TEL AVIV UNIVERSITY אוניברסיטת תל אביב

Programming for Engineers in Python

Fall 2018

Lecture 1: Introduction to Python

Programming for Engineers in Python

Welcome to the course!

- We will learn to program in **Python**.
- **Goal**: enable you to use **programming as a tool** for solving "real world" problems.
- Hard work is required!

Administration

Lectures

Recitations

Guided Lab Instructor

Assignment graders

Course websites

1. **Course website:** http://www.cs.tau.ac.il/courses/pyProg/1819a/
    - Course schedule
    - Lecture and recitation presentations
    - Code examples
    - Assignments
    - Homework guidelines
2. **MOODLE website:** http://moodle.tau.ac.il/course/view.php?id=509182099
    - Homework submissions
    - Forums (General + assignment specific)

Recitations

- Practical Sessions
- In a standard classroom
- Purposes:
  - Practice topics presented in class.
  - Preparations for next class.
  - Background for homework assignments.
  - Learn practical tools.
- **Lectures are usually harder to understand, and it is OK.**

Guided Lab

- Optional practical session in a computer lab
- Technical support (IDLE, Python files, etc.)

Homework

- Let's read the guidelines on the course website

Guidelines

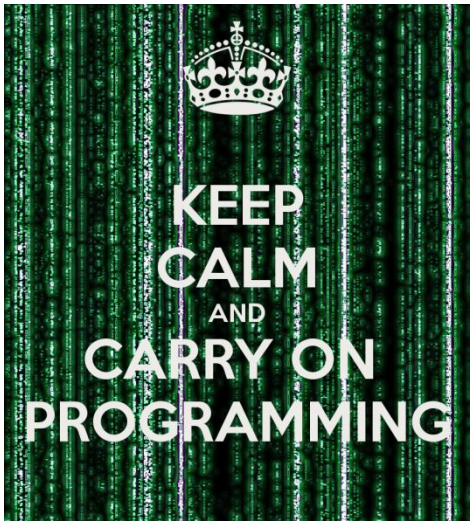
- The deadline hour is 23:55 on the deadline day.
- The exercises must be submitted via the Moodle system
- Sumbit a **py** file, file name: your_id.py (not your name).
- Submitted code must run on IDLE.
- Some of the questions are checked automatically only. Hence:
  - **Do not change** given function names, arguments list and types.
  - Make sure that the returned values are exactly as required.
- Grades will be returned via the Moodle system, solutions will appear on the course website (here).
- There will be 10 exercises: 8 during the first 8 weeks and 2 during the last 4 weeks.
- Exercises grades are 0, 60, 80, 90 and 100.
- Students must submit and pass (grade >= 60) at least 8 exercises to complete the course.
- The exercises grade will contribute 20-30% of the final grade, but only if the exam grade is >= 60. The exercises grade will be the average of the 8 best grades.
- **The exercises are personal and should be solved alone.** Students are not allowed to solve exercises together, show solutions to other students or read other students' solutions.
- Exercises can be discussed without sharing code.
- Each student has 5 days of "grace" for late submissions. For each late submission we take off a full day (or days) of "grace", even it the exercise was submitted two minutes after the deadline. If a student has no more days of "grace", late exercises will not be checked and will be graded with a zero.
- In case of a late submission, students should **not** contact staff members. If the late submission is justified, students should attach the relevant documents (sick days, reserve duty etc.) to the submission, together with a late submission form. This includes cases in which there is a malfunction with the Moodle system.

A Personal Note on HW

It will take you a lot of time and effort to make the code work.

But

There is no other way to learn how to program



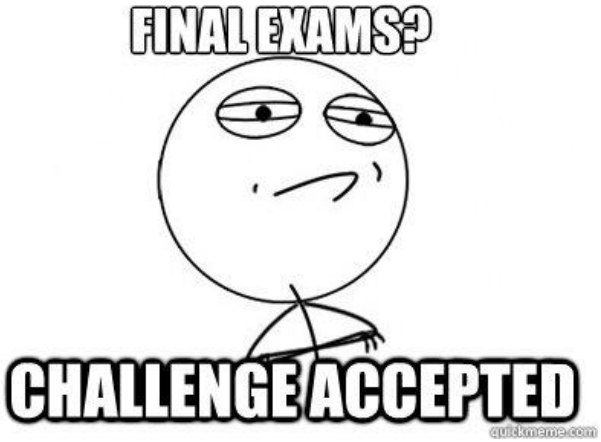
Working Environment

- Lab 008
- Home versus lab

vs.

The Exam

- Final grade is composed out of homework and final exam
- You must pass the exam to pass the course
- Written exam
- Includes all course material:
  - Lectures, recitations, and HW



Course Objectives

Develop basic **programming** and **algorithmic** skills

! Remember: we learn programming, not how computer hardware works

Syllabus

- Python programming basics
- File I/O
- Error Handling
- Recursion
- Sort & Search algorithms
- Object-Oriented Programming
- Data analysis
- Scientific Calculations using NumPy
- Image Processing

Resources

- Course slides and pointers to relevant bibliography.
- Recommended resources:
  - Book: Think Python, by Allen B. Downey (http://greenteapress.com/thinkpython/thinkpython.html)
  - Manual: Python 2.x documentation http://docs.python.org// (the official language manual)

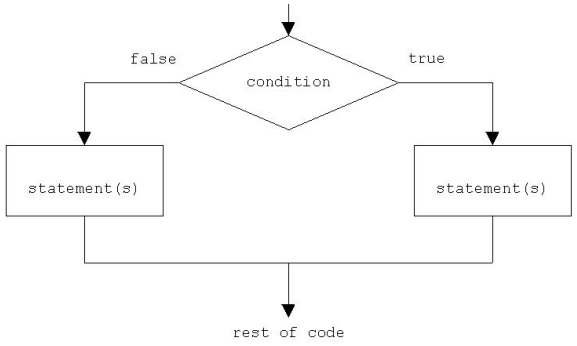
Questions?

Preface

- We assume no prior knowledge.
- However, we advance fast.
- The only way to keep on track is to **practice**!

Today

- Brief background to programming
- Python basics:
  - Variables
    - Numbers
    - Strings
  - Arithmetic Operators
  - Comparison Operators
  - Logical Operators
  - Branching (if)



Programming Languages Basics

- A computer program is a sequence of text instructions that can be “understood" by a computer and executed.
- A programming language is a machine-readable artificial language designed to express computations that can be performed by a computer.

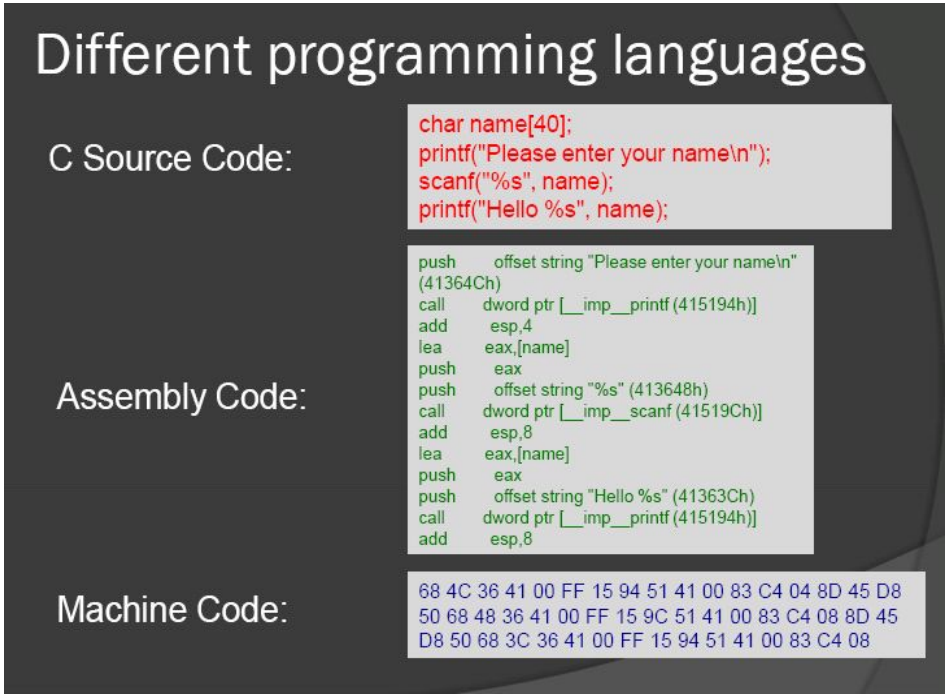
Over 500 different computer languages are listed by Wikipedia

Machine Code (Language)

- Computers understand only machine language.
  - Basically looks like a sequence of 1's and 0's.
  - Very inconvenient to work with and non-intuitive.
- All other computer languages were created for human convenience.
- The computer does not understand C/Python/Java - They must be "translated" into machine language
  - In this course, we do not care how the computer does that

Computer Program

- A sequence of instructions designed to achieve a specific purpose
- The instructions are executed sequentially. No instruction is executed before the previous is completed



Language Selection and Python

Python (since 1991):

- Quick development
- Easy to learn
- Huge community
- Short development-execution rounds
- Fast enough for most applications
- Cross-platform

Guido van Rossum

python™

Python is Good for Your Future..

Python is widely industrial used (Google, Yahoo!, YouTube, BitTorrent, IDF, NASA)

Take a look at Python's community conference

- Short development-execution rounds
- Huge community
- Fast enough for most applications
- Cross-platform

Installing and Running Python 2.7

- Python 2.7 is already installed in the computers' lab.
- Install **Anaconda** distribution for Python 2.7 from here:

  http://continuum.io/downloads

  - Available for window, Max OS and Linux
- The Anaconda package includes:
  - Python's **interpreter** required for running Python programs
  - Python editors for writing Python programs (i.e. IDLE, Spyder)
  - Many useful Python extension packages (i.e. Numpy, Scipy)
- **We do not use Python 3!**
- Regular python installation available here:

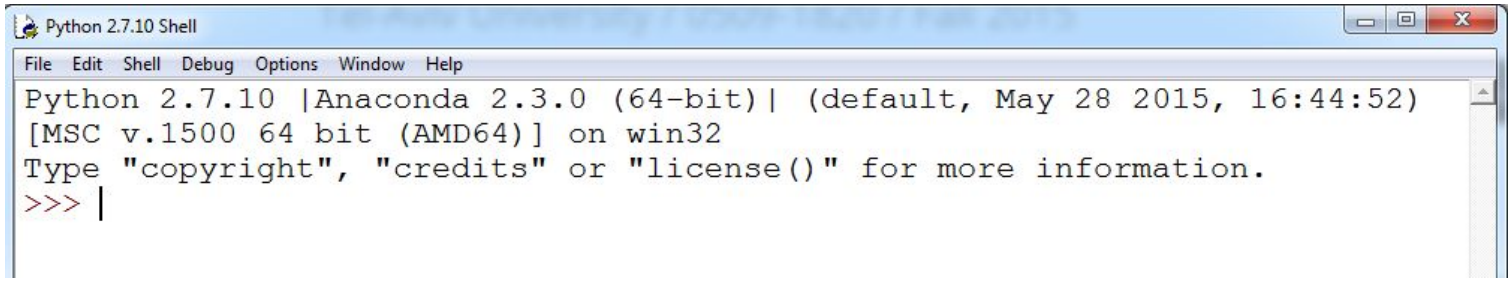
  http://python.org/download/

  This installation is **not recommended** since it does not contain all the models we are using in this course.

Using Anaconda

- Run **idle.exe** to open the Idle terminal.
  - The executable file is located in INSTALL_DIR\Anaconda\Scripts (INSTALL_DIR stands for the installation directory, usually C:\ or C:\Program Files)
  - It is recommended to create a shortcut on your desktop.

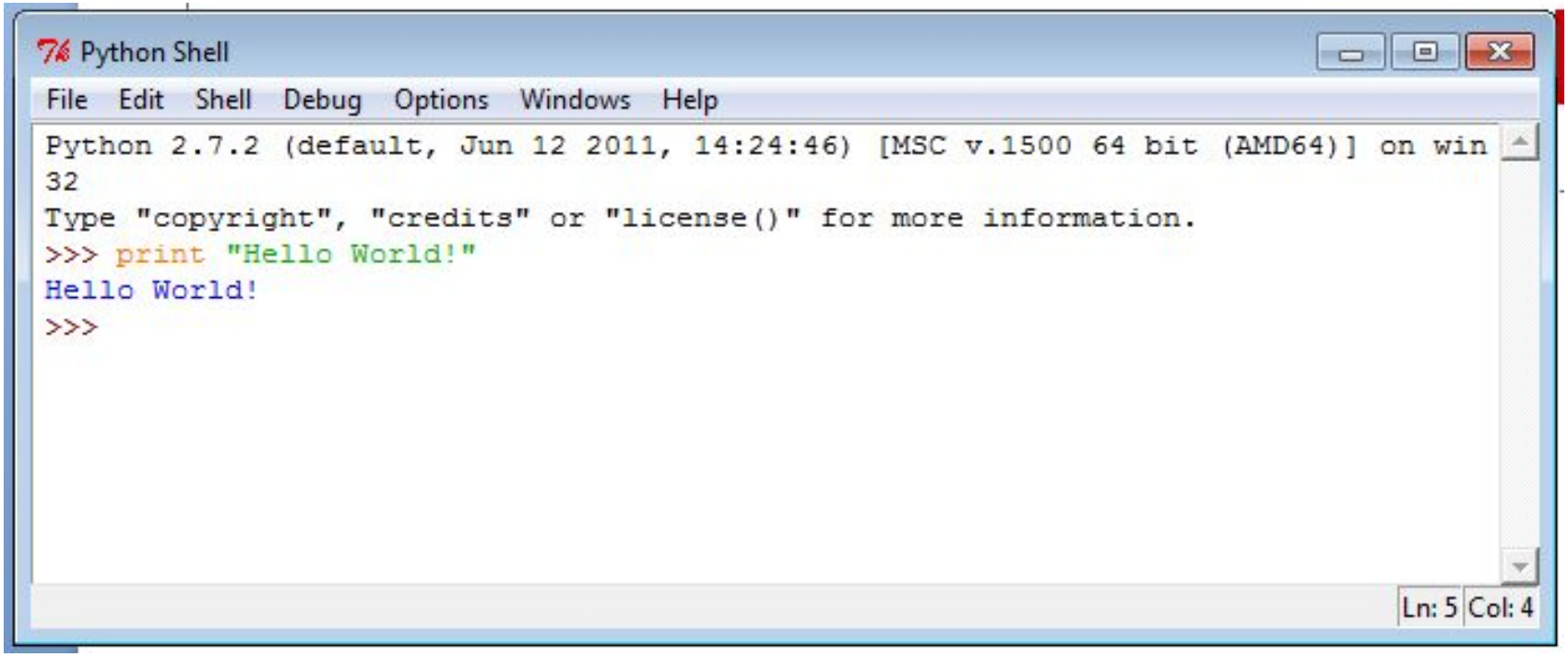
This is how idle shell looks like:



- A video on working with IDLE: http://www.youtube.com/watch?v=lBkcDFRA958

Hello World!

My First Python Program: Hello World!

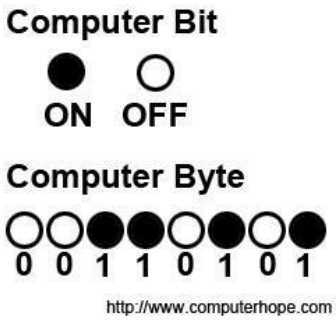


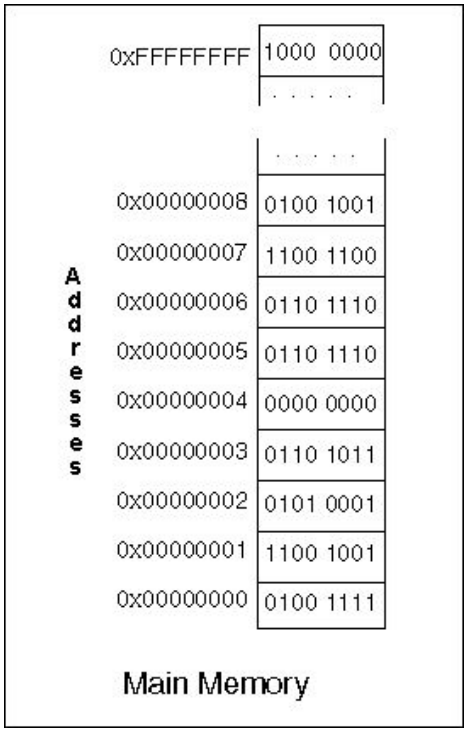
Separate commands typed in Python's shell are executed by Python's ***interpreter***, and the output is printed to the screen.

For longer programs, we will assemble several commands into a script (program), and save it to a *.py file which can be executed.

Computer's Memory

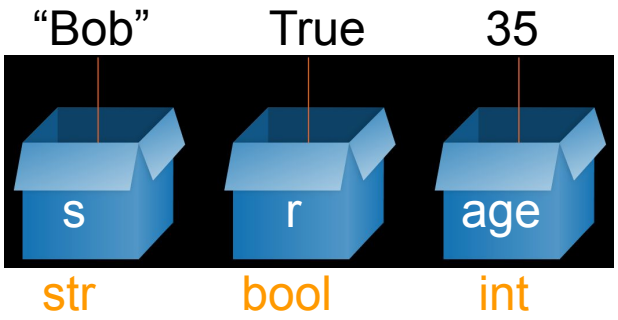
- The computer memory is composed of a long list of bits. Each bit can hold a value of either 0 or 1.
- Bits are grouped into Bytes (8 bits).
- 1 Byte can hold 256 different values (2^8).
- Every Byte is numbered sequentially. The byte's index is called its memory **address.**





Using variables to store data in memory

- Computer programs manipulate data.
- Data is given either as input, or calculated by the program.
- To access it later, data must be remembered.
- Therefore, computer programs use ***variables*** to store data in the memory.
- Each variable has…
  - A value (content, the stored data)
  - A name (a shortcut to its address in memory)
  - A type (str, int, float, bool)

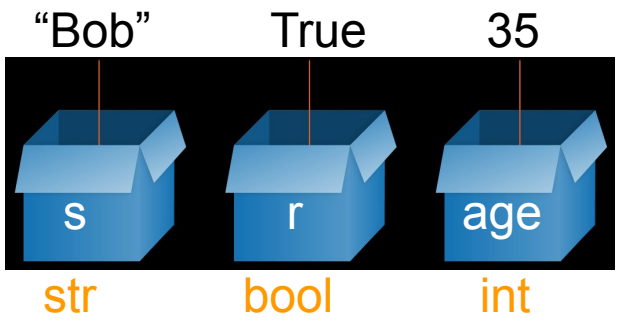


Program variables

- Each variable has: Name, Value, Type (and an Address of the location in memory where its value is stored).
- The variable's value is encoded as a binary number which is stored in one or more bytes in the computer's memory.
- In Python we create variables simply by assigning a value to a variable name:

```
s = "Bob"
r = True
age = 35
```

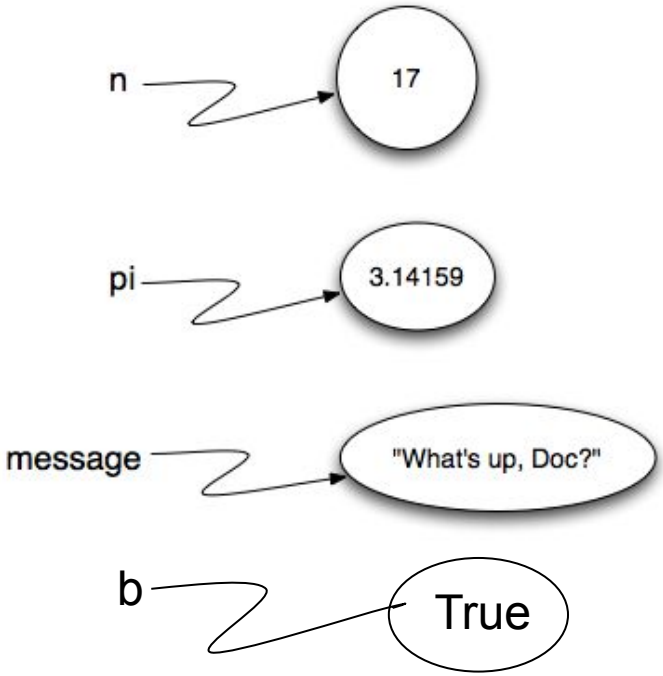
- The variable's type is automatically determined in Python based on its assigned values and actions ("duck typing")



Data Types in Python

Commonly used built in data types:

- Numeric types: **int, float, long, complex**
- Boolean: **bool**
- String: **str**



Why Do We Need Different Types?

- Saving memory
- Execution speed
- Different actions

Hands On

Python 2.7.12 Shell

File Edit Shell Debug Options Window Help

```
Python 2.7.12 (v2.7.12:d33e0cf91556, Jun 27 2016, 15:19
:22) [MSC v.1500 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more inf
ormation.
>>> a = 5
>>> a
5
>>> a = 'Hello'
>>> a
'Hello'
>>> # I never admit I don't know...
>>> 'Hello' + 6

Traceback (most recent call last):
  File "<pyshell#5>", line 1, in <module>
    'Hello' + 6
TypeError: cannot concatenate 'str' and 'int' objects
>>> 'Hello' + str(6)
'Hello6'
>>>
```

The ‘type’ command returns the type of a variable/expression

```
>>> 4
4
>>> type(4)
<class 'int'>
>>>  3.14159
3.14159
>>> type(3.14159)
<class 'float'>
```

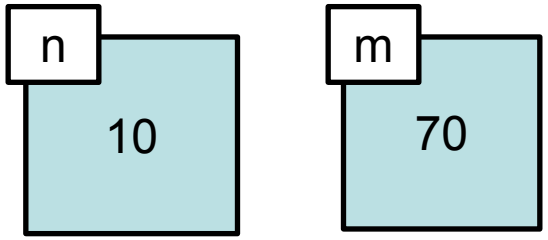
An integer number

A real number (floating point)

Variables and Assignments

```
>>> n = 10
>>> m = (10 + 4) * 5
```

Left-hand side is a variable.
Right-hand side is an expression.



The interpreter:

1. Evaluates the **expression**
2. Assigns its value to the **variable**.

Variable's name - a sequence of letters and digits, starting with a letter.

Variables and Assignments: An Example

Changing the **value** of a variable:

```
>>> n = 11
>>> n
11
```

Changing the **type** of a variable:

```
>>> n = 1.3141
>>> n
1.3141
```

Variables can be used in **expressions**:

```
>>> pi = 3.14159
>>> pi * 2 + 1
7.28318
```

Variables and Assignments – Cont.

Referring to undefined variables leads to runtime error

```
>>> check_this
Traceback (most recent call last):
  File "<pyshell#16>", line 1, in <module>
    check_this
NameError: name 'check_this' is not defined
```

Arithmetic Operators

Operator	Use	Description
+	x + y	Sum of x and y
-	x - y	Subtracts y from x
*	x * y	Multiplies x by y
**	x ** y	x to the power y
/	x / y	Divides x by y
%	x % y	Modulu: the remainder of dividing x by y

What's the type of 8/5 ? And of 8/5.0 ?

The result of int/int is an int !

Playing with Variables

```
>>> a = 3
>>> a
3
>>> b = 5
>>> b
5
>>> c = a + b
>>> c
8
>>> c = c * 2
>>> c
16
```

```
>>> first = (a + b) * 2
>>> second  = a + b * 2
>>> first, second
(16, 13)
>>> a, b
(3, 5)
>>> b / a
1
>>> b % a
2
>>> b**a
125
```

Strings

- String variables are used to save text.
- An ordered sequence of characters.

```
>>> s = "Hello"
>>> print s
Hello
>>> print s[0]
H
>>> print s[-2]
l
```

H	e	l	l	o
0	1	2	3	4
-5	-4	-3	-2	-1

String Access

```
>>> a = 'Hello'
>>> a[1]
'e'
>>> a[1:3]
'el'
>>> a[1:]
'ello'
>>> a[-4:-2]
'el'
>>> a[:-3]
'He'
>>> a[-3:]
'llo'
```

H	e	l	l	o	
0	1	2	3	4	5
-5	-4	-3	-2	-1	

Strings are a sequence of characters

- Every character in a string is mapped to a specific number based on the famous **ASCII table.**
- Strings are saved in memory as a sequence of numbers in binary form.
- In python:
  - **\n** represents new line
  - **\t** represents tab

ASCII table.

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	00	Null	32	20	Space	64	40	@	96	60	`
1	01	Start of heading	33	21	!	65	41	A	97	61	a
2	02	Start of text	34	22	"	66	42	B	98	62	b
3	03	End of text	35	23	#	67	43	C	99	63	c
4	04	End of transmit	36	24	$	68	44	D	100	64	d
5	05	Enquiry	37	25	%	69	45	E	101	65	e
6	06	Acknowledge	38	26	&	70	46	F	102	66	f
7	07	Audible bell	39	27	'	71	47	G	103	67	g
8	08	Backspace	40	28	(	72	48	H	104	68	h
9	09	Horizontal tab	41	29	)	73	49	I	105	69	i
10	0A	Line feed	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage return	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	46	2E	.	78	4E	N	110	6E	n
15	0F	Shift in	47	2F	/	79	4F	O	111	6F	o
16	10	Data link escape	48	30	0	80	50	P	112	70	p
17	11	Device control 1	49	31	1	81	51	Q	113	71	q
18	12	Device control 2	50	32	2	82	52	R	114	72	r
19	13	Device control 3	51	33	3	83	53	S	115	73	s
20	14	Device control 4	52	34	4	84	54	T	116	74	t
21	15	Neg. acknowledge	53	35	5	85	55	U	117	75	u
22	16	Synchronous idle	54	36	6	86	56	V	118	76	v
23	17	End trans. block	55	37	7	87	57	W	119	77	w
24	18	Cancel	56	38	8	88	58	X	120	78	x
25	19	End of medium	57	39	9	89	59	Y	121	79	y
26	1A	Substitution	58	3A	:	90	5A	Z	122	7A	z
27	1B	Escape	59	3B	;	91	5B	[	123	7B	{
28	1C	File separator	60	3C	<	92	5C	\	124	7C	|
29	1D	Group separator	61	3D	=	93	5D	]	125	7D	}
30	1E	Record separator	62	3E	>	94	5E	^	126	7E	~
31	1F	Unit separator	63	3F	?	95	5F	_	127	7F	□

String Type

Python 2.7.12 Shell

File Edit Shell Debug Options Window Help

```
Python 2.7.12 (v2.7.12:d33e0cf91556, Jun 27 2016
, 15:19:22) [MSC v.1500 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for m
ore information.
>>> a = 'Hello'
>>> type(a)
<type 'str'>
>>> len(a)
5
>>> a + "world"
'Helloworld'
>>> a
'Hello'
>>> # strings are Immutable
>>> str.lower(a)
'hello'
>>> a
'Hello'
>>>
```

Strings concatenation

```
>>> s1 = "He"
>>> s2 = "llo"
>>> s3 = s1 + s2
>>> s3
'Hello'
>>> s4 = s3 + " World"
>>> c = "!"
>>> print s4, 2015, c
Hello World 2015 !
```

Strings Indices

Python 2.7.12 Shell

File Edit Shell Debug Options Window Help

```
Python 2.7.12 (v2.7.12:d33e0cf91556, Jun 27 2016
, 15:19:22) [MSC v.1500 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for m
ore information.
>>> a = 'Hello'
>>> str.find(a,'H')
0
>>> a[0]
'H'
>>> a[1]
'e'
>>> a[4]
'o'
>>> a[5]

Traceback (most recent call last):
  File "<pyshell#5>", line 1, in <module>
    a[5]
IndexError: string index out of range
>>>
```

Strings are Immutable

You cannot mutate (change) existing strings. Only create new ones !

```
>>> a = "abc"
>>> a[0] = 'a'
Traceback (most recent call last):
  File "<pyshell#21>", line 1, in <module>
    a[0]='a'
TypeError: 'str' object does not support item assignment
```

However, pointing to another string is valid:

```
>>> a = "abc"
>>> a = "ggg"
```

- Immutable variables cannot be changed after created.
- Applying operations on immutable variables usually return a new variable rather changing the original variable

Special characters and string operators

http://www.tutorialspoint.com/python/python_strings.htm

- Special characters: \n (new line) \t (tab)
- Special string operators:

'a = 'Hello', b = 'Python

Operator	Description	Example
+	Concatenation - Adds values on either side of the operator	a + b will give 'HelloPython'
*	Repetition - Creates new strings, concatenating multiple copies of the same string	a*2 will give 'HelloHello'
[]	Slice - Gives the character from the given index	a[1] will give 'e'
[:]	Range Slice - Gives the characters from the given range	a[1:4] will give 'ell'
in	Membership - Returns true if a character exists in the given string	'H' in a will give True
not in	Membership - Returns true if a character does not exist in the given string	'M' not in a will give True
%	Format - Performs String formatting	See at next section

Strings - Built In Methods

http://docs.python.org/release/2.5.2/lib/string-methods.html

The *str* type in Python includes many built-in commands for working with Strings

Python 2.7.6 Shell

File Edit Shell Debug Options Windows Help

```
Python 2.7.6 (default, Nov 10 2013, 19:24:24) [MSC v.1500 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> str.upper('Assaf')
'ASSAF'
>>> str.lower('Noga')
'noga'
>>> str.replace('Python class occurs on Mondays','Mondays','Tuesdays')
'Python class occurs on Tuesdays'
>>> 'Py'+'thon' #concatenate
'Python'
>>> str.title('nach nachman nachman meuman')
'Nach Nachman Nachman Meuman'
>>> str.title(str.replace('Python class occurs on Mondays','Mondays','Tuesdays'))
'Python Class Occurs On Tuesdays'
>>> 'repeat' * 4 #repetition
'repeatrepeatrepeatrepeat'
```

Strings - Built In Methods

http://www.tutorialspoint.com/python/python_strings.htm

- String Formatting Operator

```
>>> print "My name is %s and my age is %d !" % ('Zara', 21)
My name is Zara and my age is 21 !
```

- Useful String methods:
  - len
  - find, startswith, endswith
  - isalpha, isdigit, islower,…
  - join, replace
  - strip, rstrip
  - split

Type Conversion

Convert variable type using *int()*, *str()* and *float()*

```
>>> num = 123
>>> num
123
>>> num_str = str(num)
>>> num_str
'123'
>>> int(2.5)
2
```

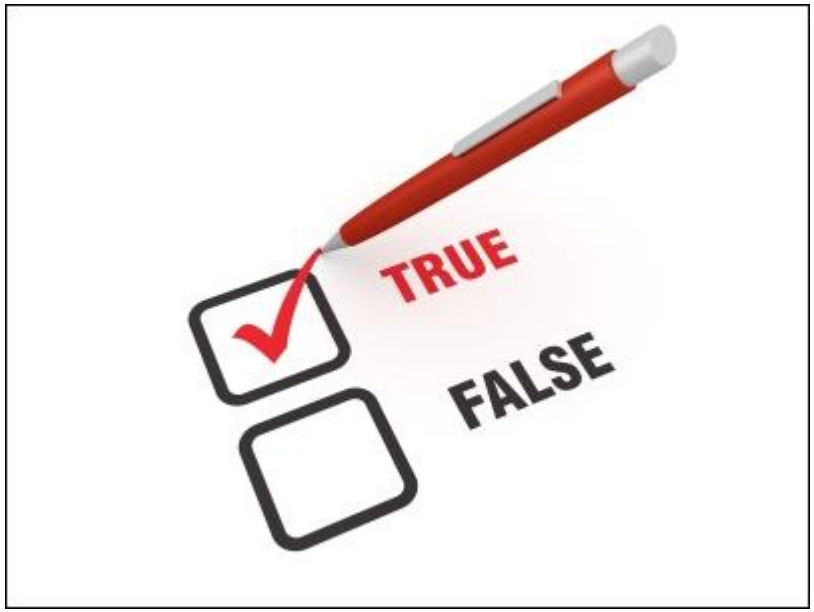
Comparison Operators

Compares two variables and returns a **Boolean type** result/variable

Operator	Name	Description
x < y	Less than	true if x is less than y, otherwise false.
x > y	Greater than	true if x is greater than y, otherwise false.
x <= y	Less than or equal to	true if x is less than or equal to y, otherwise false.
x >= y	Greater than or equal to	true if x is greater than or equal to y, otherwise false.
x == y	Equal	true if x equals y, otherwise false.
x != y	Not Equal	true if x is not equal to y, otherwise false.

Comparison Operators

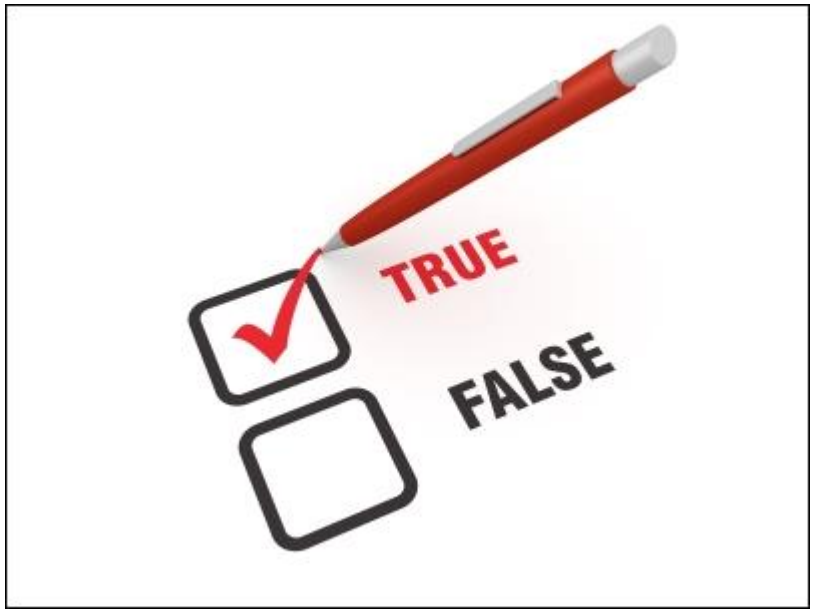
```
>>> 5 == 5.0
True
>>>  6 != 2*3
False
>>> -2 >= 1
False
>>> 3 <= 3
True
>>> x = 3 < 3
>>> x
False
```




```
>>> type(x)
<type 'bool'>
```

Comparison Operators

```
>>> 'a' != 'b'
True
>>> 'a' < 'b'
True
```



Logical Operators

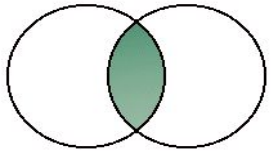
Operate on two Booleans and return Booleans

Operator	Description
x **and** y	Both True: **True**, otherwise: **False**.
x **or** y	At least one is rue: **True**, Otherwise: **False**.
not x	x is False □ **True**, x is True □ **False**

And, or, not

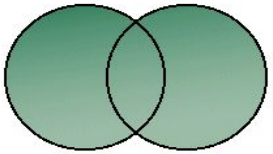
and

	0	1
0	0	0
1	0	1



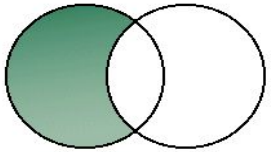
or

	0	1
0	0	1
1	1	1



not

0	1
1	0



Logical Operators

```
>>> a = True
>>> b = True
>>> c = False
>>> d = False
>>>  a and b
True
>>>  a and c
False
>>>  a or c
True
```

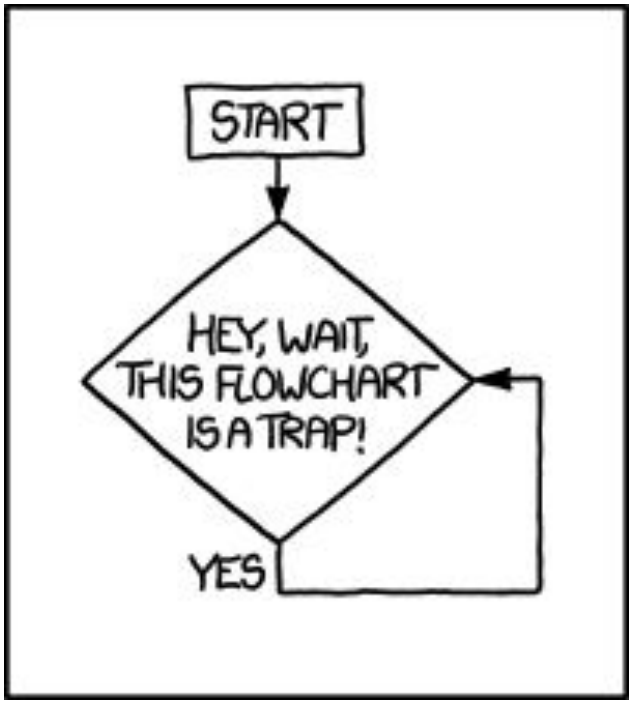
Flow Control

Different inputs □Different execution order

- Computer games
- Illegal input

Control structures

- **if-else**
- **for loop**
- **while** loop



http://xkcd.com/1195/

Conditional Statement: if

Used to execute statements conditionally

<u>Syntax</u>

if *condition:*

    statement1

    statement2

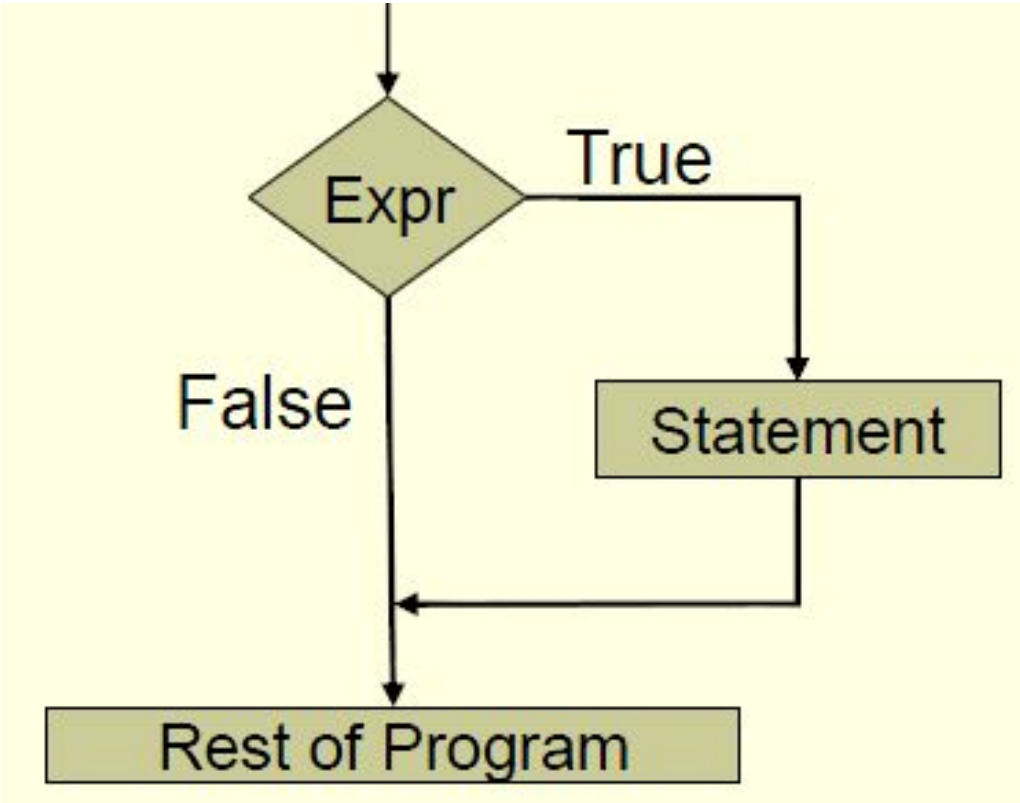
    ...

Condition = expression that evaluates to a Boolean

Indentation = determines the scope of the **if** block

- If *condition* is True, *statements* are executed

Conditional Statements



Conditional Statements - Examples

```
num = 54 # choose a number
if num % 18 == 0: # num is a multiplication of 18
    print num, "is divisible by 18"
    res = num / 18
print "Goodbye“
```

```
54 is divisible by 18
Goodbye
```

Conditional Statements

Indentation:

- Following the if statement:

  Open a new scope = one tab to the right.
- Indicates the commands within the scope of this if.

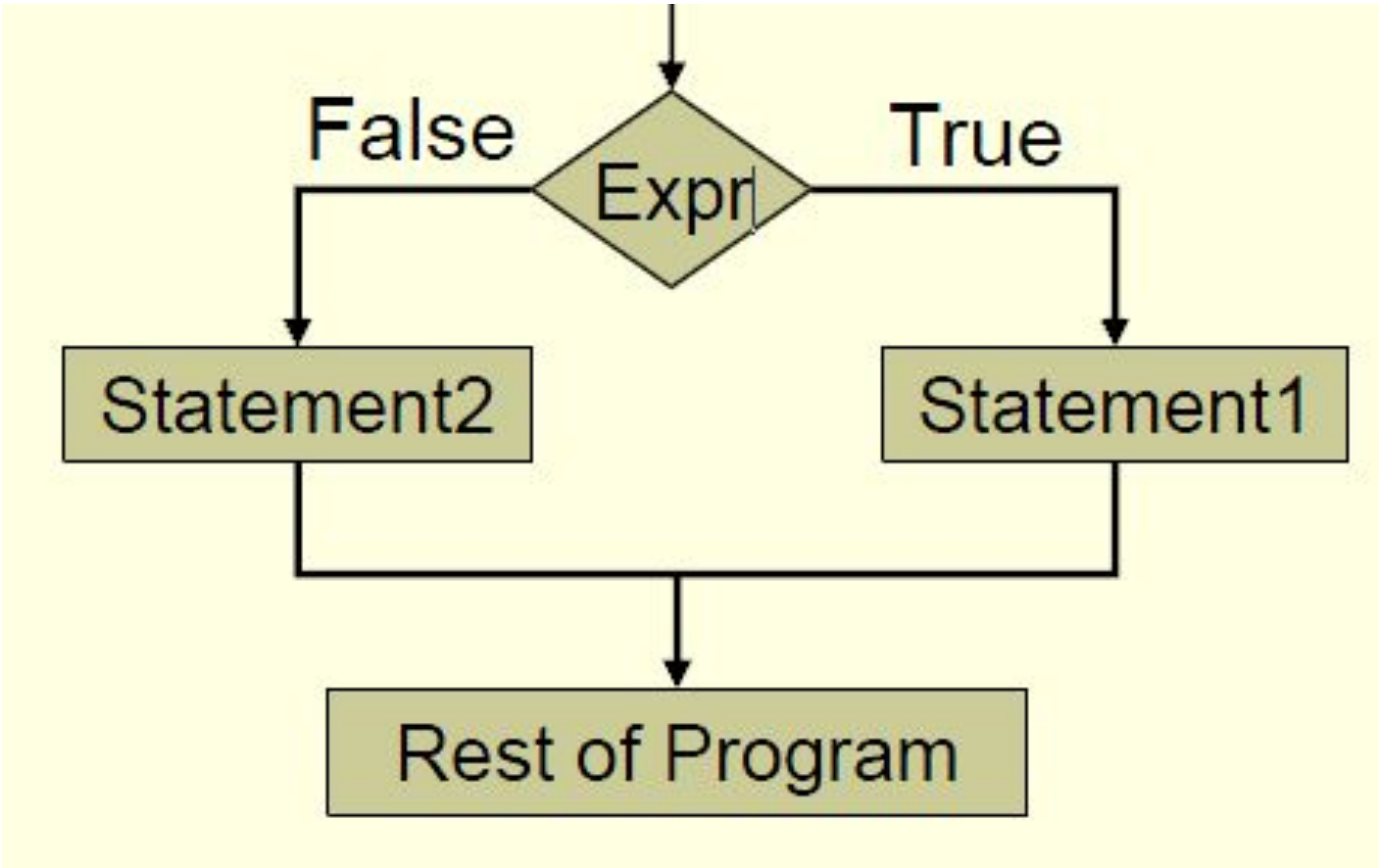
if-else

if *condition*$_1$:
  statement$_1$
else:
  statement$_2$

rest of code

condition$_1$ is true □ execute *statement*$_1$
condition$_1$ is false □ execute *statement*$_2$
execute *rest of code*

if-else



if-else

```
if width == height:
    print "found a square"
else:
    print "found a rectangle"
    width = height
    print "now it is a square"
```

Indentation:

else is <u>not</u> a part of the if scope!

The commands under else are indented.

if-else

```
a = 4
b = 5
c = 6

if a + b > c and a + c > b and b + c > a:
    print "Building a triangle"
else:
    print "Cannot build a triangle"
```

if-elif-else

if *condition*$_1$:
    statement$_1$
elif *condition*$_2$:
    statement$_2$
else:
    statement$_3$

rest of code

elif = if-else

condition$_1$ is true □ execute *statement*$_1$
condition$_1$ false and *condition*$_2$ true □ execute *statement*$_2$
condition$_1$ and *condition*$_2$ are false □ execute *statement*$_3$
execute *rest of code*

if-elif-else

```
if price < 100:
    print "too cheap"
elif price > 200:
    print "too expensive“
else:
    print "reasonable price"
```