ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

Lecture 1: Introduction to Python

About me

- Dr. Dinh Viet Sang
- Working room: 902 B1
- Email: sangdv@soict.hust.edu.vn
- Homepage: https://users.soict.hust.edu.vn/sangdv/

CONTENTS

Course info

Course info

Module	Evaluation method	Percent
A1. Mid-term (*)	Progress	50%
	A1.1. Mid-term exam	20%
	A1.2. Homework	10%
	A1.3. Practice	20%
A2. Final term	A2.1. Final exam	50%

(*) The evaluation about the progress can be adjusted with some bonus based on attendance, performance in class and so on.
The bonus should belong to [-2, +1], according to the policy of HUST.

Course info

Chapter 1. Introduction

Chapter 2. Control flow

Chapter 3. Functions

Chapter 4. Strings

Chapter 5. Tuples and lists

Chapter 6. Modules

Chapter 7. Files

Chapter 8. Classes and Objects

Chapter 9. Exceptions

Chapter 10. Testing and debugging

Materials

- E-Book:

Think Python 2nd Edition by Allen B. Downey

- Online courses:

MIT: Introduction to Computer Science and Programming in Python

Cornell University: CS 1110: Introduction to Computing Using Python

University of Whashington: CSE 143: Computer Programming

NOTE

- Highly encourage you
  - to bring computers to answer **in-class practice exercises!**
  - to take notes and run code files when I do

Computer and programming language

PROBLEM SOLVING

- Problem solving: the most important skill for a computer scientist
- Problem solving is the ability to formulate problems, think creatively about solutions, and express a solution clearly and accurately
- Learning to **program** is an excellent opportunity to practice problem-solving skills

WHAT DOES A COMPUTER DO

- Fundamentally:
  - performs **calculations**
    - a billion calculations per second!
  - **remembers** results
    - 100s of gigabytes of storage!
- What kinds of calculations?
  - **built-in** to the language
  - ones that **you define** as the programmer
- computers only know what you tell them

TYPES OF KNOWLEDGE

- **declarative knowledge** is **statements of fact**.
- **imperative knowledge** is a **recipe** or "how-to".

A numerical example:

- square root of a number `x` is `y` such that `y*y = x`
- recipe for deducing square root of a number `x` (16)

1) Start with a **guess**, `g`
2) If `g*g` is **close enough** to `x`, stop and say `g` is the answer
3) Otherwise make a **new guess** by averaging `g` and `x/g`
4) Using the new guess, **repeat** process until close enough

g	g*g	x/g	(g+x/g)/2
3	9	16/3	4.17
4.17	17.36	3.837	4.0035
4.0035	16.0277	3.997	4.000002

WHAT IS A RECIPE

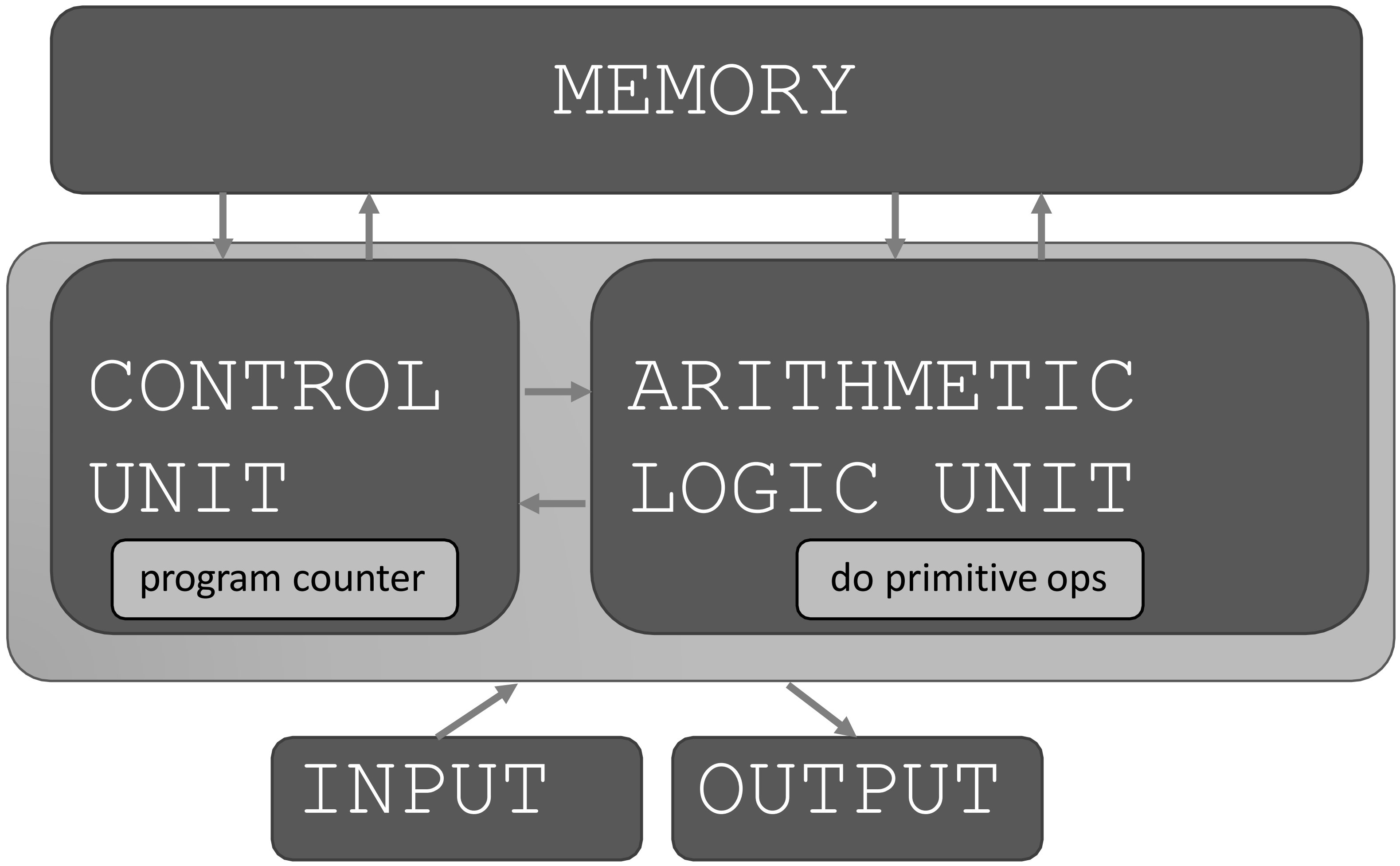
1) sequence of simple **steps**
2) **flow of control** process that specifies when each step is executed
3) a means of determining **when to stop**

- 1+2+3 = an **algorithm**!

COMPUTERS ARE MACHINES

- how to capture a recipe in a mechanical process
- **fixed program** computer
  - calculator
- **stored program** computer
  - machine stores and executes instructions

BASIC MACHINE ARCHITECTURE



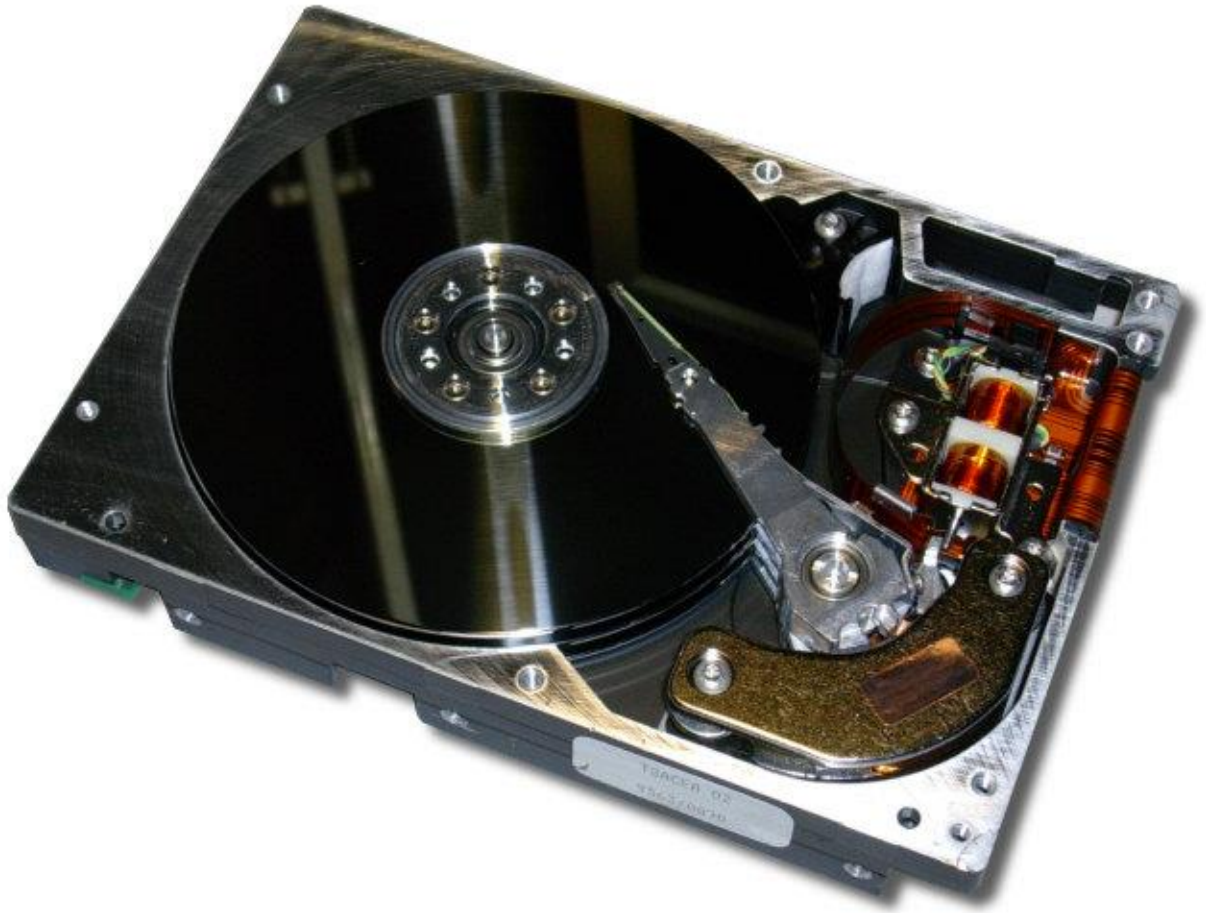
Definitions

- **Central Processing Unit:** Runs the Program - The CPU is always wondering “what to do next”? Not the brains exactly – not smart 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 Random Access Memory (RAM)
- **Secondary Memory:** Slower large permanent storage - lasts until deleted - disk drive / memory stick

Central Processing Unit (CPU)

Memory (RAM)

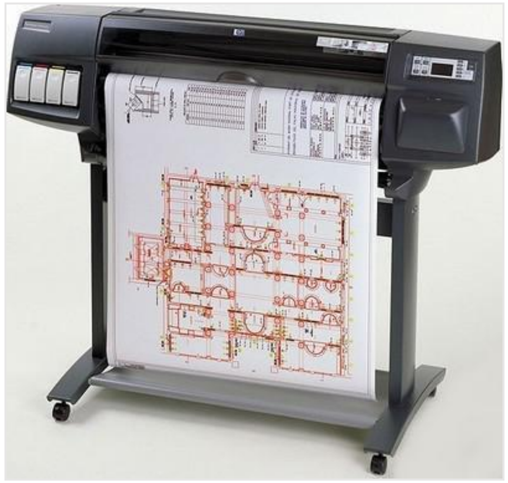
Hard Drive



Source: http://commons.Wikimedia.org

Input devices

Output devices



Program and Programming language

- **Program**: sequence of **instructions stored** inside computer
- **Programming language**: Language used for programming: Used to communicate with computers, help computers understand and perform specific tasks

Programming language

- **Machine language:**

– Machine language is a set of primitive instructions built into every computer.

– Each type of computer has its own machine language.

– The only type of language for writing a program that a computer can directly understand and execute.

– The instructions of this language are written in binary or remote code.

– Attached to the hardware architecture of the machine, thus exploiting the hardware specifications.

– Not favorable for programmers due to the difficulty of remembering the code, the lack of structure, ...

– For example, to add two numbers, you might write an instruction in binary like this:

```
1101101010011010
```

Programming language

- **Assembly language:**
  - Assembly languages were developed to make programming easy
  - Allows the programmer to use some acronyms to write instructions.
  - For example, add the contents of registers AX and BX and write the result to AX:

    Machine code (8086): `01D8`

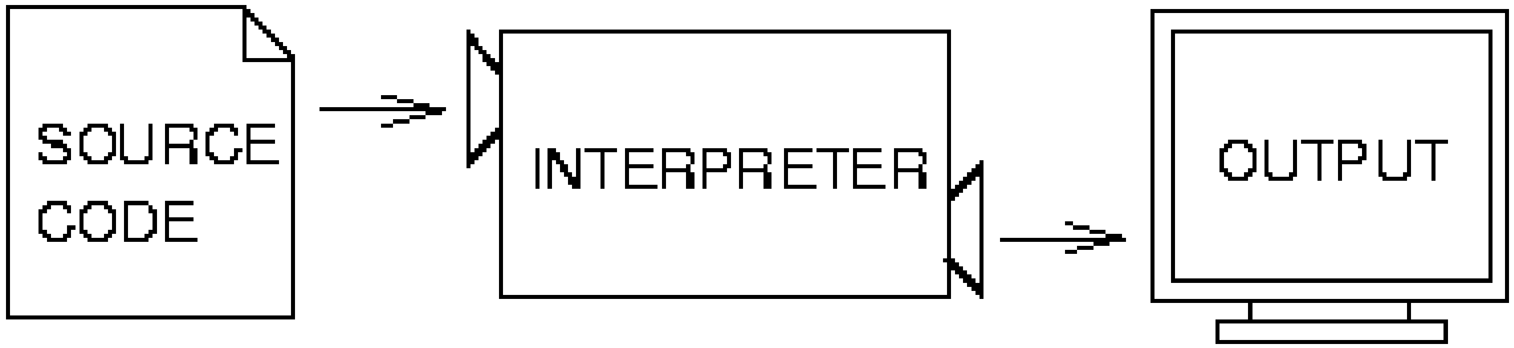
    Assembly statement: `ADD AX, BX`
  - The assembly language program must be translated into the machine language before a computer can execute it.
  - A program called assembler is used to convert assembly language programs into machine code.

Programming language

- **High-level language:**
  - The high-level languages are English-like and easy to learn and program.
  - Less dependent on computer hardware architecture
  - **Portable:** can run on different kinds of computers with few or no modifications
  - Must be translated into the machine language before a computer can execute it.
  - For example: FORTRAN, COBOL, ALGOL60, BASIC, Pascal, Foxpro, Visual Foxpro, Visual Basic, C, Visual C, C ++, Java, C#, **Python**, ...
  - **Almost all programs** are written in **high-level languages**. Low-level languages are used only for a few specialized applications.

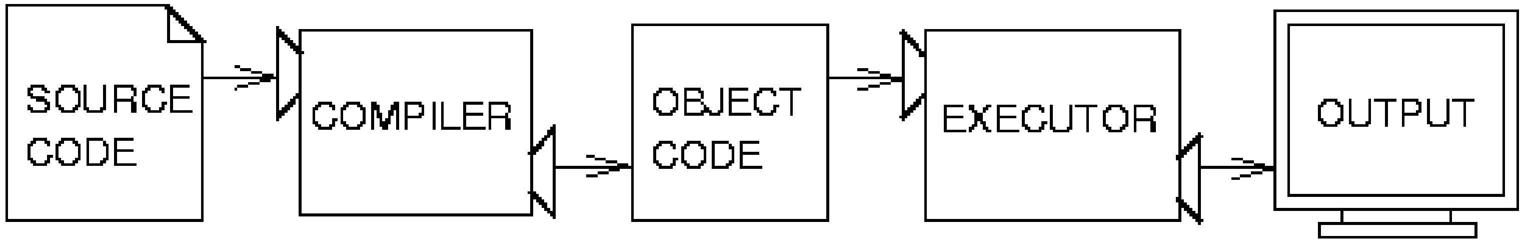
Interpreter and compiler

- Two kinds of programs process high-level languages into low-level languages: **interpreters** and **compilers**
- An **interpreter** reads a high-level program and executes it.
- It processes the program a little at a time, alternately reading lines and performing computations.



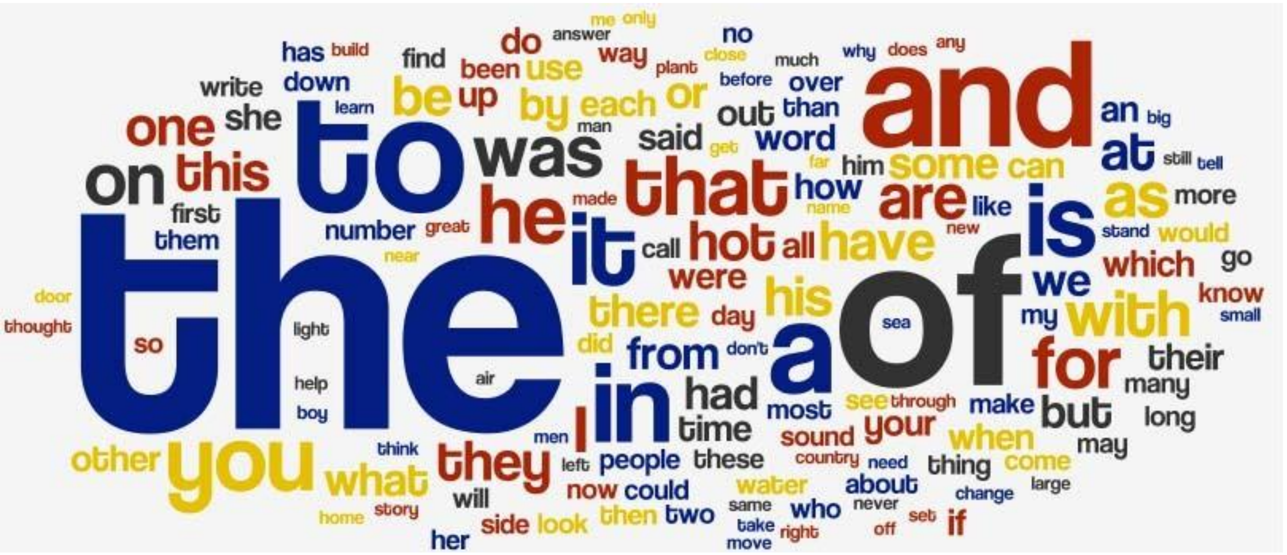
Interpreter and compiler

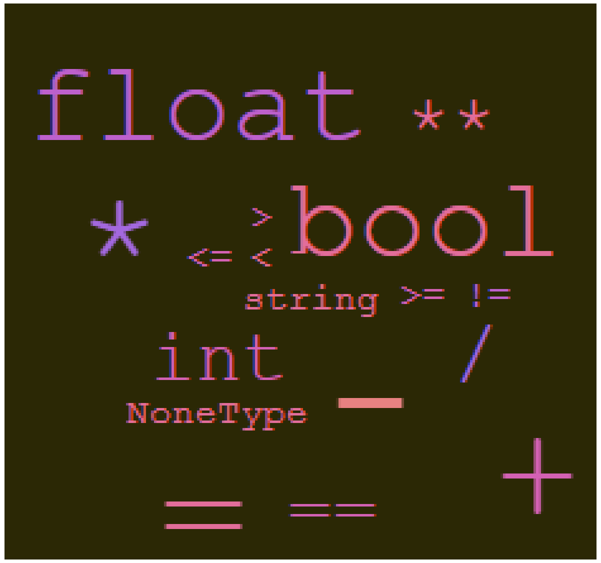
- A **compiler** reads the program and translates it completely before the program starts running.
- The high-level program is called the **source code**
- the translated program is called the **object code** or **the executable**
- The object code is often then linked with other supporting library code before the object can be executed on the machine.
- Once a program is compiled, you can execute it repeatedly without further translation



ASPECTS OF LANGUAGES

- **primitive constructs**
  - English: words
  - programming language: numbers, strings, simple operators





ASPECTS OF LANGUAGES

- **syntax**
  - English: `"cat dog boy"` → not syntactically valid
    `"cat hugs boy"` → syntactically valid
  - programming language: `"hi"5` → not syntactically valid
    `3.2*5` → syntactically valid

ASPECTS OF LANGUAGES

- **static semantics** is which syntactically valid strings have meaning
  - English: `"I are hungry"` → syntactically valid but static semantic error
  - programming language: `3.2*5` → syntactically valid
    `3+"hi"` → static semantic error

ASPECTS OF LANGUAGES

- **semantics** is the meaning associated with a syntactically correct string of symbols with no static semantic errors
  - English: can have many meanings
  - programming languages: have only one meaning but may not be what programmer intended

WHERE THINGS GO WRONG

- **syntactic errors**
  - common and easily caught
- **static semantic errors**
  - some languages check for these before running program
  - can cause unpredictable behavior
- no semantic errors but **different meaning than what programmer intended**
  - program crashes, stops running
  - program runs forever
  - program gives an answer but different than expected

What is Python?

What is Python?

- Python is a general purpose programming language. That means you can use Python to write code for any programming tasks.
- Python are now used in Google search engine, in mission critical projects in NASA, in processing financial transactions at New York 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. Data in Python are objects created from classes. A class is essentially a type that defines the objects of the same kind with properties and methods for manipulating objects. Object-oriented programming is a powerful tool for developing reusable software.

PYTHON PROGRAMS

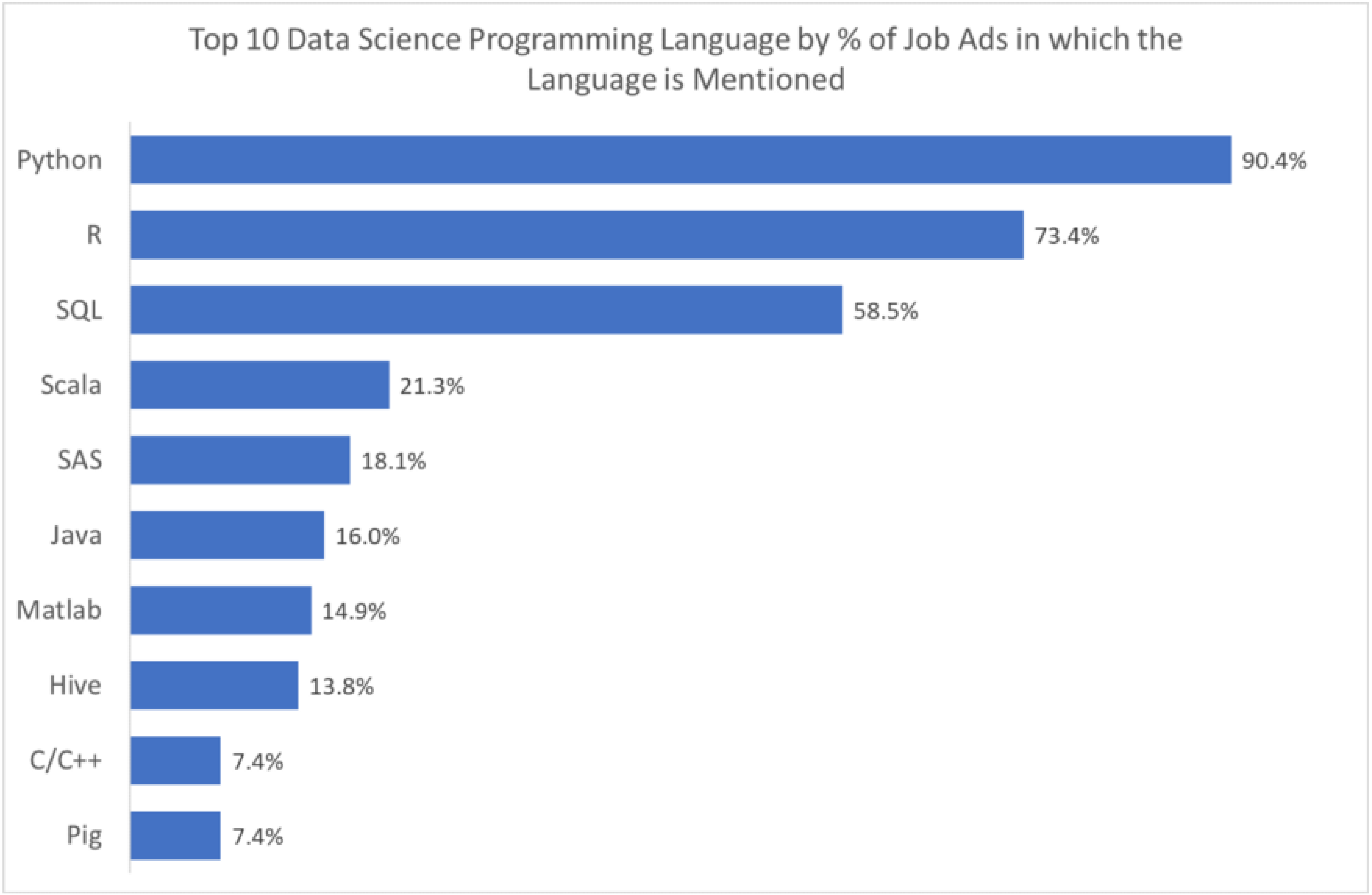
- a **program** is a sequence of definitions and commands
  - definitions **evaluated**
  - commands **executed** by Python interpreter in a shell
- **commands** (statements) instruct interpreter to do something
- can be typed directly in a **shell** or stored in a **file** that is read into the shell and evaluated

What is Python?

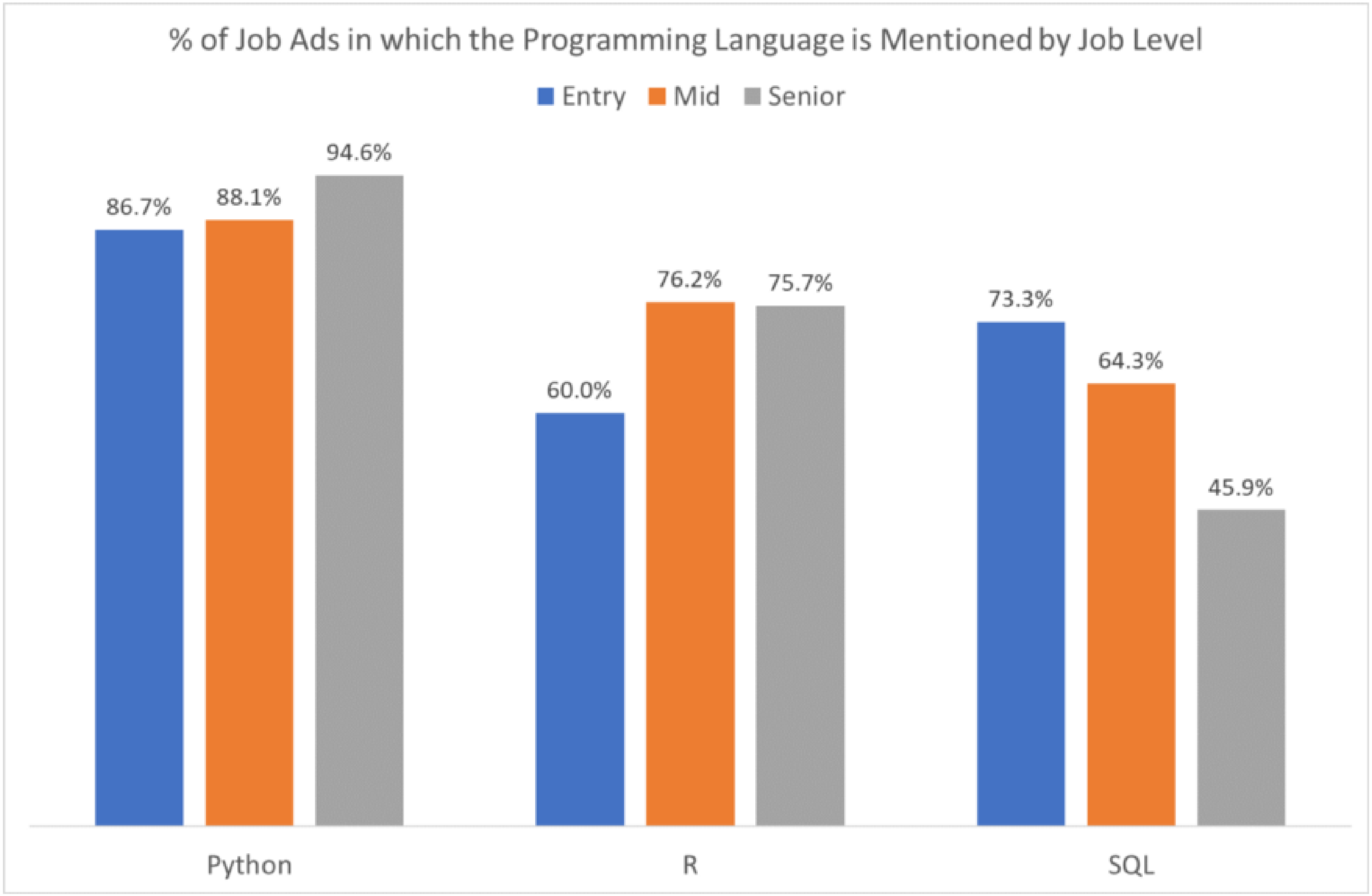
Popularity of Programming Language
Worldwide, Mar 2021
https://pypl.github.io/PYPL.html

Rank	Language	Share	Trend
1	Python	30.17 %	-0.2 %
2	Java	17.18 %	-1.2 %
3	JavaScript	8.21 %	+0.2 %
4	C#	6.76 %	-0.6 %
5	C/C++	6.71 %	+0.8 %
6	PHP	6.13 %	+0.0 %
7	R	3.81 %	+0.0 %
8	Objective-C	3.56 %	+1.1 %
9	Swift	1.82 %	-0.4 %
10	Matlab	1.8 %	-0.0 %
11	Kotlin	1.76 %	+0.2 %
12	TypeScript	1.74 %	-0.1 %
13	Go	1.34 %	+0.0 %
14	VBA	1.22 %	-0.1 %
15	Ruby	1.13 %	-0.1 %

What is Python?



What is Python?

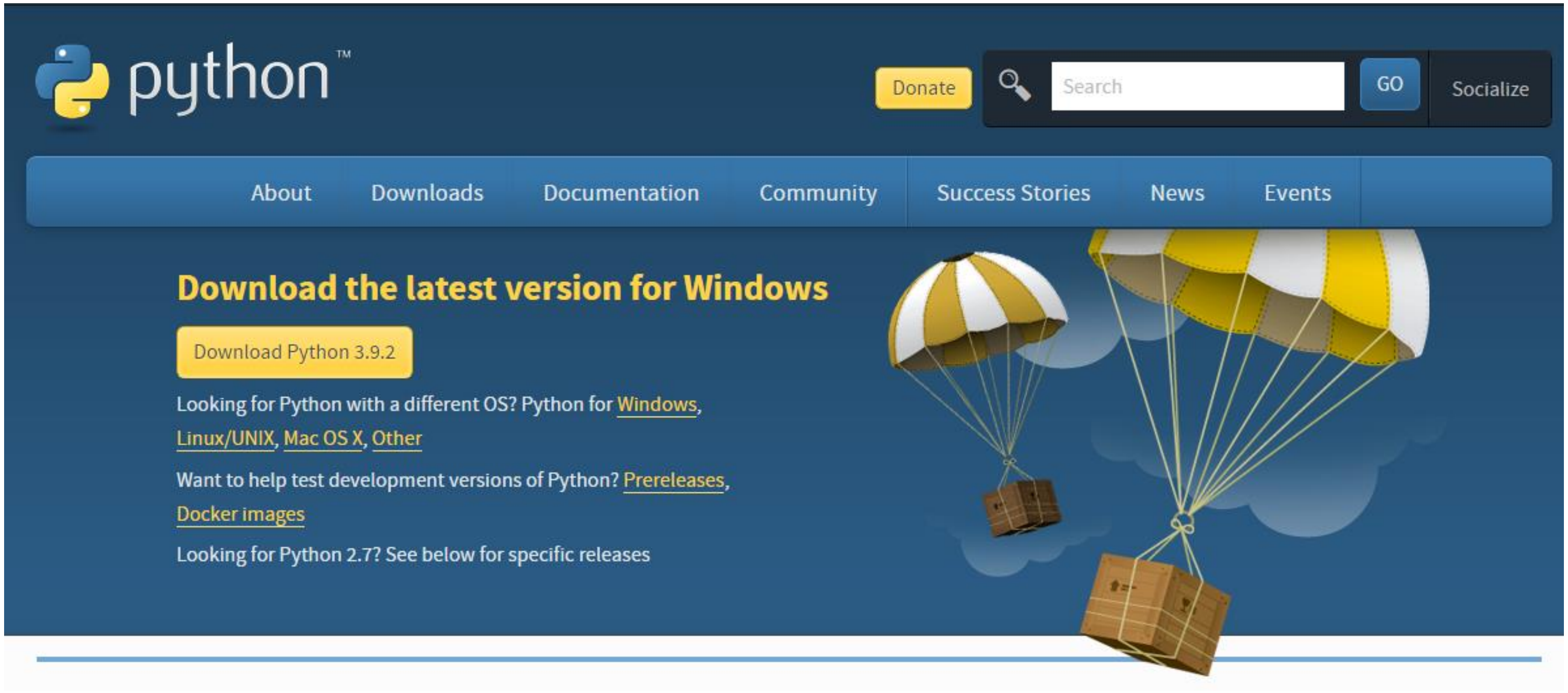


Python's History

- Created by Guido van Rossum in Netherlands in 1990
- Open source
- Python 3 is a newer version, but it is not backward compatible with Python 2. That means if you write a program using Python 2, it may not work on Python 3.

Python installation

- Direct installation: https://www.python.org/downloads/



Download the latest version for Windows

Looking for Python with a different OS? Python for Windows, Linux/UNIX, Mac OS X, Other

Want to help test development versions of Python? Prereleases, Docker images

Looking for Python 2.7? See below for specific releases

Python installation

- Anaconda: https://www.anaconda.com/products/individual
- Anaconda is a distribution of the Python and R programming languages for scientific computing (data science, machine learning…), that aims to simplify package management and deployment.
- Can create many independent environments with different package versions
- Over 250 packages automatically installed, and over 7,500 additional open-source packages
- It also includes a GUI, Anaconda Navigator, as a graphical alternative to the command line interface (CLI).

Variables, Expressions, and Statements

Constants

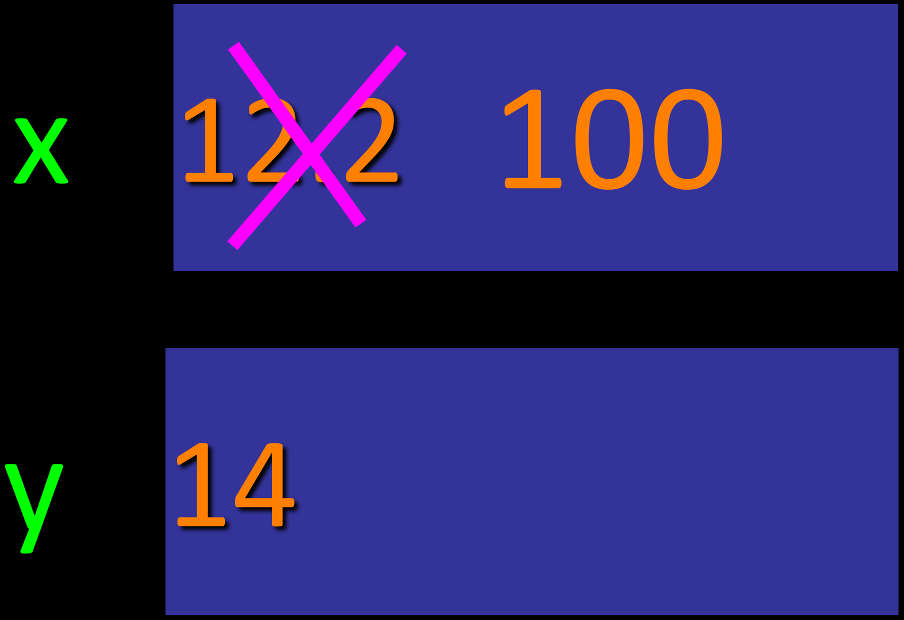
- **Fixed values** such as numbers, letters, and strings are called "**constants**" - because their value does not change
- Numeric constants are as you expect
- String constants use single-quotes (‘) or double-quotes (")

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

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
```



Identifiers/Variable Names

- Identifiers are names given to variables, functions, etc.
- An identifier is a sequence of characters that consists of letters, digits, underscores (_), and asterisk (*).
- An identifier must start with a letter or an underscore. It cannot start with a digit.
- An identifier cannot be a reserved word. Reserved words have special meanings in Python, which we will later.
- An identifier can be of any length.

Python Variable Name Rules

- Can consist of letters, numbers, or underscores (but cannot start with a number)
- Case Sensitive
- Good: spam eggs spam23 _speed
- Bad: 23spam #sign var.12
- Different: spam Spam SPAM

Reserved words (Keywords)

- You can NOT use reserved words as variable names / identifiers

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

BINDING VARIABLES AND VALUES

- equal sign is an **assignment** of a value to a variable name

variable value

```
pi = 3.14159
pi_approx = 22/7
```

  - value stored in computer memory
  - an assignment binds name to value
  - retrieve value associated with name or variable by invoking the name, by typing `pi`

ABSTRACTING EXPRESSIONS

- why **give names** to values of expressions?
- to **reuse names** instead of values
- easier to change code later

```
pi = 3.14159
radius = 2.2
area = pi*(radius**2)
```

PROGRAMMING vs MATH

- in programming, you do not "solve for x"

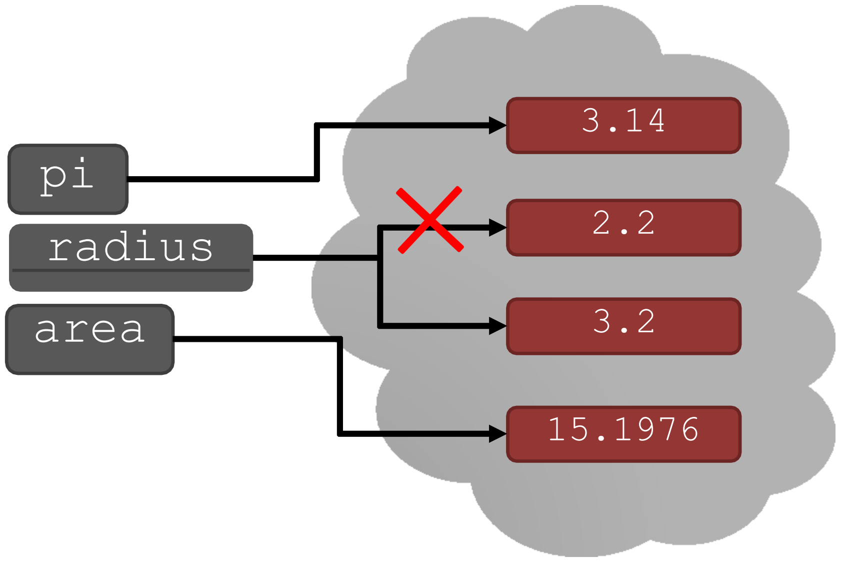
```
pi = 3.14159
radius = 2.2
# area of circle
area = pi*(radius**2)
radius = radius+1
```

an assignment
* expression on the right, evaluated to a value
* variable name on the left
* equivalent expression to `radius = radius + 1` is `radius += 1`

CHANGING BINDINGS

- can re-bind variable names using new assignment statements
- previous value may still stored in memory but lost the handle for it
- value for area does not change until you tell the computer to do the calculation again

```
pi = 3.14
radius = 2.2
area = pi*(radius**2)
radius = radius+1
```

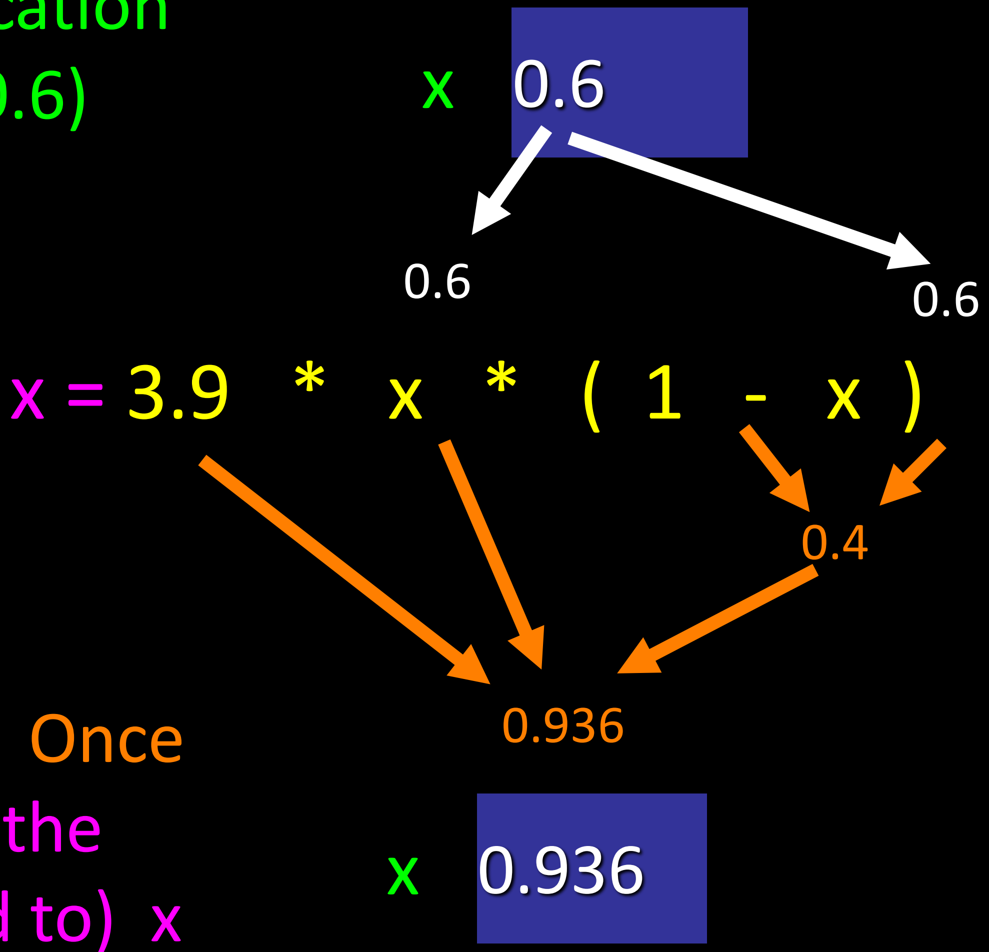


Another example: Assignment Statements

- Consider the following assignment

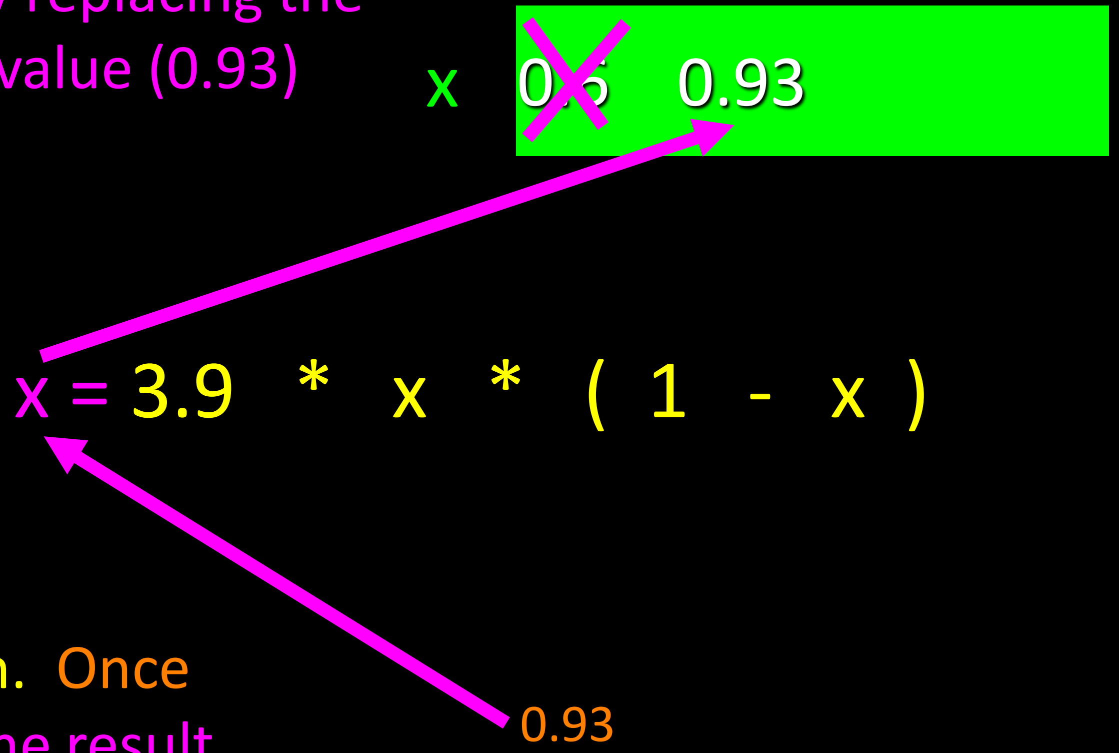
```
x = 3.9 * x * ( 1 - x )
```

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



Right side is an expression. Once 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 (0.93)



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

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

Named Constants

- The value of a variable may change during the execution of a program, but a named constant or simply constant represents permanent data that never changes.
- Python does not have a special syntax for naming constants.
- You can simply create a variable to denote a constant. To distinguish a constant from a variable, use all uppercase letters to name a constant.

```
PI = 3.14
GRAVITY = 9.8
```

Numeric Operators

- `i+j` → the **sum**
- `i-j` → the **difference**
- `i*j` → the **product**
- `i/j` → **division**

if both are ints, result is int
if either or both are floats, result is float

result is float

- `i%j` → the **remainder** when `i` is divided by `j`
- `i**j` → `i` to the **power** of `j`

Numeric Operators

Name	Meaning	Example	Result
+	Addition	34 + 1	35
-	Subtraction	34.0 - 0.1	33.9
*	Multiplication	300 * 30	9000
/	Float Division	1 / 2	0.5
//	Integer Division	1 // 2	0
**	Exponentiation	4 ** 0.5	2.0
%	Remainder	20 % 3	2

Integer Division

Python 2

- Integer division truncates (floor division)
- Floating point division produces floating point numbers (true division)

```
>>> print (10 / 2)
5
>>> print (9 / 2)
4
>>> print (99 / 100)
0
>>> print (10.0 / 2.0)
5.0
>>> print (99.0 / 100.0)
0.99
```

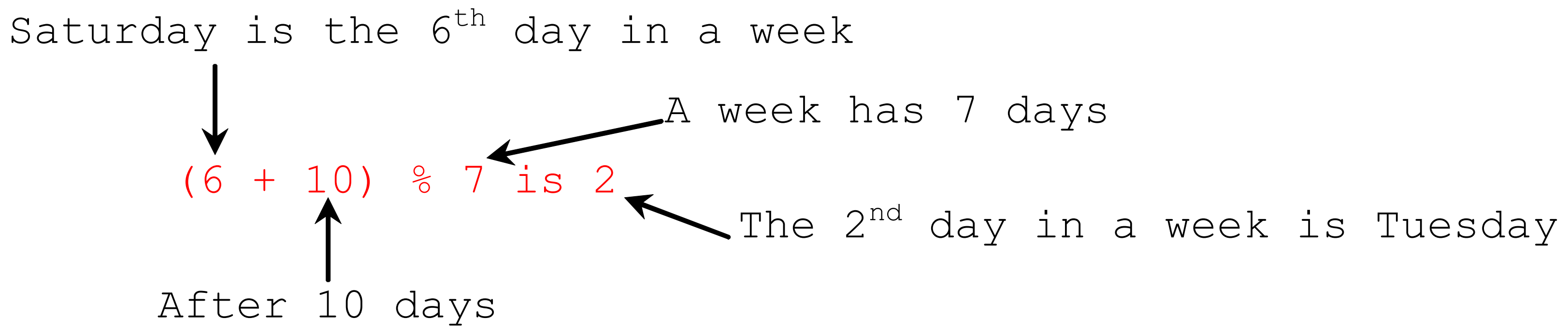
Python 3

“/” does true division for all types

“//” does floor division for integers

Remainder Operator

- Remainder is very useful in programming. For example, an even number % 2 is always 0 and an odd number % 2 is always 1. So you can use this property to determine whether a number is even or odd. Suppose today is Saturday and you and your friends are going to meet in 10 days. What day is in 10 days? You can find that day is Tuesday using the following expression:


Saturday is the 6th day in a week
A week has 7 days
(6 + 10) % 7 is 2
The 2nd day in a week is Tuesday
After 10 days

Overflow

- When a variable is assigned a value that is too large (in size) to be stored, it causes overflow. For example, executing the following statement causes overflow.

```
>>>245.0 ** 1000

OverflowError: 'Result too large'
```

Underflow

- When a floating-point number is too small (i.e., too close to zero) to be stored, it causes underflow. Python approximates it to zero. So normally you should not be concerned with underflow.

Scientific Notation

- Floating-point literals can also be specified in scientific notation, for example,
- 1.23456e+2, same as 1.23456e2, is equivalent to 123.456, and
- 1.23456e-2 is equivalent to 0.0123456.
- E (or e) represents an exponent and it can be either in lowercase or uppercase.

Arithmetic Expressions

$$\frac{3+4x}{5}-\frac{10(y-5)(a+b+c)}{x}+9(\frac{4}{x}+\frac{9+x}{y})$$

is translated to

(3+4*x)/5 – 10*(y-5)*(a+b+c)/x + 9*(4/x + (9+x)/y)

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

Operator Precedence Rules

- Highest precedence rule to lowest precedence rule
- Parenthesis 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
```

Parenthesis
Power
Multiplication
Addition
Left to Right

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

$$\begin{tikzcd}
1 + \boxed{2 ** 3} / 4 * 5 \arrow[d] \\
1 + \boxed{8 / 4} * 5 \arrow[d] \\
1 + \boxed{2 * 5} \arrow[d] \\
\boxed{1 + 10} \arrow[d] \\
11
\end{tikzcd}$$

Note 8/4 goes before 4*5 because of the left-right rule.

Parenthesis
Power
Multiplication
Addition
Left to Right

Operator Precedence

- Remember the rules -- top to bottom
- When writing code - use parenthesis
- When writing code - keep mathematical expressions simple enough that they are easy to understand
- Break long series of mathematical operations up to make them more clear

Parenthesis
Power
Multiplication
Addition
Left to Right

Exam Question: x = 1 + 2 * 3 - 4 / 5

Mixing Integer and Floating

- When you perform an operation where one operand is an integer and the other operand is a floating point the result is a floating point
- The integer is converted to a floating point before the operation

```
>>> print (1 + 2 * 3 / 4.0 – 5)
-2.5
>>>
```

Augmented Assignment Operators

Operator	*Example*	*Equivalent*
`+=`	`i += 8`	`i = i + 8`
`-=`	`f -= 8.0`	`f = f - 8.0`
`*=`	`i *= 8`	`i = i * 8`
`/=`	`i /= 8`	`i = i / 8`
`%=`	`i %= 8`	`i = i % 8`

Data types

- In Python variables, literals, and constants have a "data type"
- In Python variables are "**dynamically**" typed. In some other languages you have to explicitly declare the type before you use the variable

In C/C++:

```
int a;
float b;
a = 5
b = 0.43
```

In Python:

```
a = 5
a = "Hello"
a = [ 5, 2, 1]
```

More on “Types”

- 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

```
>>> d = 1 + 4
>>> print (d)
5

>>> e = 'hello ' + 'there'
>>> print (e)
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.

```
>>> e = 'hello ' + 'there'
>>> e = e + 1
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in
<module>
    e = e + 1
TypeError: can only concatenate str
(not "int") to str
>>> type(e)
<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

- can **convert object of one type to another**
- 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)  – 5)
2.0
>>>
```

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 "<pyshell#10>", line 1, in <module>
    print (sval + 1)
TypeError: can only concatenate str (not "int") to 
str
>>> ival = int(sval)
>>> type(ival)
<class 'int'>
>>> print (ival + 1)
124
>>> nsv = 'hello bob'
>>> niv = int(nsv)
Traceback (most recent call last):
  File "<pyshell#9>", line 1, in <module>
    niv = int(nsv)
ValueError: invalid literal for int() with base 10: 
'hello bob'
```

User Input

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

```
name = input(‘Who are you?’)
print ('Welcome ', name)
```

```
Who are you? Anna
Welcome Anna
```

Converting User Input

- If we want to read a number from the user, we must convert it from a string to a number using a type conversion function
- Later we will deal with bad input data

```
inp = input('Europe floor?')
usf = int(inp) + 1
print ("US floor: ", usf)
```

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

Comments in Python

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

```
# Get the name of the file and open it
name = input("Enter file:")
handle = open(name, "r")   # This is a file handle
text = handle.read()
words = text.split()
```

String Operations

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

```
>>> print ('abc' + '123‘)
abc123

>>> print ('Hi' * 5)
HiHiHiHiHi
```

Mnemonic Variable Names

- Since we programmers are given a choice in how we choose our variable names, there is a bit of “best practice”
- We name variables to help us remember what we intend to store in them (“mnemonic” = “memory aid”)
- This can confuse beginning students because well named variables often “sound” so good that they must be keywords

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

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

What is this code doing?

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

Exercise 1

Whenever you are experimenting with a new feature, you should try to make mistakes. This kind of experiment helps you remember what you read; it also helps when you are programming, because you get to know what the error messages mean.

1. In a print statement, what happens if you leave out one of the parentheses, or both?
2. If you are trying to print a string, what happens if you leave out one of the quotation marks, or both?
3. You can use a minus sign to make a negative number like -2. What happens if you put a plus sign before a number? What about 2++2?
4. In math notation, leading zeros are ok, as in 09. What happens if you try this in Python? What about 011?
5. What happens if you have two values with no operator between them?

Exercise 2

Start the Python interpreter and use it as a calculator.

1. How many seconds are there in 42 minutes 42 seconds?
2. How many miles are there in 10 kilometers? Hint: there are 1.61 kilometers in a mile.
3. If you run a 10 kilometer race in 42 minutes 42 seconds, what is your average pace (time per mile in minutes and seconds)? What is your average speed in miles per hour?

Exercise 3

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

- Enter Hours: 35
- Enter Rate: 2.75
- Pay: 96.25

Exercise 4

- Rewrite the following figure in Python using print statement

```
  ______
 /      \
/        \
\        /
 \______/
\        /
 \______/
+--------+
 /      \
/        \
|  STOP  |
\        /
 \______/
 /      \
/        \
+--------+
```