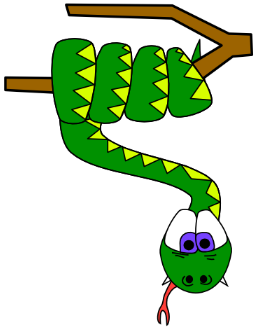
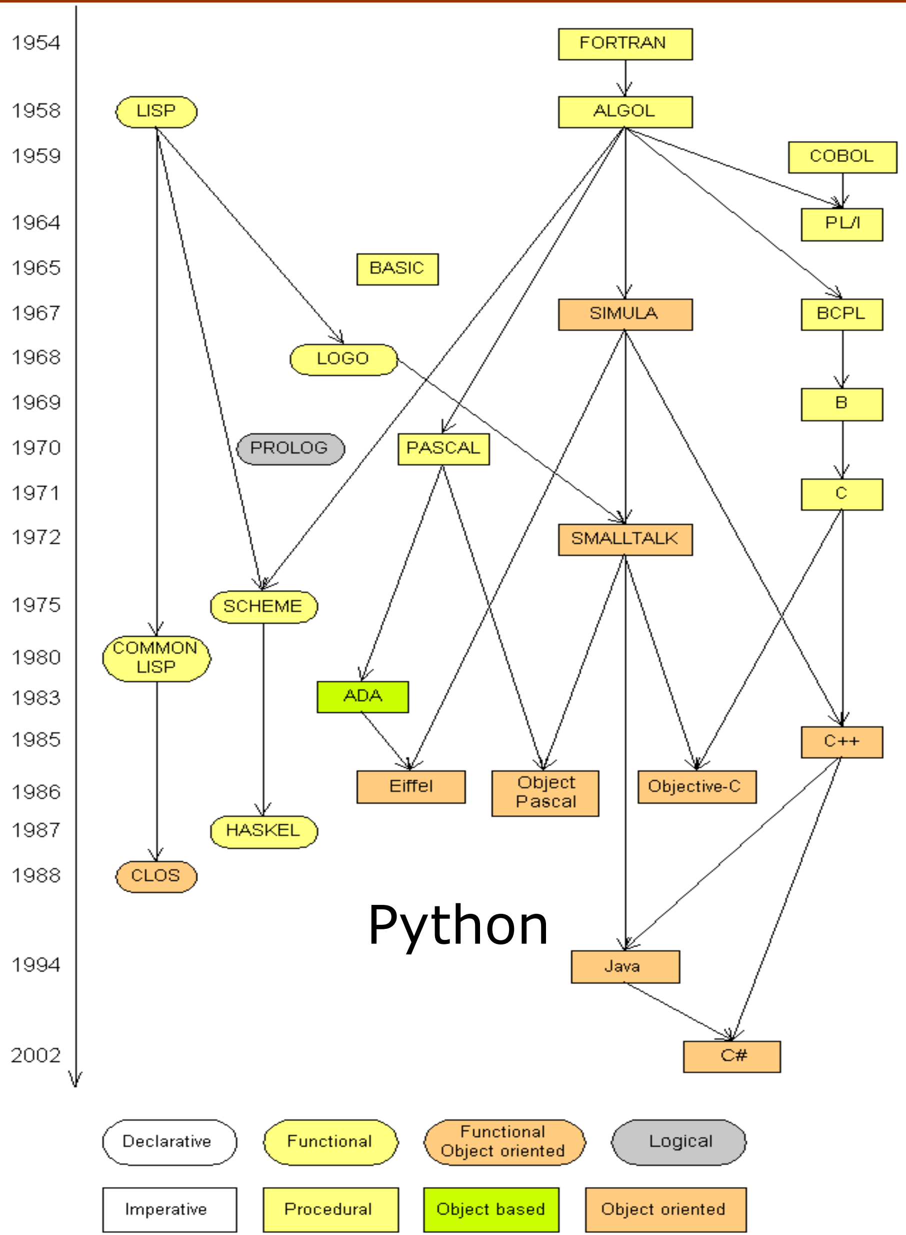


Introduction to Python Programming

Dr. Vinay Chamola
EEE F411 Internet of Things, BITS-Pilani

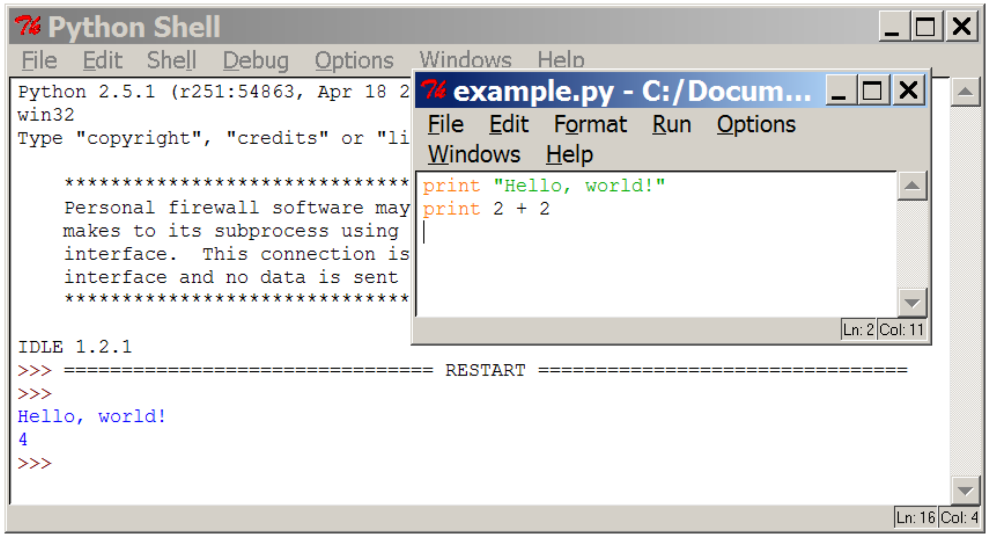
Languages

- Some influential ones:
  - FORTRAN
    - science / engineering
  - COBOL
    - business data
  - LISP
    - logic and AI
  - BASIC
    - a simple language



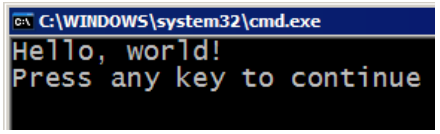
Programming basics

- **code** or **source code**: The sequence of instructions in a program.
- **syntax**: The set of legal structures and commands that can be used in a particular programming language.
- **output**: The messages printed to the user by a program.
- **console**: The text box onto which output is printed.
  - Some source code editors pop up the console as an external window, and others contain their own console window.



```
Python 2.5.1 (r251:54863, Apr 18 2
win32
Type "copyright", "credits" or "li

    ******************************
    Personal firewall software may
    makes to its subprocess using
    interface.  This connection is
    interface and no data is sent
    ******************************

IDLE 1.2.1
>>> ================================ RESTART ================================
>>>
Hello, world!
4
>>>
```

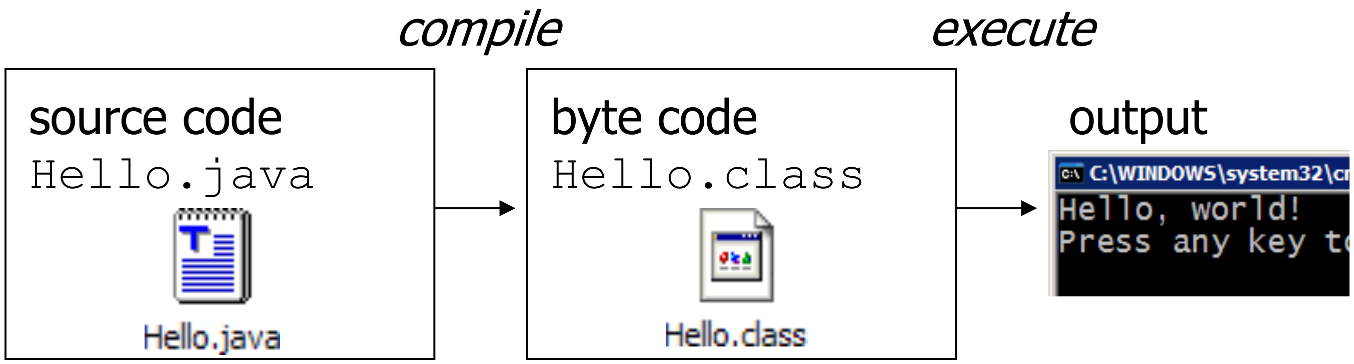
```
print "Hello, world!"
print 2 + 2
```



```
Hello, world!
Press any key to continue
```

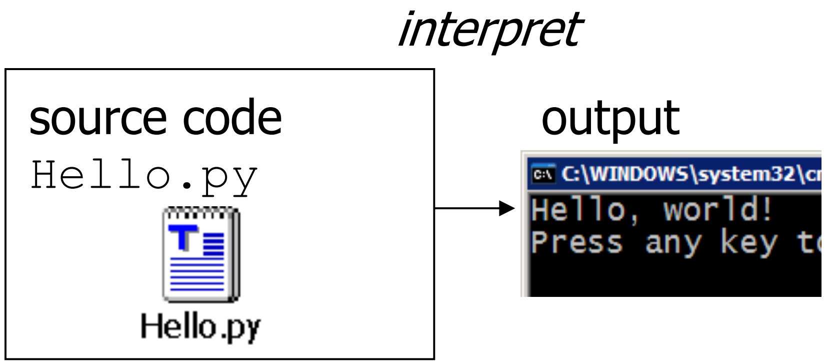
Compiling and interpreting

- Many languages require you to *compile* (translate) your program into a form that the machine understands.



```
Hello, world!
Press any key t
```

- Python is instead directly *interpreted* into machine instructions.



```
Hello, world!
Press any key t
```

Expressions

- **expression**: A data value or set of operations to compute a value.

  Examples: `1 + 4 * 3`

  `42`

- Arithmetic operators we will use:

`+ - * /`	addition, subtraction/negation, multiplication, division
`%`	modulus, a.k.a. remainder
`**`	exponentiation

- **precedence**: Order in which operations are computed.
  - `* / % **` have a higher precedence than `+ -`

    `1 + 3 * 4` is `13`

  - Parentheses can be used to force a certain order of evaluation.

    `(1 + 3) * 4` is `16`

Real numbers

- Python can also manipulate real numbers.
  - Examples: `6.022` `-15.9997` `42.0` `2.143e17`

- The operators `+ - * / % ** ( )` all work for real numbers.
  - The `/` produces an exact answer: `15.0 / 2.0` is **`7.5`**
  - The same rules of precedence also apply to real numbers: Evaluate `( )` before `* / %` before `+ -`

- When integers and reals are mixed, the result is a real number.
  - Example: `1 / 2.0` is `0.5`
  - The conversion occurs on a per-operator basis.

```
7 / 3 * 1.2 + 3 / 2
    2 * 1.2 + 3 / 2
      2.4   + 3 / 2
      2.4   +   1
         3.4
```

Math commands

- Python has useful commands for performing calculations.

Command name	Description
`abs(`***value***`)`	absolute value
`ceil(`***value***`)`	rounds up
`cos(`***value***`)`	cosine, in radians
`floor(`***value***`)`	rounds down
`log(`***value***`)`	logarithm, base *e*
`log10(`***value***`)`	logarithm, base 10
`max(`***value1***`,` ***value2***`)`	larger of two values
`min(`***value1***`,` ***value2***`)`	smaller of two values
`round(`***value***`)`	nearest whole number
`sin(`***value***`)`	sine, in radians
`sqrt(`***value***`)`	square root

Constant	Description
`e`	2.7182818...
`pi`	3.1415926...

- To use many of these commands, you must write the following at the top of your Python program:

```
from math import *
```

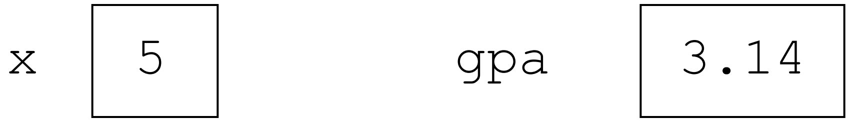
Variables

- **variable**: A named piece of memory that can store a value.
  - Usage:
    - Compute an expression's result,
    - store that result into a variable,
    - and use that variable later in the program.

- **assignment statement**: Stores a value into a variable.
  - Syntax:

    name = ***value***

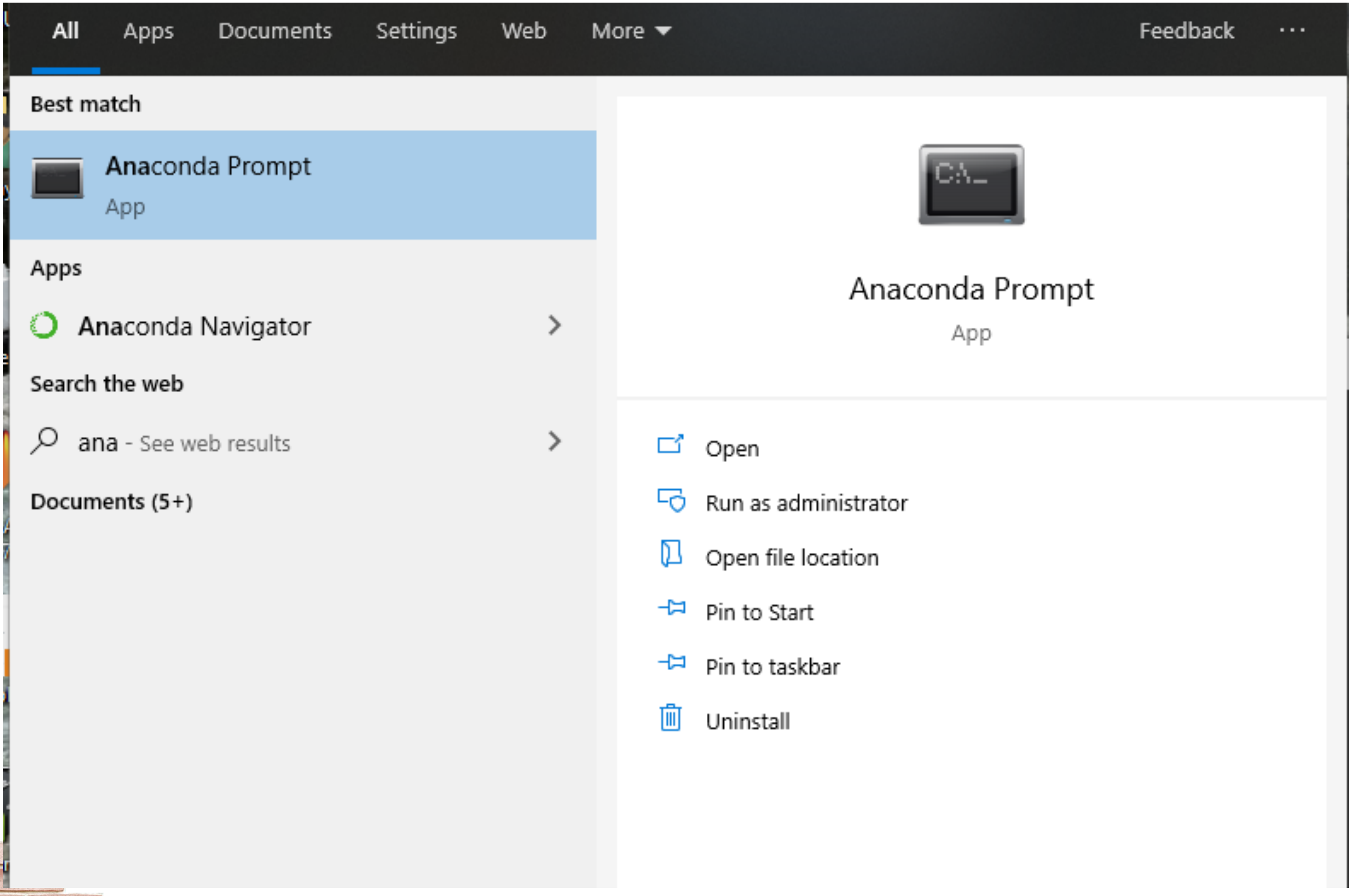
  - Examples:
    ```
    x = 5
    gpa = 3.14
    ```



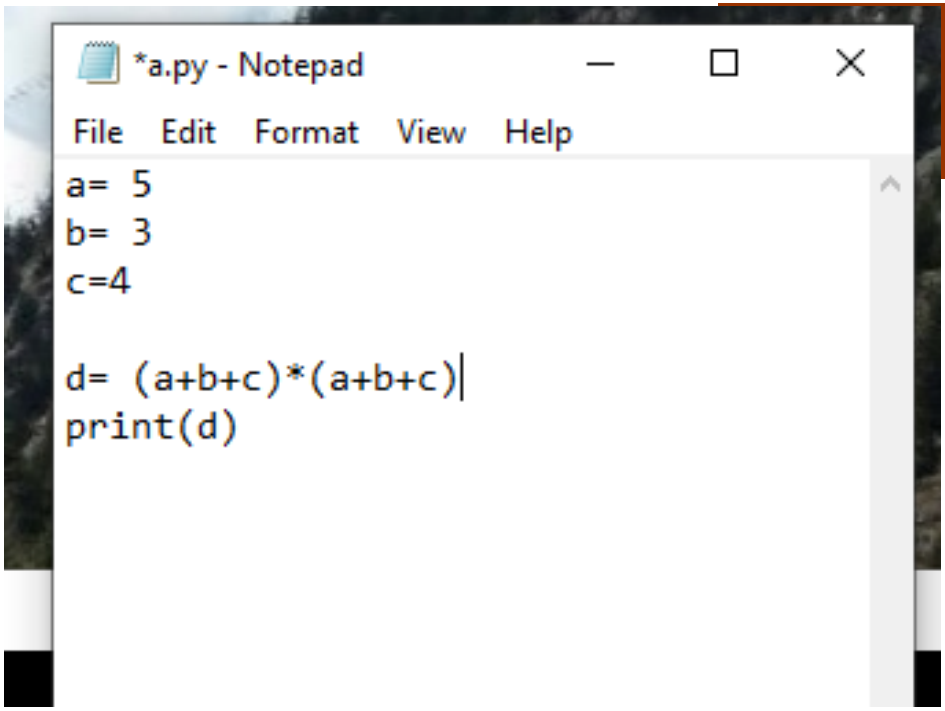
  - A variable that has been given a value can be used in expressions.
    `x + 4` is `9`

- **Exercise:** Evaluate the quadratic equation for a given *a*, *b*, and *c*.
- (a + b+ c)^2

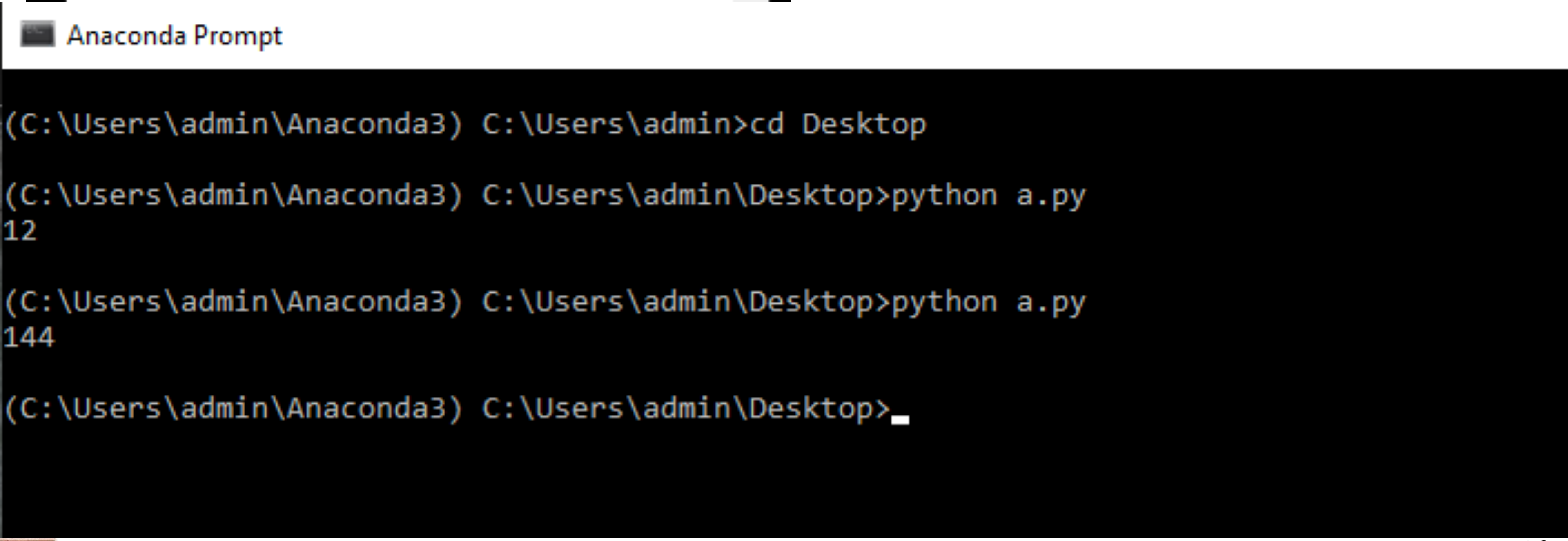
Anaconda prompt



Workout



```
a= 5
b= 3
c=4

d= (a+b+c)*(a+b+c)
print(d)
```



```
(C:\Users\admin\Anaconda3) C:\Users\admin>cd Desktop

(C:\Users\admin\Anaconda3) C:\Users\admin\Desktop>python a.py
12

(C:\Users\admin\Anaconda3) C:\Users\admin\Desktop>python a.py
144

(C:\Users\admin\Anaconda3) C:\Users\admin\Desktop>
```

print

- `print` : Produces text output on the console.

- Syntax:

  `print ('`***Message***`')`

  `print (`***Expression)***

  - Prints the given text message or expression value on the console, and moves the cursor down to the next line.

  `print (`***Item1***`,` ***Item2***`, ...,` ***ItemN)***

  - Prints several messages and/or expressions on the same line.

- Examples:

```
print ('Hello, world!') ; print ('Hello', 'world');
age = 30
print "You have", 65 - int(age), "years until retirement"
```

Output:

```
Hello, world!
You have 35 years until retirement
```

input

- `input` : Reads a number from user input.
  - You can assign (store) the result of `input` into a variable.
  - Example:

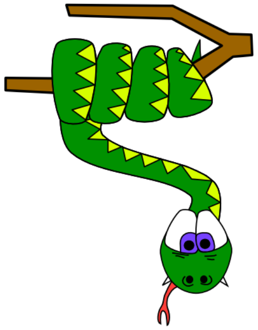
```
age = input('How old are you?')
print ('Your age is', age)
print ('You have", 65 - int(age), 'years until retirement'
```

Output:

```
How old are you? 30
Your age is 30
You have 35 years until retirement
```

- **Exercise:** Write a Python program that prompts the user for his/her amount of money, then reports how many cars the person can afford. (car cost: 100000)

- money= input('how much money do you have')
- print('The number of cars you can purchase is', round(int(money)/100000))



Repetition (loops) and Selection (if/else)

The `for` loop

- **`for` loop**: Repeats a set of statements over a group of values.
  - Syntax:

```
for variableName in groupOfValues:
    statements
```

    - We indent the statements to be repeated with tabs or spaces.
    - ***variableName*** gives a name to each value, so you can refer to it in the ***statements***.
    - ***groupOfValues*** can be a range of integers, specified with the `range` function.

  - Example:

```
for x in range(1, 6):
    print (x, "squared is", x * x)
```

Output:

```
1 squared is 1
2 squared is 4
3 squared is 9
4 squared is 16
5 squared is 25
```

```
for x in range(1, 6):
 print (x, "squared is", x * x)
 print('I am done')
print('hello')
```

range

- The `range` function specifies a range of integers:
  - `range(`***start*****,** ***stop***`)` - the integers between ***start*** (inclusive) and ***stop*** (exclusive)
- It can also accept a third value specifying the change between values.
  - `range(`***start*****,** ***stop*****,** ***step***`)` - the integers between ***start*** (inclusive) and ***stop*** (exclusive) by ***step***
- Example:

```
for x in range(5, 0, -1):
    print (x)
print ('Congratulations')
```

Output:

```
5
4
3
2
1
Congratulations!
```

- **Exercise:** Print 1 to 10 and then 10 to 1 by one code?

Cumulative loops

- Some loops incrementally compute a value that is initialized outside the loop. This is sometimes called a *cumulative sum*.

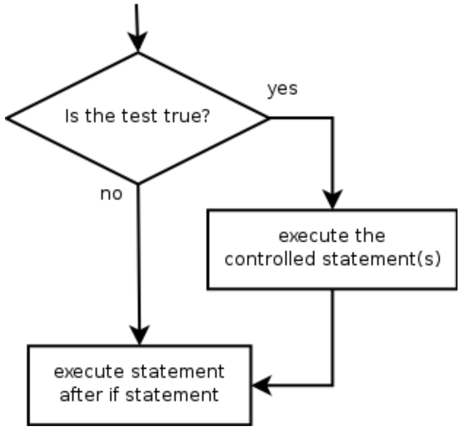
```
sum = 0
for i in range(1, 11):
    sum = sum + (i * i)
print ('sum of first 10 squares is', sum)

Output:
sum of first 10 squares is 385
```

- **Exercise:** Write a Python program that computes the factorial of an integer.

`if`

- **`if` statement**: Executes a group of statements only if a certain condition is true. Otherwise, the statements are skipped.
  - Syntax:

```
if condition:
    statements
```


Is the test true?
yes
no
execute the controlled statement(s)
execute statement after if statement

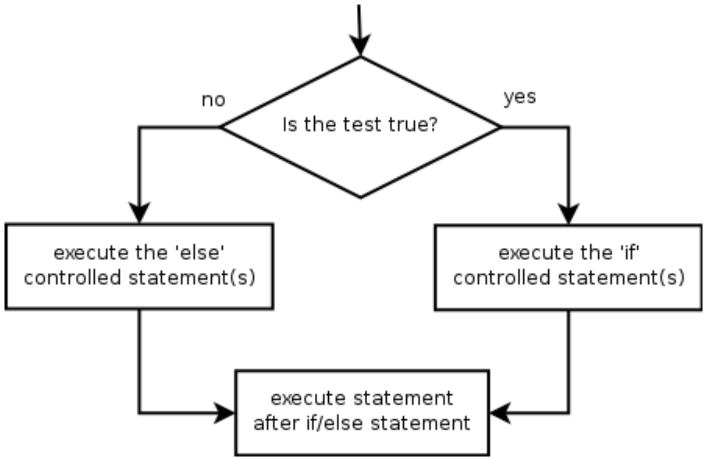
- Example:

```
x = 3.4
if x > 2.0:
    print "Your application is accepted."
```

if/else

- **`if/else` statement**: Executes one block of statements if a certain condition is True, and a second block of statements if it is False.
  - Syntax:

```
if condition:
    statements
else:
    statements
```



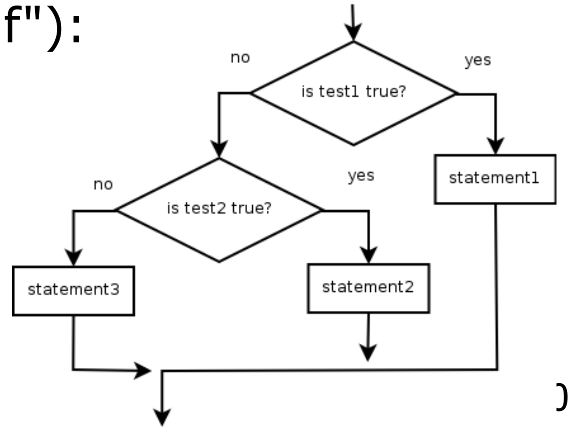
no
yes
Is the test true?
execute the 'else' controlled statement(s)
execute the 'if' controlled statement(s)
execute statement after if/else statement

- Example:

```
x = 6
if x > 4.0:
    print ('x is greater than 4!')
else:
    print ('x is smaller than 4')
```

- Multiple conditions can be chained with `elif` ("else if"):

```
if condition:
    statements
elif condition:
    statements
else:
    statements
```



`while`

- **`while` loop**: Executes a group of statements as long as a condition is True.
  - good for *indefinite loops* (repeat an unknown number of times)
- Syntax:

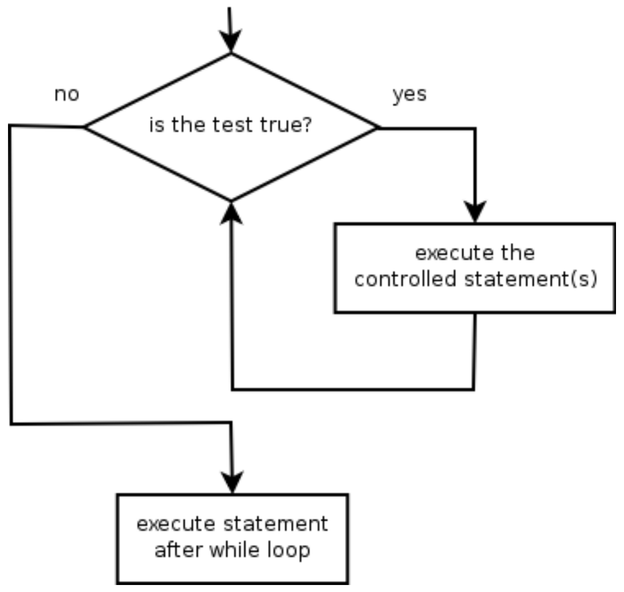
```
while condition:
      statements
```

- Example:

```
number = 1
while number < 200:
    print (number),
    number = number * 2
```

  - Output:

```
1 2 4 8 16 32 64 128
```


no
is the test true?
yes
execute the controlled statement(s)
execute statement after while loop

Logic

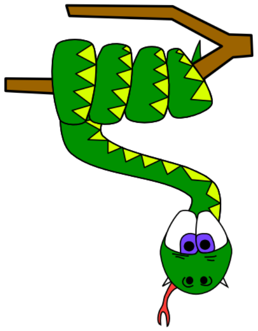
- Many logical expressions use *relational operators*:

Operator	Meaning	Example	Result
`==`	equals	`1 + 1 == 2`	`True`
`!=`	does not equal	`3.2 != 2.5`	`True`
`<`	less than	`10 < 5`	`False`
`>`	greater than	`10 > 5`	`True`
`<=`	less than or equal to	`126 <= 100`	`False`
`>=`	greater than or equal to	`5.0 >= 5.0`	`True`

- Logical expressions can be combined with *logical operators*:

Operator	Example	Result
`and`	`9 != 6 and 2 < 3`	`True`
`or`	`2 == 3 or -1 < 5`	`True`
`not`	`not 7 > 0`	`False`

- **Exercise: Write code to display the factors of a number.**



Text and File Processing

Strings

- **string**: A sequence of text characters in a program.
  - Strings start and end with quotation mark `"` or apostrophe `'` characters.
  - Examples:

```
"hello"
"This is a string"
"This, too, is a string.   It can be very long!"
```

- A string may not span across multiple lines or contain a " character.

```
"This is not
a legal String."
```

```
"This is not a "legal" String either."
```

- A string can represent characters by preceding them with a backslash.
  - `\t` tab character
  - `\n` new line character
  - `\"` quotation mark character
  - `\\` backslash character
  - Example: print ("Hello\tthere\nHow are you?")

Indexes

- Characters in a string are numbered with *indexes* starting at 0:
  - Example:

```
name = “VChamola"
```

index	0	1	2	3	4	5	6	7
character	V	C	h	a	m	o	l	a

- Accessing an individual character of a string:

  variableName [***index***]

  - Example:

```
print name, "starts with", name[0]
```

  Output:

```
Vchamola starts with V
```

String properties

- `len(`***string***`)` - number of characters in a string (including spaces)
- `str.lower(`***string***`)` - lowercase version of a string
- `str.upper(`***string***`)` - uppercase version of a string

- Exercise: Print your name length and name in uppercase

```
  name = input (‘what is your name’)


length = len(name)
  big_name = str.upper(name)
  print (length, big_name)
```

Text processing

- **text processing**: Examining, editing, formatting text.
  - often uses loops that examine the characters of a string one by one

- A `for` loop can examine each character in a string in sequence.
  - Example:

```
for c in "sun-moon":
    print c
```

Output:

```
s
u
n
-
m
o
o
n
```

Strings and numbers

- `ord(`***text***`)` - converts a string into a number.
  - Example: `ord("a")` is `97`, `ord("b")` is `98`, ...
  - Characters map to numbers using standardized mappings such as *ASCII* and *Unicode*.

- `chr(`***number***`)` - converts a number into a string.
  - Example: `chr(99)` is `"c"`