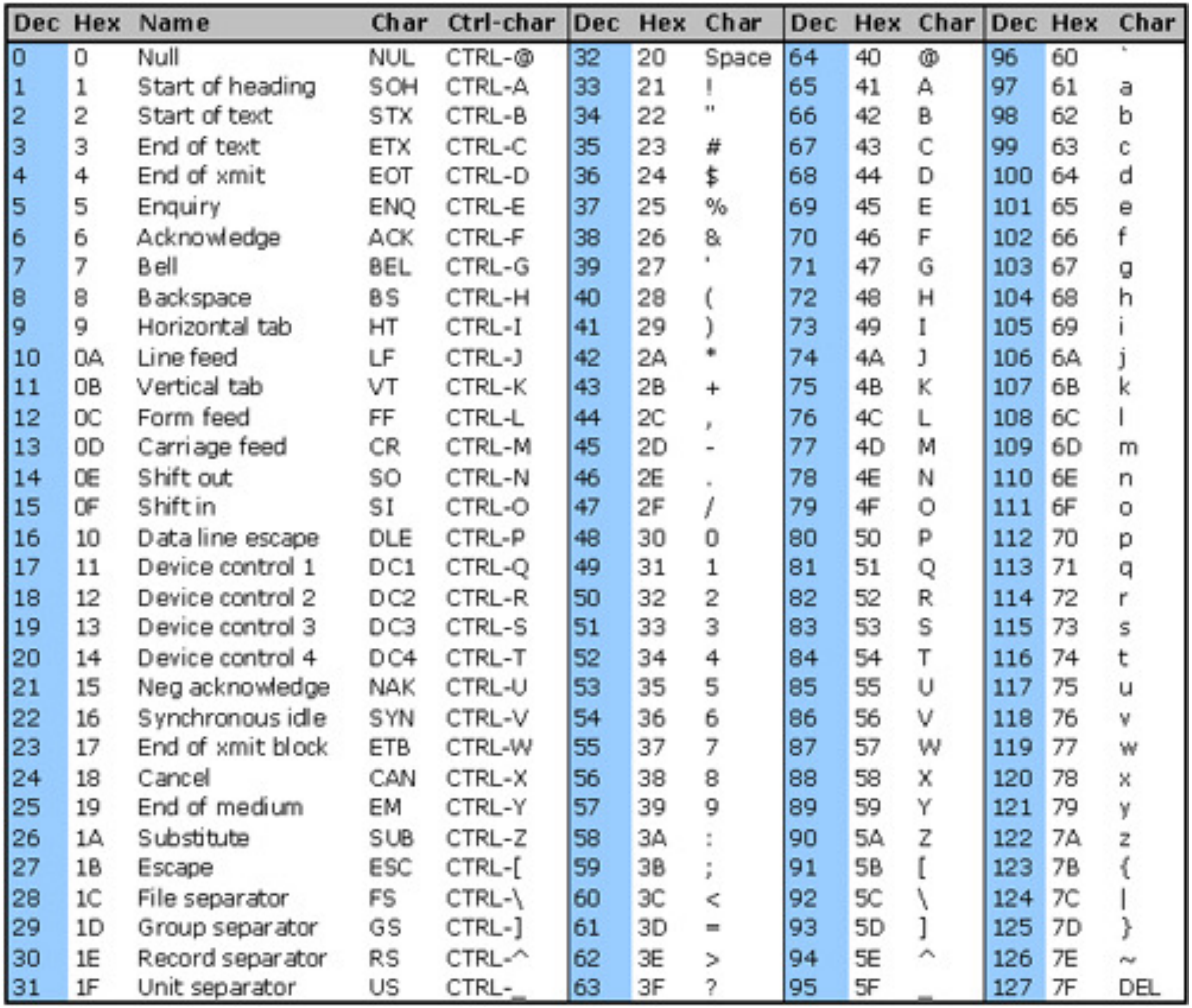
And the `string` “25” (two characters) is represented by:

```
0011 0010 0011 0101
```

`float` numbers are represented in an even more complicated way, since you have to account for an exponent. (Think “scientific notation”.) So the number “25.0” is represented in yet a third way.

Chart from day 1

ASCII table



Dec	Hex	Name	Char	Ctrl-char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	0	Null	NUL	CTRL-@	32	20	Space	64	40	@	96	60	`
1	1	Start of heading	SOH	CTRL-A	33	21	!	65	41	A	97	61	a
2	2	Start of text	STX	CTRL-B	34	22	"	66	42	B	98	62	b
3	3	End of text	ETX	CTRL-C	35	23	#	67	43	C	99	63	c
4	4	End of xmit	EOT	CTRL-D	36	24	$	68	44	D	100	64	d
5	5	Enquiry	ENQ	CTRL-E	37	25	%	69	45	E	101	65	e
6	6	Acknowledge	ACK	CTRL-F	38	26	&	70	46	F	102	66	f
7	7	Bell	BEL	CTRL-G	39	27	'	71	47	G	103	67	g
8	8	Backspace	BS	CTRL-H	40	28	(	72	48	H	104	68	h
9	9	Horizontal tab	HT	CTRL-I	41	29	)	73	49	I	105	69	i
10	0A	Line feed	LF	CTRL-J	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	VT	CTRL-K	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	FF	CTRL-L	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage feed	CR	CTRL-M	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	SO	CTRL-N	46	2E	.	78	4E	N	110	6E	n
15	0F	Shift in	SI	CTRL-O	47	2F	/	79	4F	O	111	6F	o
16	10	Data line escape	DLE	CTRL-P	48	30	0	80	50	P	112	70	p
17	11	Device control 1	DC1	CTRL-Q	49	31	1	81	51	Q	113	71	q
18	12	Device control 2	DC2	CTRL-R	50	32	2	82	52	R	114	72	r
19	13	Device control 3	DC3	CTRL-S	51	33	3	83	53	S	115	73	s
20	14	Device control 4	DC4	CTRL-T	52	34	4	84	54	T	116	74	t
21	15	Neg acknowledge	NAK	CTRL-U	53	35	5	85	55	U	117	75	u
22	16	Synchronous idle	SYN	CTRL-V	54	36	6	86	56	V	118	76	v
23	17	End of xmit block	ETB	CTRL-W	55	37	7	87	57	W	119	77	w
24	18	Cancel	CAN	CTRL-X	56	38	8	88	58	X	120	78	x
25	19	End of medium	EM	CTRL-Y	57	39	9	89	59	Y	121	79	y
26	1A	Substitute	SUB	CTRL-Z	58	3A	:	90	5A	Z	122	7A	z
27	1B	Escape	ESC	CTRL-[	59	3B	;	91	5B	[	123	7B	{
28	1C	File separator	FS	CTRL-\	60	3C	<	92	5C	\	124	7C	|
29	1D	Group separator	GS	CTRL-]	61	3D	=	93	5D	]	125	7D	}
30	1E	Record separator	RS	CTRL-^	62	3E	>	94	5E	^	126	7E	~
31	1F	Unit separator	US	CTRL-_	63	3F	?	95	5F	_	127	7F	DEL

Data Type Conversion

Python provides functions to *explicitly* convert numbers from one type to another:

`float` (<*number, variable, or string*>)
`int` (<*number, variable, or string*>)
`str` (<*number or variable*>)

Note: `int` truncates, meaning it throws away the decimal point and anything that comes after it. If you need to *round off* to the nearest whole number, use:

`round` (<*number or variable*>)

Data Types and arithmetic operations

Most arithmetic operations behave as you would expect for all data types.

- Combining two `float`s results in an `float`.
- Combining two `int`s results in an `int` (provided you do division with //).
- Dividing two `ints` using float division is an exception: it behaves as you probably want it to. For instance, `5 / 2` gives you 2.5.

Python will figure out what the result should be and make the result the appropriate data type.

Keyboard Input

The `input()` function is used to read data from the user during program execution.

Format:

`input (<`*prompt string*`>)`

When it's called:

- It displays the "prompt string", a `str`. The intent is that it should be a message to the user that the program is waiting for the user to type in a string.
- It will wait until the user types something and hits the "Enter" or "Return" key.
- It returns whatever the user typed as a `str` as a *return value.*

Line Continuation

If your line of Python code is too long, you can extend it using the backslash character “\”

- Place it at the end of the line and it “escapes” the carriage return at the end
- The Python interpreter will assume the next line is part of the same line

Example:

```
name = “Jarvis”
print (“Hello, my name is ”, name, \
        “ How are you?”)
```

produces:

```
Hello, my name is Jarvis.  How are you?
```

In-class Exercise:

A bowler’s handicap in a particular league is calculated with the following formula:

```
handicap = (200 – average) * 80%
```

where the bowler’s average is truncated to the next lower integer, and the handicap is also truncated to the next lower integer.

For example: if the average is 147.8, you would calculate

```
handicap = (200 – 147) * 80%
```

```
               = 53 * 80% = 42.4  truncated to 42.
```

Write a complete Python program that calculates a bowler’s average and handicap after three games. Prompt the user to enter three bowling games (integers between zero and 300) one at a time, print them out neatly, print out the average of the three numbers, and print out the handicap. Remember that the final handicap is an integer.

Python Libraries

Some of the many Python libraries:

`os`	interact with the operating system (change directory)
`sys`	interact with the system (command-line arguments)
`math`	access to `log()`, `sin()`, `cos()`, `sqrt`, `pi`, etc.
`random`	random number generation
`time`	clock and time functions

Introduction to Computer Programming in Python

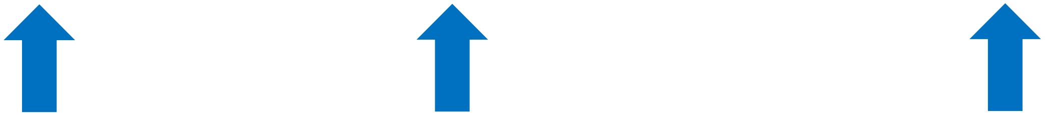
Dr. William C. Bulko

String Formatting

Formatting Strings for Output

If you wanted to print out the value of the variable value in a particular way, you can format it first by using the `format()` method. It looks something like this:

`newString = ”`*format string*`”.format(`*data items*`)`



A variable where you want to store the desired (formatted) string

A string that describes what you want the output string to look like.

It includes a format specification for each data item.

A list of constants, variables, and expressions that you want to format

Format Specifications

A format specification starts with `{:` and ends with `}` .

Each format specification contains up to three things:

- An optional “justification symbol”

`>`	right-justified
`^`	centered
`<`	left-justified

- A field size

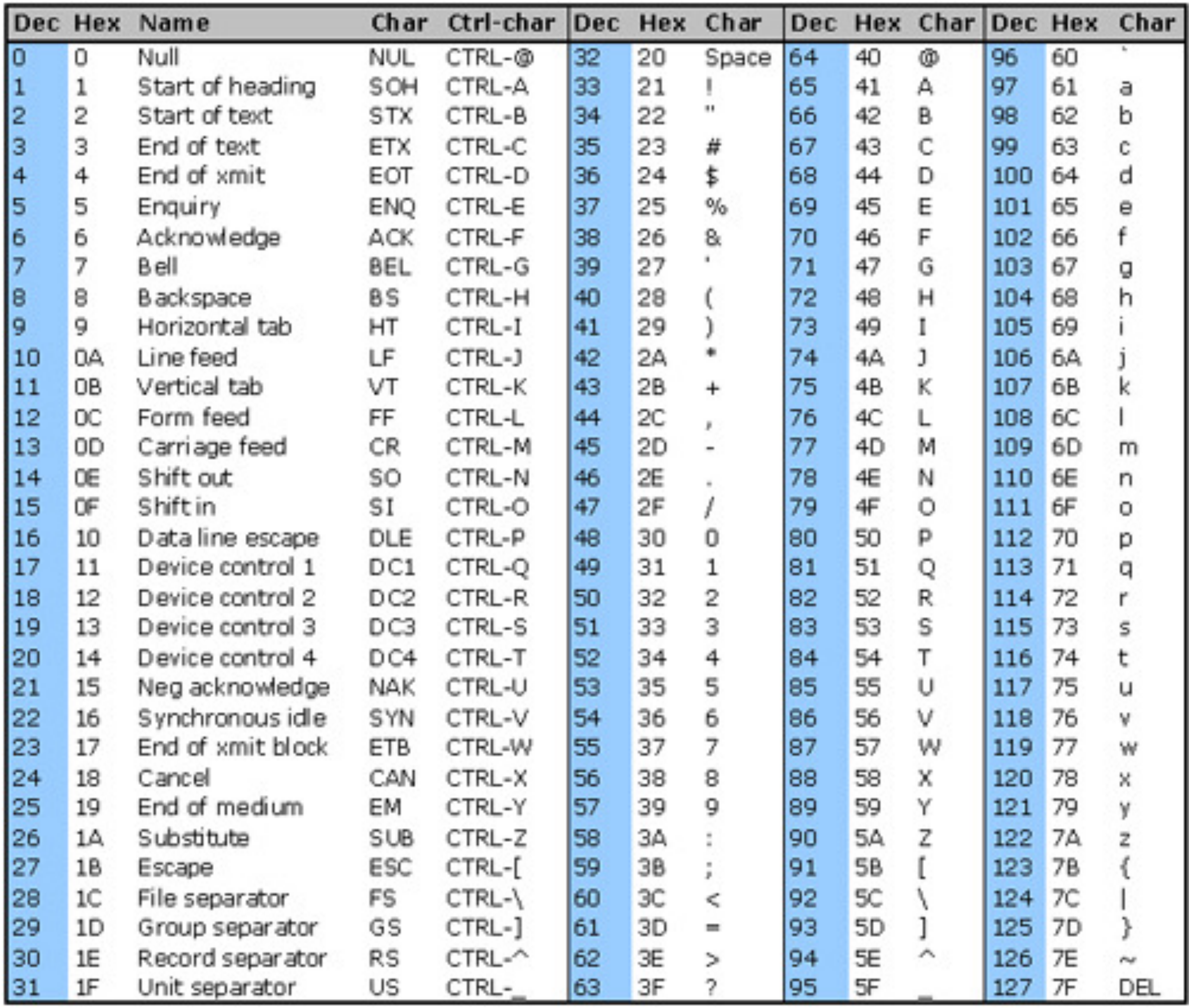
- A Type Code indicating the type of the data item

`d`	Integer (“decimal”)
`f`	Floating-point number
`e`	Scientific notation
(*nothing*)	String

Booleans

Chart from day 1

ASCII table



Dec	Hex	Name	Char	Ctrl-char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	0	Null	NUL	CTRL-@	32	20	Space	64	40	@	96	60	`
1	1	Start of heading	SOH	CTRL-A	33	21	!	65	41	A	97	61	a
2	2	Start of text	STX	CTRL-B	34	22	"	66	42	B	98	62	b
3	3	End of text	ETX	CTRL-C	35	23	#	67	43	C	99	63	c
4	4	End of xmit	EOT	CTRL-D	36	24	$	68	44	D	100	64	d
5	5	Enquiry	ENQ	CTRL-E	37	25	%	69	45	E	101	65	e
6	6	Acknowledge	ACK	CTRL-F	38	26	&	70	46	F	102	66	f
7	7	Bell	BEL	CTRL-G	39	27	'	71	47	G	103	67	g
8	8	Backspace	BS	CTRL-H	40	28	(	72	48	H	104	68	h
9	9	Horizontal tab	HT	CTRL-I	41	29	)	73	49	I	105	69	i
10	0A	Line feed	LF	CTRL-J	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	VT	CTRL-K	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	FF	CTRL-L	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage feed	CR	CTRL-M	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	SO	CTRL-N	46	2E	.	78	4E	N	110	6E	n
15	0F	Shift in	SI	CTRL-O	47	2F	/	79	4F	O	111	6F	o
16	10	Data line escape	DLE	CTRL-P	48	30	0	80	50	P	112	70	p
17	11	Device control 1	DC1	CTRL-Q	49	31	1	81	51	Q	113	71	q
18	12	Device control 2	DC2	CTRL-R	50	32	2	82	52	R	114	72	r
19	13	Device control 3	DC3	CTRL-S	51	33	3	83	53	S	115	73	s
20	14	Device control 4	DC4	CTRL-T	52	34	4	84	54	T	116	74	t
21	15	Neg acknowledge	NAK	CTRL-U	53	35	5	85	55	U	117	75	u
22	16	Synchronous idle	SYN	CTRL-V	54	36	6	86	56	V	118	76	v
23	17	End of xmit block	ETB	CTRL-W	55	37	7	87	57	W	119	77	w
24	18	Cancel	CAN	CTRL-X	56	38	8	88	58	X	120	78	x
25	19	End of medium	EM	CTRL-Y	57	39	9	89	59	Y	121	79	y
26	1A	Substitute	SUB	CTRL-Z	58	3A	:	90	5A	Z	122	7A	z
27	1B	Escape	ESC	CTRL-[	59	3B	;	91	5B	[	123	7B	{
28	1C	File separator	FS	CTRL-\	60	3C	<	92	5C	\	124	7C	|
29	1D	Group separator	GS	CTRL-]	61	3D	=	93	5D	]	125	7D	}
30	1E	Record separator	RS	CTRL-^	62	3E	>	94	5E	^	126	7E	~
31	1F	Unit separator	US	CTRL-_	63	3F	?	95	5F	_	127	7F	DEL

Lexicographic Order

- Strings are rated according to *lexicographic* order, <u>not</u> dictionary order
- Words are ordered from A-Za-z

  Capital letters first in alphabetical order

  Lower-case letters second in alphabetical order
- This means <u>all</u> upper-case letters come before <u>all</u> lower-case letters

Introduction to Computer Programming in Python

Dr. William C. Bulko

Conditionals

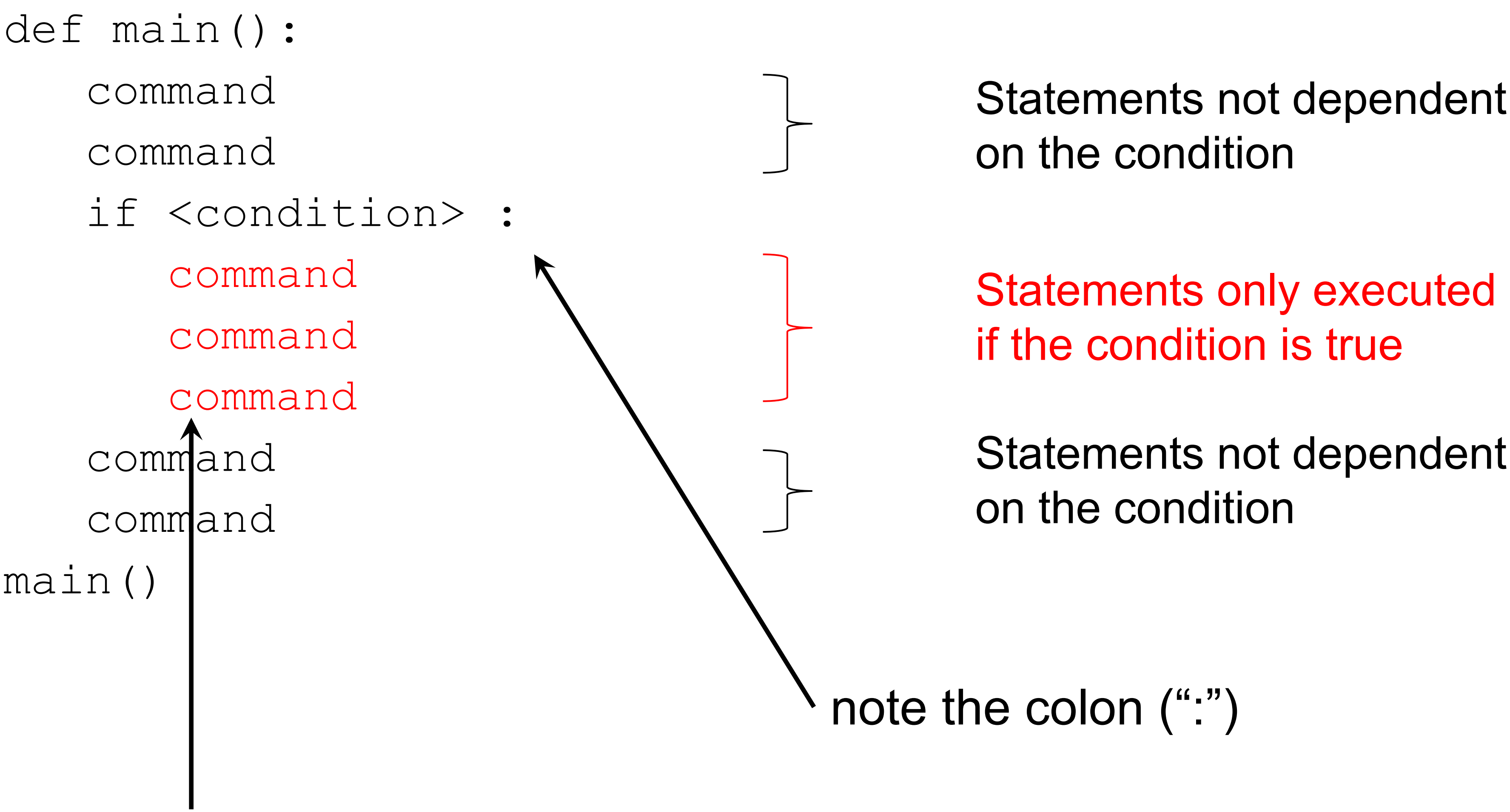
Conditionals

Conditional statements give you the ability to specify different instructions based on whether or not a specified condition is met.

Test for the condition (to see if it's true)

- If condition is met, perform the action
- If condition is not met, skip the action

The Python conditional statement: `if`



```
def main():
    command
    command
    if <condition> :
        command
        command
        command
    command
    command
main()
```

Statements not dependent on the condition

Statements only executed if the condition is true

Statements not dependent on the condition

note the colon (“:”)

Indentation is very important!

In-class exercise

Write a complete program that asks the user to enter a number.

If the number is 9, print a message indicating that they entered your favorite number. Then, on a separate line, print out the square of the number entered by the user.

“Area” exercise

Write a complete program that calculates the area of a geometric object.

It asks the user, “what is the shape of the object?” The user must type in either the string "circle" or "square".

It asks the user for a number.

If the user entered, “square”, it calculates and prints the area of the square whose side is equal to the number entered.

If the user entered, “circle”, it calculates and prints the area of the circle whose radius is equal to the number entered.

The `if-else` statement

```
if <condition> :
    command
    command
    command
else :
    command
    command
command
command
```

Statements only executed if the condition is true

Statements only executed if the condition is false

Statements not dependent on the condition

- Note the <u>two</u> colons (“:”)
- Note the indentation

Even / Odd Exercise

Write a complete program that asks the user to enter a number.

- If the number is even, print the number, followed by “ is even”.
- If the number is odd, print the number, followed by “ is odd”.

Hint: use the remainder operator “%” !

The if-elif-else statement

```
if <condition> :
    command
    command
    command
elif <condition> :
    command
    command
elif <condition> :
    command
    command
else :
    command
    command
command
command
```

You can have as many of these blocks as you like

These statements are only executed if all of the conditions fail

Conditionals: nested `if`s

You can put `if` statements inside the body of the `if` (or `elif` or `else`) statement:

```
if (<condition>):

   if(<some other condition>):
      command
   else:
      command

elif (<condition>):
…
```

What is the expected output?

```
if (125 < 140):
   print ("first one")
elif (156 >= 140):
   print ("second one")
else:
   print ("third one")
```

A. first one

B. second one

C. third one

D. first one
second one

Gotchas with conditionals

Exactly one of the clauses of an `if-elif-else` statement will be executed

- Only the first `True` condition
- Think carefully about the construction of your `if` statements before coding
- Think about a flowchart: you will only follow ONE arrow at a time

What is the expected output?

```
x = 1 - (8/9)

if(x == (1/9)):
     print ("It’s one-ninth")
else:
     print ("It’s not one-ninth")
```

Gotchas with conditionals #2

Arithmetic with floating-point numbers is not necessarily exact!

- 1/9 = .1111111111111 . . . out to infinity
- But a computer can't and won't store an infinite number of digits. It will have some fixed number of 1s.
- Therefore, .1111111 . . . + .8888888 . . . does not add up to exactly 1.

When you want to compare two floating-point numbers, it is better to test to see if they differ by a sufficiently small amount.

```
        if ( (x - (1/9)) < .000001):
```

Exercise: Do you want fries with that?

Write a complete program that does the following:

```
Welcome to Gonzo Burger!

Enter a “1” if you want a hamburger, or a “2” if
you want a cheeseburger.

Order: 2

Thank you!  Next, enter a “1” if you want a Coke,
or a “2” if you want a Sprite.

Order: 1

Thank you!  You ordered:

- Cheeseburger

- Coke
```