Extended Introduction to Computer Science
CS1001.py

Chapter A First Acquaintance with Python

Amir Rubinstein, Michal Kleinbort

School of Computer Science
Tel-Aviv University

Fall Semester 2023-4
http://tau-cs1001-py.wikidot.com

* Slides based on a course designed by Prof. Benny Chor

Lecture Goals

- Acquaintance with IDLE
  - Our programming environment for this semester

- First acquaintance with Python
  - Our programming language for this semester
    - Types of data: str, int, float
    - Basic operations
    - Variables
    - Conditionals

Programming Languages Basics

- Informally, a computer program is a text containing a sequence of instructions that can be "understood" and executed by a computer
  - a more formal definition please?
  - wait for the computational models course (2nd yr)

- In some sense, a computer program resembles a recipe.

  - Pots, ovens, and even the final consumer of food, are typically quite tolerant. Putting a bit more sugar or a little less nutmeg will hardly be felt.

  - By way of contrast, an extra parenthesis, or a missing colon or quotation marks, will most likely cause a program to crash.

Online Demo: ID Control Digit

- Let’s start right off with an example: computing the control digit of an Israeli ID number.
  - You are NOT expected to understand the code right now, but you will be able to, within a week!
  - We may encounter this example again towards the end of the course (error detection codes)

Writing Programs

- Getting a program to work as planned is an interesting, and often challenging process. It can be frustrating as well.

- Planning what your program should do, and how it is going to do it, is crucial.

  - It is very tempting to skip such planning and go straight to writing lines of code.

  - When things go wrong, it is even more tempting to change a line of the code and hope this will solve the problem.

  - We strongly advise you not to skip the planning stage (both before and during the process).

High Level → Machine Code

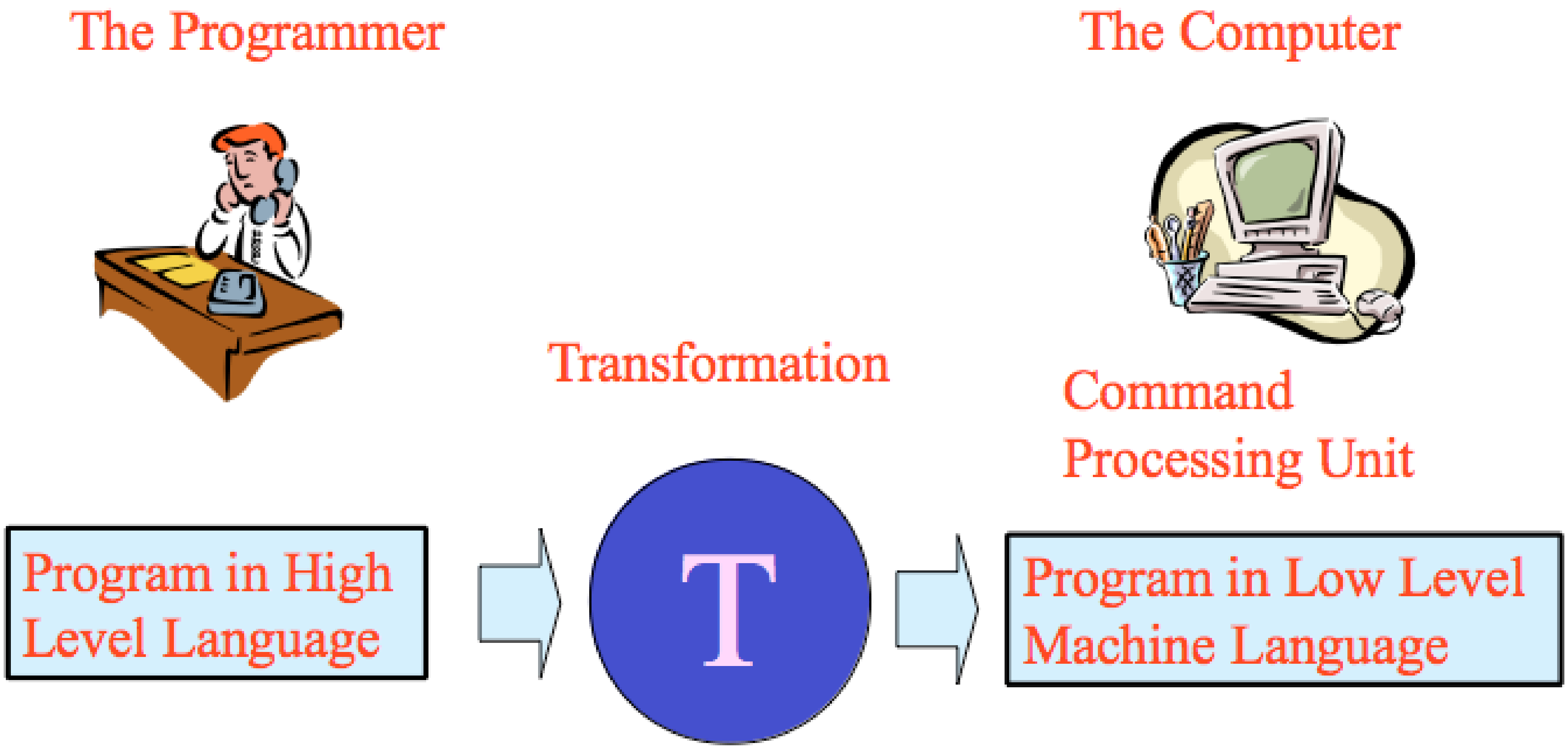
- Most programs these days are written in high level programming languages (Python, C, Java, C++, Fortran, R, and many many more)

- These are formal languages with a strict syntax, yet are fairly comprehensible to experienced programmers.

- By way of contrast, the computer hardware “understands” a lower level machine code.

- The high level language needs to be transformed to the machine code (by yet another computer program).

High Level → Machine Code



The Programmer

The Computer

Transformation

Command Processing Unit

Program in High Level Language

T

Program in Low Level Machine Language

(figure taken from TAU's old intro to CS in Scheme course)

High Level → Machine Code

- The transformation from high level to machine level languages comes in two flavors:

  - by interpreters (as in Python) and
  - by compilers (as in C)

- More details in the appendix

IDLE

- An integrated development environment (IDE) is a software that provides facilities to computer programmers:
  - writing programs
  - executing programs
  - debugging programs

- Python has various IDEs: IDLE, PyCharm, Eclipse, Notepad++,…

- We will use IDLE, one of the simplest programming environments for Python, suitable for beginners.
  - For large industrial projects, IDLE may be too simple. But it is completely adequate for the rather simple programs we (and you) will write in this course.

IDLE

- There are two Python versions: 2 and 3. We use 3
  - Python 3 is not fully compatible with Python 2.
  - If you use Python 2, your programs will most likely crash in our HW execution tests. This will have negative effects on the “wet” part of your homework assignments’ grades, so is best avoided.

- Go to www.python.org
  - Download the latest Python 3 interpreter
  - More instructions on the course website

Interactive (shell) mode vs. Script mode

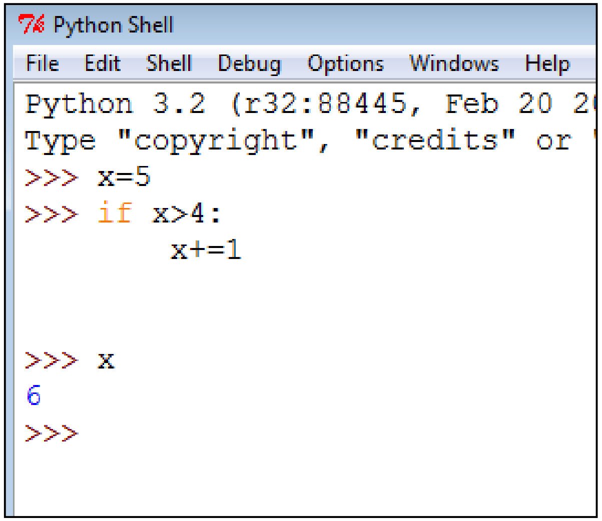
- When we open IDLE we get the shell mode,

  also called Interactive mode .

  This is a "ping-pong" mode.

  we can run a single command at a time.

  Very convenient for short computations.



```
Python 3.2 (r32:88445, Feb 20 2
Type "copyright", "credits" or '
>>> x=5
>>> if x>4:
        x+=1


>>> x
6
>>>
```

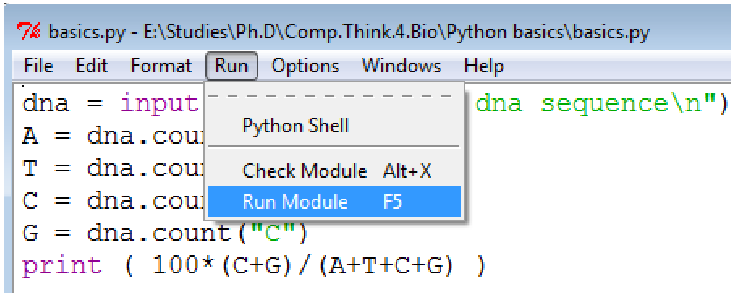
- Script mode enables writing the whole program first, saving it in a .py file, and only then running it line by line.

To work in script mode:

1a. File → New File OR

1b. right click on an existing .py file → edit

2. Run → Run Module (or F5)


basics.py - E:\Studies\Ph.D\Comp.Think.4.Bio\Python basics\basics.py
File Edit Format Run Options Windows Help
Python Shell
Check Module Alt+X
Run Module F5

```
dna = input                          dna sequence\n")
A = dna.cou
T = dna.cou
C = dna.cou
G = dna.count("C")
print ( 100*(C+G)/(A+T+C+G) )
```

Also see:
https://www.tutorialsteacher.com/python/python-idle

Comic Relief *

The first few classes introduce Python. Some of you will probably feel like this:

Or, desirably, like this:

* אני מזמין אתכם לשלוח לי הצעות לתמונות שיופיעו על שקפים אלו לאורך הסמסטר

Python Programming Basics: "Gidday, Mate"

- The first line of code taught in all programming languages is a print command of a greeting. We do not dare to deviate from this inspirational tradition, but will add an Aussie touch to it.

```
>>> print("Gidday, mate!")
Gidday, mate!
```

- The text to the right of the prompt, >>>, is the "command" to the Python interpreter. The text in the next line is the result of the command

- print is a built-in Python function (colored purple by IDLE). We will learn about functions later
- The text to be printed appears between " ", colored green.
- We will also see later that Python has a collection of reserved words, with fixed meaning, usually displayed in orange

Read, eval, print

An interaction with the interpreter has 3 steps

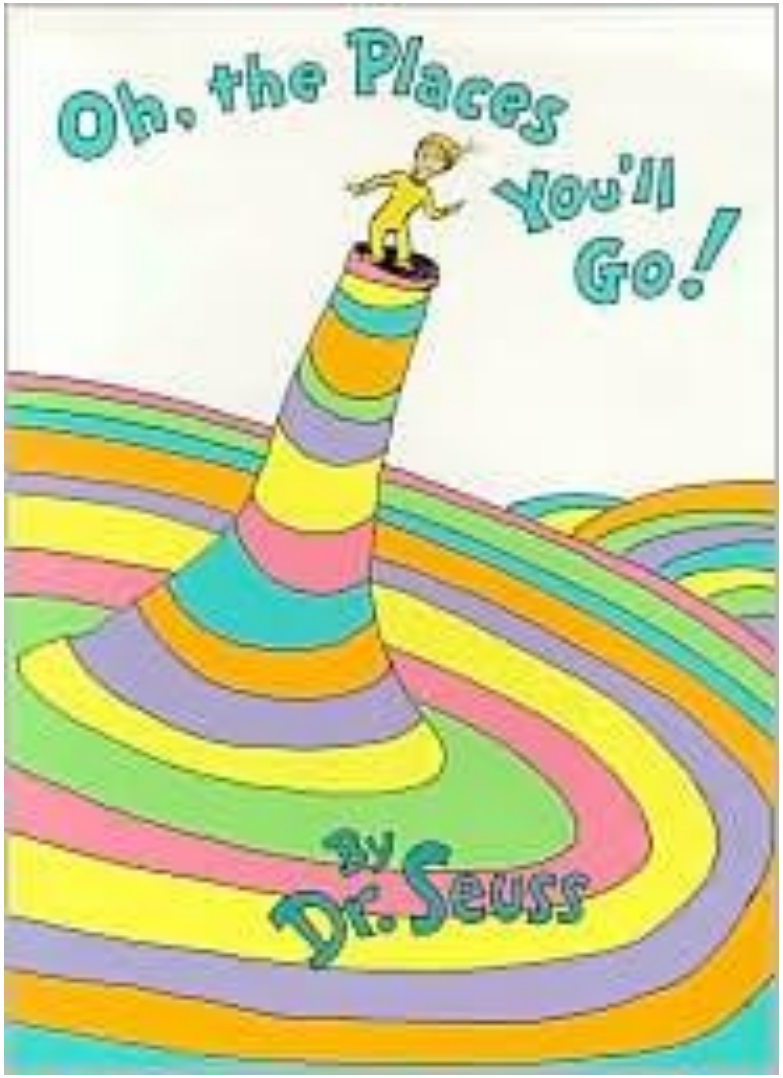
- **Read**: the interpreter reads the sequence of characters we type following the prompt (it converts this sequence to some internal form)

- **Eval**: the interpreter evaluates (computes) the code that was read, and produces a result (and perhaps additional effects)

- **Print**: the interpreter prints the result as a sequence of characters (converts internal form to text), then prints the prompt for the next interaction.

You Will Get Stuck!

I'm sorry to say so
but, sadly, it's true
that Bang-ups
and Hang-ups
can happen to you.
You can get all hung up
in a prickle-ly perch.
And your gang will fly on.
You'll be left in a Lurch.
You'll come down from the Lurch
with an unpleasant bump.
And the chances are, then,
that you'll be in a Slump.
And when you're in a Slump,
you're not in for much fun.
Un-slumping yourself
is not easily done.



What to Do When You Get Stuck?

1) Python interpreter has built-in help for all built-in and library functions/methods/classes (see next slide)

Admittedly, help response may be somewhat cryptic at times.

2) Check Python documentation at http://docs.python.org/py3k/.

3) Use your favorite search engine. With high probability, any problem you ran into was already tackled by someone who documented the solution on the web.

4) The course forum may come in handy.

Help example

```
>>> help(print)
Help on built-in function print in module builtins:

print(...)
    print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

    Prints the values to a stream, or to sys.stdout by default.
    Optional keyword arguments:
    file:  a file-like object (stream); defaults to the current sys.stdout.
    sep:   string inserted between values, default a space.
    end:   string appended after the last value, default a newline.
    flush: whether to forcibly flush the stream.
```

Python Programming Basics: Strings and Type ‘str’

```
>>> print("Gidday, mate!")
Gidday, mate!
```

In IDLE’s interactive mode we can omit the print command:

```
>>> "Gidday, mate!"
’Gidday, mate!’
```

The interpreter ‘‘response’’ -- prints the value of the last command.

We now ask for the type of "Gidday, mate!"

```
>>> type("Gidday, mate!")
<class ’str’>
```

It is of type str, indicating this is a string. In computer science, a sequence of characters, enclosed by single or double quotes, is termed a string. Strings are colored green by IDLE.

Examples of String Operations (1)

Strings have many built-in methods, like converting to lower (or upper) case, replacing a substring by another, concatenation, etc. Some of these methods' names have str. as their prefix, indicating they operate on the class “string”.

```
>>> str.upper("Michal")
'MICHAL'
>>> str.lower("Amir")
'amir'
>>> str.replace("breaking news", "breaking", "fake")
’fake news’
>>> str.upper(str.replace("breaking news", "breaking", "fake"))
'FAKE NEWS'
```

Examples of String Operations (2)

```
>>> "Py" + "thon"
’Python’
```

+ denotes concatenation

```
>>> "na " + "nach " + "nachman " + "nachman meUman "
’na nach nachman nachman meUman ’
```

```
>>> "Trump" + ""
'Trump'
```

"" is the empty string

```
>>> "Trump" + " "
'Trump '
```

a space

Examples of String Operations (3)

```
>>> "Bakbuk Bli Pkak " * 4
'Bakbuk Bli Pkak Bakbuk Bli Pkak Bakbuk Bli Pkak Bakbuk
  Bli Pkak '
```

* denotes repetition

```
>>> "Bakbuk Bli Pkak " * 0
????
>>> "Bakbuk Bli Pkak " * -3
????
>>> "Bakbuk Bli Pkak " * 2.7
????
```

error? empty string?

There are obviously many other string methods, but for the time being, these will do.

Numerical Types and Operations

```
>>> 4
4
>>> type(4)
<class 'int '>                                   integer type
>>> 3.141592653589793238462643383279502 88
3.141592653589793                                ouch ! truncated ...
>>> type(3.14159265358979323846264338327950288)
<class 'float '>                                 floating point type representing “reals”

>>> 8/5
1.6                                               / returns a float, the result of division
>>> 8//5
1                                                 // returns an integer, the floor of division
>>> 8%5
3                                                 % returns an integer, the remainder of division

>>> type(8/5)
<class 'float '>
>>> type(8//5)
<class 'int '>
```

Addition, subtraction, multiplication exist as well (mix, match, try!).

Numerical Types and Operations (2)

In arithmetic operations mixing integers and floating point numbers, the result is typically a floating point number (changing the type this way is termed coersion or casting).

```
>>> 4+3.14
7.1400000000000001
>>> 4/3.14
1.2738853503184713
>>> 4*3.14
12.56
>>> 3.14**4
97.21171216000002
>>> 3.14*0
0.0
>>> 3.14/0
Traceback (most recent call last):
File "<pyshell#8>", line 1, in <module>
3.14/0
ZeroDivisionError: float division by zero
```

** is exponentiation

Operator precedence and associativity

- An expression may include more than one operator
- The order of evaluation depends on operator’s precedence and associativity:
  - Higher precedence operators are evaluated before lower precedence operators.
  - The order of evaluation of equal precedence operators depends on their associativity.
- Parentheses override this default ordering.
  - No need to know/remember the details
  - When in doubt, use parentheses!

Operator precedence and associativity: examples

```
>>> 20-4*3           * before – (higher precedence)
8
>>> 20-(4*3)         equivalent to the previous one
8
>>> (20-4)*3         Parentheses can change the order
48
>>> 3*5//2           these equal precedence ops are evaluated
7                    from left to right (left associative)

>>> 3*(5//2)         Parentheses can change the order
6
>>> 2**3**2          ** right->left (unlike most other ops)
512
>>> (2**3)**2        Parentheses can change the order
64
```

Error Messages

```
>>> 3*"3"
'333'

>>> 3+"3"
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    3+"3"
TypeError: unsupported operand type(s) for +: 'int' and 'str'

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

- Note that the error message gives you some information on the source of the error. Get used to reading (and understanding) those errors.

Variables and Assignments

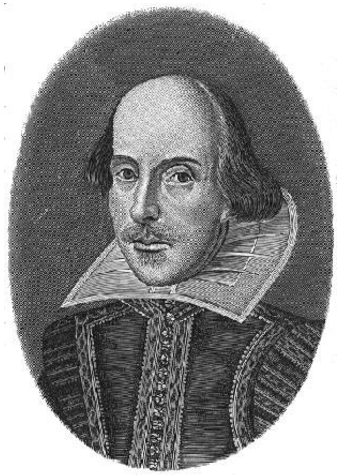
- The following line is an assignment in Python. The left hand side is a variable, used to store values for future reference. The right hand side is an expression.

```
>>> n = 10
```

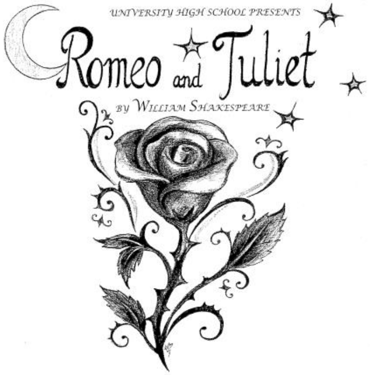
- The interpreter evaluates the expression and assigns its value to the variable.
- Think of this assignment as creating a new “mapping“, where the variable’s name, n, becomes bound to the value 10.
- In python, a variable is just a name.
  - The variable’s name is a sequence of letters, digits and _ (underscore)
  - The name must not start with a digit.
  - Names are case sensitive: for example, `grade` and `Grade` differ.

The importance of names

What's in a name? that which we call a rose
By any other name would smell as sweet;



Shakespeare/Romeo and Juliet
ACT II, SCENE II



But In programming, names are important:
Programs should be readable by other programmers.

(reference to Romeo and Juliet due to John Guttag)

Variables and Assignments: An Example

```
>>> n = 10
>>> print(n)
10
```

The value can be changed by a subsequent assignment:

```
>>> n = 11
>>> print(n)
11
```

In python the type of a variable is dynamic: it can change by subsequent assignments:

```
>>> type(n)
<class 'int'>
>>> n = 1.3141
>>> print(n)
1.3141
>>> type(n)
<class 'float'>
```

More Variables and Assignments

- Variables with assigned values can be used as part of the evaluation of other expressions:

```
>>> a = 1
>>> b = 2
>>> print(a+b)
3

>>> c = 2*a - b
>>> print(c)
0

>>> c = c+1
>>> print(c)
1

>>> print(c+d)
NameError: name 'd' is not defined
```

A Convenient Shorthand

- Consider the following sequence of instructions:

```
>>> a = 0
>>> a = a+6
>>> a
6
```

- Now suppose that, following the advice given by the course staff to give meaningful names to variables, you rewrite the code, using a more meaningful, albeit longer, name:

```
>>> votes_Biden = 0
>>> votes_Biden = votes_Biden + 6000
>>> votes_Biden
6000
```

- Python provides a shorthand for the addition, which may appeal to the young and impatient…:

```
>>> votes_Biden = 0
>>> votes_Biden += 6000
>>> votes_Biden
6000
```

A Convenient Shorthand (2)

- This shorthand is applicable to any assignment where a variable appears on both the right and left hand sides.

```
>>> x = 0
>>> x+=10
>>> x
10
>>> x*=4
>>> x
40
>>> x**=2
>>> x
1600
>>> x**=0.5
>>> x
40.0
>>> title = "Dr."
>>> title += " Strangelove"
>>> title
'Dr. Strangelove'
```

- Use with some caution: the shorthand is not always equivalent to the original expression (more in the "Tirgul").

Conditionals (if, else)

```
temp = 30 # degrees centigrade
wind = 17 # knots ( nautical miles per hour )

if temp>25 and wind>13:
     print("go windsurfing ")
else:
     if temp>25 and wind<=13:
        print("go to the beach ")
     else:
        if temp>30:
            print("put your hat on")
        else:
            print("attend class ")
```

Output:

```
Go windsurfing
```

Lecture 1: Highlights

- High level programs are transformed into machine language. For Python this is done by an interpreter.
- IDLE is one such IDE for Python, which we recommend for this course
- Values in Python belong to types (a.k.a classes). We saw str, int, float (more later)
  - Strings are enclosed within " "
  - Integers (from latin: “whole”): …,-3,-2,-1,0,1,2,3,…
  - Numbers of class ’float’ represent real numbers, often approximating the full (infinite precision) value
- Different types enable different operations, including some (but not every) “mixing”
- Assignments to variables are used to store values in the memory for later use
  - Subsequent assignments to the same variable can change its value and even its type (types in Python are dynamic)
- It’s important to read error messages that indicate the source of the “problem”
- Conditional statements allow branching in the program’s flow

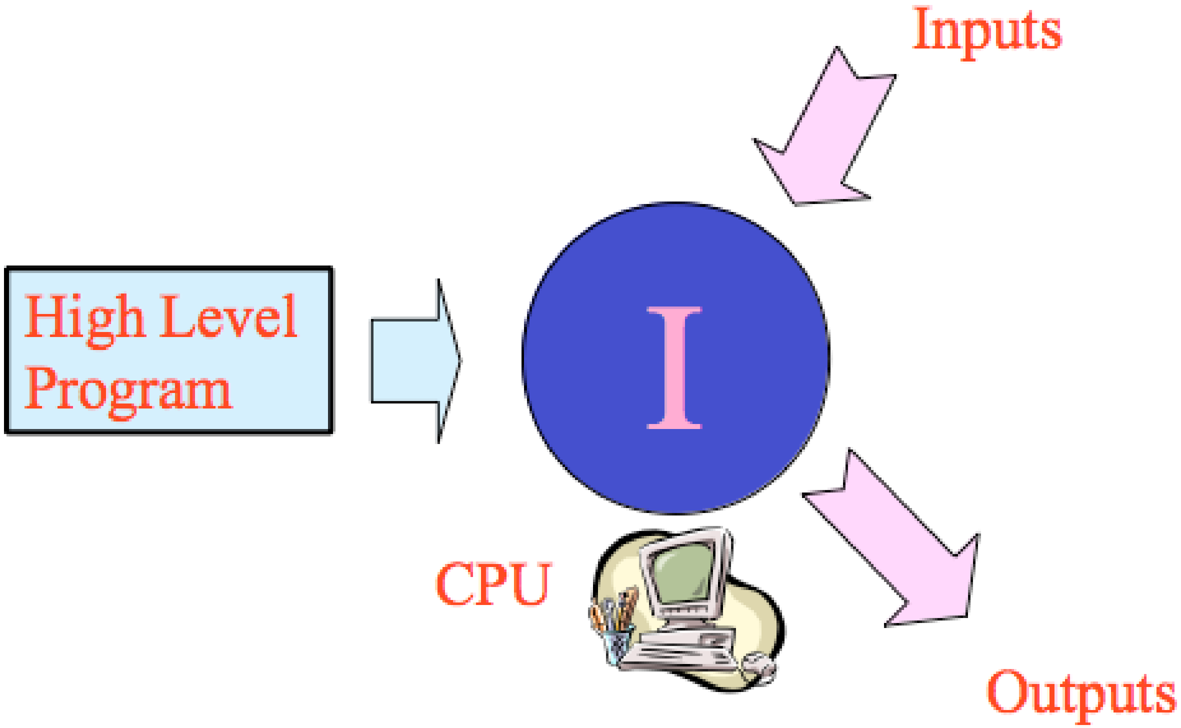
Appendix

From High Level to Machine Level

- The transformation from high level to machine level languages comes in two flavors:
  - By interpreters (as in Python) and
  - by compilers.
- A brief description follows.

The Interpreter

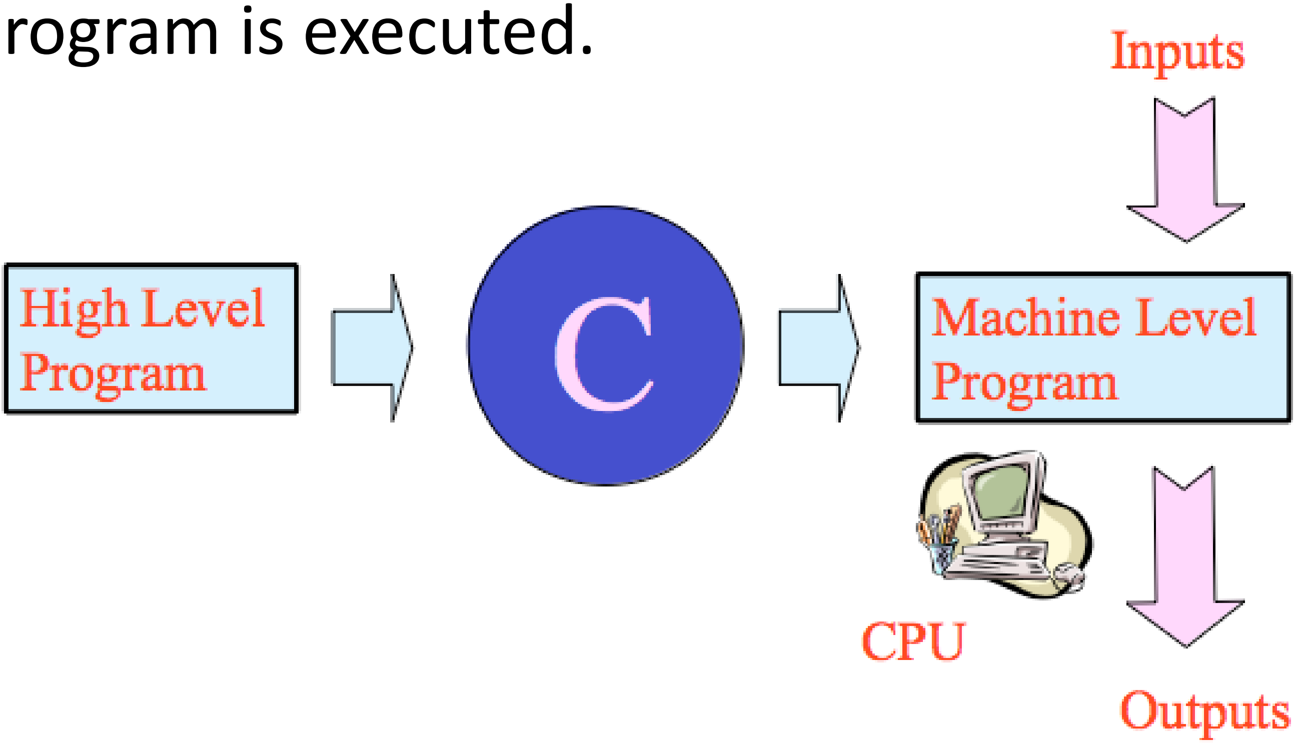
The interpreter translates and executes the high level program, line by line.



(figure taken from the old intro to CS Scheme course)

The Compiler

The compiler first translates the complete high level program to a machine level program. only then the program is executed.



(figure taken from the Scheme course site)

Specific Programming Language

- Python is an interpreted programming language.
  - So are JavaScript, Lisp (and its variant, Scheme), MATLAB, Perl, PHP, Ruby, and many many other programming languages.
- In contrast, C is a compiled programming language.
  - So are C, C++, Fortran, Haskell, Pascal, and many many other programming languages (more precisely, Java is compiled to “bytecode”, which is then interpreted)

Compiled vs. Interpreted Programming Languages

- The difference between a compiler and an interpreter usually reflects language difference.
- A compiler is useful if the language allows checking certain properties of the program before running it.
- The main difference in this respect is between languages with static types and those with dynamic types
- Python has dynamic types. The meaning of this will be understood later today.
- It is believed that dynamic types give the programmer more flexibility, while static types provide more safety, because certain errors may be detected before running the program.