

# TODAY

---

What is programming about?

Why Python?

# What is programming?

## Programmer



What my friends think I do



What my mom thinks I do



What society thinks I do



What my boss thinks I do



What I think I do



What I actually do

# Imagine



# Running a Booth

- Dropping a ball from the top and you will get
  - 3: Big prize
  - 2: Medium prize
  - 1: Small prize



# Let's see what we have for prizes

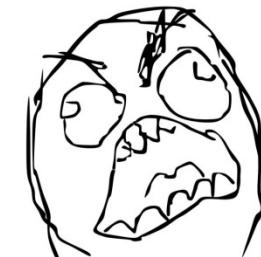
- Giant Teddy Bears
  - X 10



- Water guns
  - X 50



- Candy
  - X 200



# Your Job is

- Run the booth until the end of the day or all prizes given out
  - 3: Giant Teddy Bear
  - 2: Water Gun
  - 1: Candy
- What is the potential problem?
- You may run out of Teddy Bears!
  - Or Water guns or candies
- Any suggestion?



# Version 1

Let him play the Plinko  
If someone strikes a “3”  
    Give him a bear  
If someone strikes a “2”  
    Give him a water gun  
If someone strikes a “1”  
    Give him one pathetic candy



# Version 1.1

Let him play the Plinko

If someone strikes a “3”

If we have bears

Give him a bear

If someone strikes a “2”

If we have water guns

Give him a water gun

If someone strikes a “1”

If we have candies

Give him one pathetic candy



# Version 1.2

Let him play the Plinko  
If someone strikes a “3”  
    If we have bears  
        Give him a bear  
    Otherwise  
        Give him a water gun  
If someone strikes a “2”  
    If we have water guns  
        Give him a water gun  
If someone strikes a “1”  
    If we have candies  
        Give him one pathetic candy



# Version 1.3

Let him play the Plinko  
If someone strikes a “3”  
    If we have bears  
        Give him a bear  
    Otherwise if we have water guns  
        Give him a water gun  
Otherwise  
    Give him a pathetic candy  
If someone strikes a “2”  
    If we have water guns  
        Give him a water gun  
If someone strikes a “1”  
    If we have candies  
        Give him one pathetic candy



# Version 2

While we have prizes left, we do the following:

Let someone play the Plinko

If he strikes a “3”

If we have bears

    Give him a bear

Otherwise let him choose any prize

If someone strikes a “2”

If we have water guns

    Give him a water gun

Otherwise give him a candy

If someone strikes a “1”

If we have candies

    Give him one pathetic candy

Otherwise, sorry, you don’t even have a pathetic candy

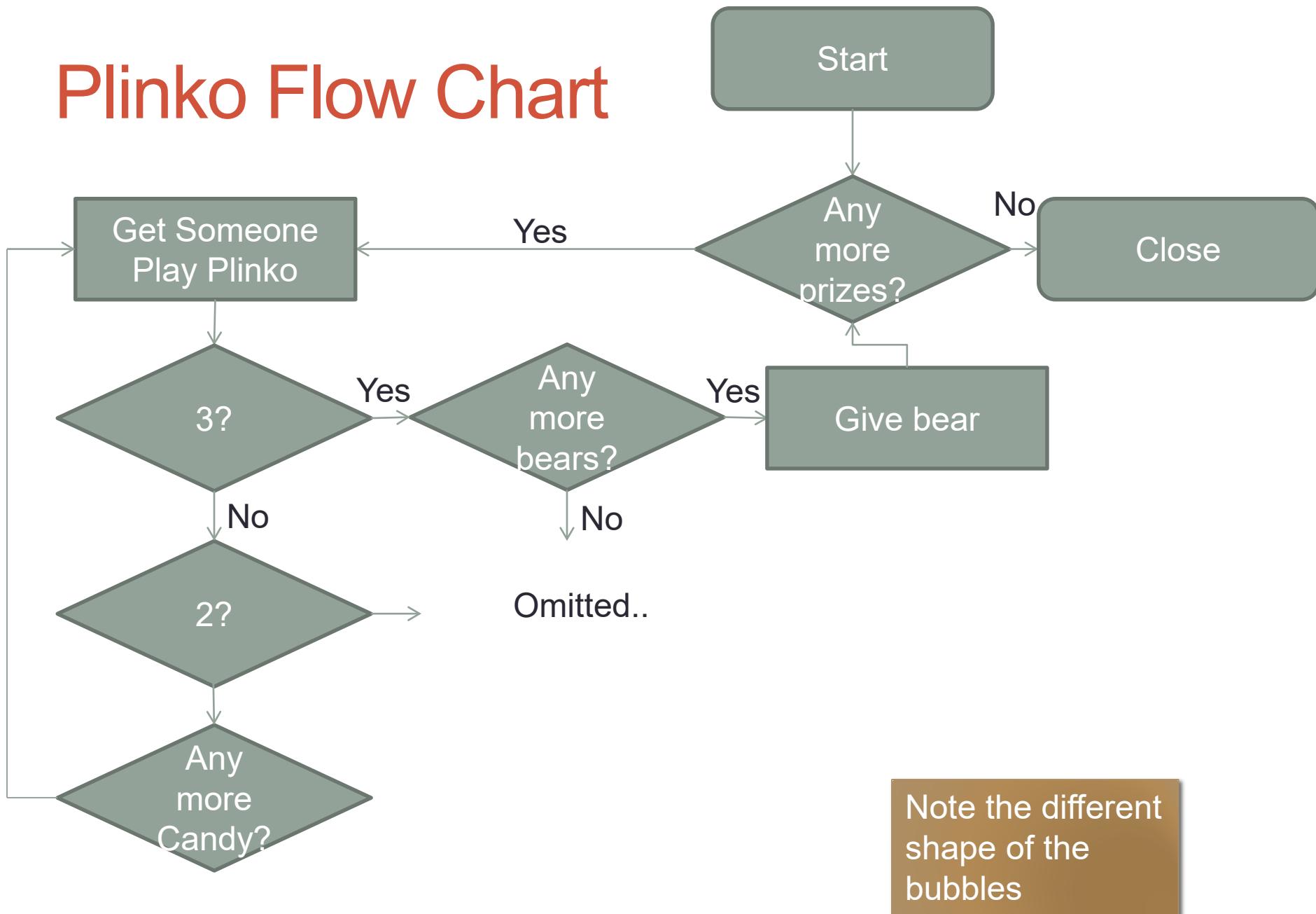
This is called  
**pseudocode**

But still not perfect

# Before Real Coding

- A Program can be expressed by
  - **Pseudocode**
    - An artificial and informal language that helps programmers develop algorithms.
    - "text-based"
  - Or A **Flow Chart**

# Plinko Flow Chart



Note the different  
shape of the  
bubbles

# Choose Your Own Adventure

From Wikipedia, the free encyclopedia

This article is about the trademarked book series. For the genre, see [Gamebook](#). For the TV series, see [Lawrence Leung's Choose Your Own Adventure](#).

**Choose Your Own Adventure** is a series of children's [gamebooks](#) where each story is written from a second-person point of view, with the reader assuming the role of the protagonist and making choices that determine the main character's actions and the plot's outcome. The series was based upon a concept created by [Edward Packard](#) and originally published by Constance Cappel's and [R. A. Montgomery](#)'s Vermont Crossroads Press as the "Adventures of You" series, starting with Packard's *Sugarcane Island* in 1976.<sup>[1]</sup>

*Choose Your Own Adventure*, as published by [Bantam Books](#), was one of the most popular children's series during the 1980s and 1990s, selling more than 250 million copies between 1979 and 1998.<sup>[2]</sup> When Bantam, now owned by [Random House](#), allowed the *Choose Your Own Adventure* trademark to lapse, the series was relaunched by [Chooseco](#), which now owns the trademark. Chooseco does not reissue titles by Packard, who has started his own imprint, U-Ventures.<sup>[3]</sup>

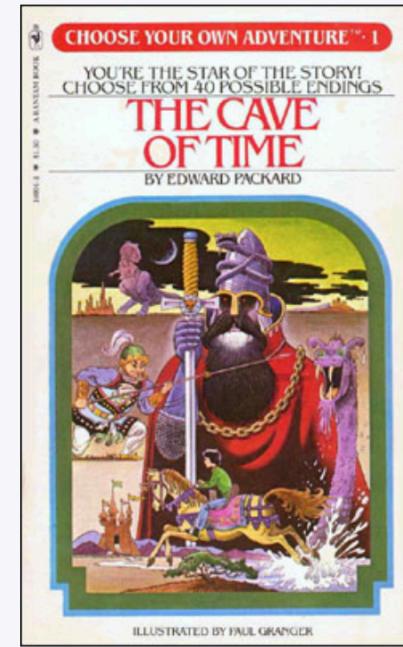
## Contents [hide]

- 1 Format
- 2 History
- 3 See also
- 4 References
- 5 External links

## Format [edit]

Originally created for 7- to 14-year-olds, the books are written in the second person. The protagonist—that is, the reader—takes on a role relevant to the adventure; for example, private investigator, mountain climber, race car driver, doctor, or spy. Stories are generally gender and race neutral, though in some cases, particularly in illustrations, presumption of a male

## Choose Your Own Adventure



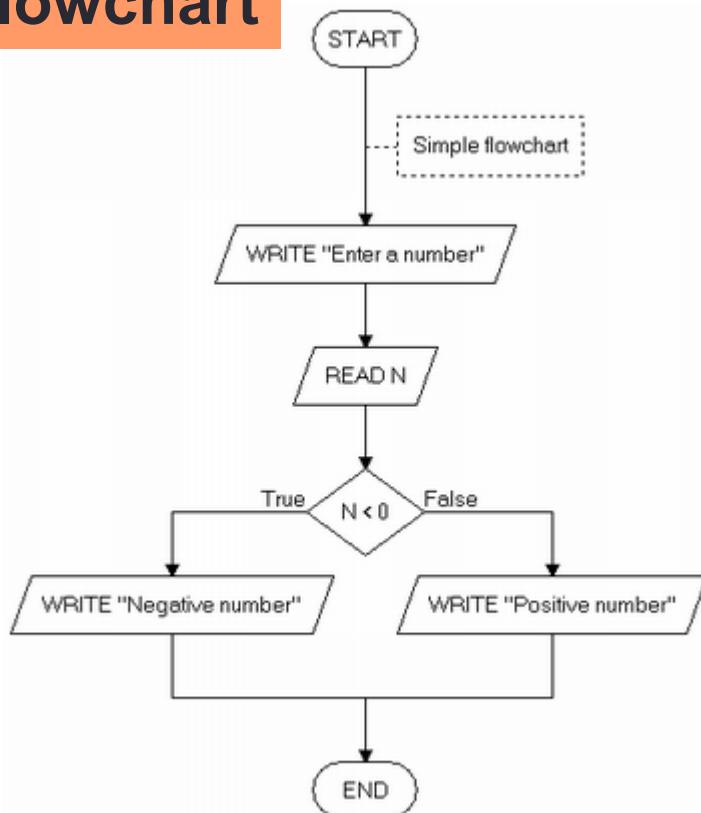
*The Cave of Time* by Edward Packard, the first book in the series

Cover artist Paul Granger  
Language English

# Algorithm

## ■ Ways of representing an algorithm:

### Flowchart



### Pseudocode

#### PSEUDOCODE

set total to zero

get list of numbers

loop through each number in the list  
    add each number to total  
end loop

if number more than zero  
    print "it's positive" message  
else  
    print "it's zero or less" message  
end if

1ynda.com

# Algorithms

- Named for al-Khwārizmī (780-850)
  - Persian mathematician
- Many ancient algorithms
  - Multiplication: Rhind Papyrus
    - Babylon and Egypt: ~1800BC
  - Euclidean Algorithm: Elements
    - Greece: ~300BC
  - Sieve of Eratosthenes
    - Greece: ~200BC



[https://youtu.be/dsZP0iSWj\\_A](https://youtu.be/dsZP0iSWj_A)



# What is the difference between

## Algorithm vs Program

- Algorithm
  - Ideas
  - Machine independent

- Program
  - The final code on a machine
  - Machine dependent

## Research Idea vs Thesis

- Research Idea
  - Can be drawings, sketches, concepts, in any languages
- Thesis
  - Well-written documents in any **languages**, English, German, Chinese, etc...

# AN OVERVIEW OF

---



# Programming Languages History

- <https://james-iry.blogspot.sg/2009/05/brief-incomplete-and-mostly-wrong.html>



THURSDAY, MAY 7, 2009

## A Brief, Incomplete, and Mostly Wrong History of Programming Languages

1801 - Joseph Marie Jacquard uses punch cards to instruct a loom to weave "hello, world" into a tapestry. Redditors of the time are not impressed due to the lack of tail call recursion, concurrency, or proper capitalization.

1842 - Ada Lovelace writes the first program. She is hampered in her efforts by the minor inconvenience that she doesn't have any actual computers to run her code. Enterprise architects will later relearn her techniques in order to program in UML.

1936 - Alan Turing invents every programming language that will ever be but is changed into British Intelligence to be eat before he can



---

### ABOUT ME

 JAMES IRY

SAN FRANCISCO, CA, UNITED STATES

# Why are we learning Python?

- Clear and readable syntax
- Intuitive
- Natural expression
- Powerful
- Popular & Relevant
- Example: Paypal
  - ASF XML Serialization
    - C++
      - 1580 lines
    - Python
      - 130 lines



# Who uses Python?

- Google
- Red Hat
- Dropbox
- Rackspace
- Twitter
- Facebook
- Raspberry Pi
- NASA
- CERN
- ITA
- Yahoo!
- Walt Disney
- IBM
- Reddit
- YouTube

# Python Program without Learning

```
a = 1  
b = 2  
c = a + b  
if c < 0:  
    print('Yes')  
else:  
    print('No')
```

Intuitive!



# Automatic Vs. Manual Transmission: Which is the best choice for you?



# The Environment: IDLE

- We use **IDLE** as an IDE
  - IDE: Integrated development environment
    - Meaning you can edit your program, run and debug it
    - Many different IDEs for different languages
- IDLE stands for
  - Integrated development and learning environment

# A Screenshot of IDLE

Console  
- Input  
- output

The screenshot shows the Python 3.6.0 Shell and a code editor window in the IDLE interface.

**Python 3.6.0 Shell:**

```
Python 3.6.0 (v3.6.0:41df79263a11, Dec 23 2016, 08:06:12) [MSC v.1900 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
```

**Editor and Your program:**

```
hi_graph.py - C:\Users\dcschl\Dropbox\Courses\IT1007\re...
File Edit Format Run Options Window Help
from tkinter import *
from math import sin, cos, pi, sqrt, atan2
import time
import atexit, sys, threading

## Increase recursion stack size
sys.setrecursionlimit(2**20)

## Configuration
WINDOW_SIZE = 512
BORDER_OFFSET = 8

## Some derived values
CANVAS_SIZE = {'x': BORDER_OFFSET,
               'y': BORDER_OFFSET,
               'width': WINDOW_SIZE-2*BORDER_OFFSET,
               'height': WINDOW_SIZE-2*BORDER_OFFSET}

## Some helper functions
def identity(x):
    return x

def composed(f, g):
    return lambda x: f(g(x))

def repeated(f, n):
    if (n == 0):
        return identity
    return composed(f, repeated(f, n-1))

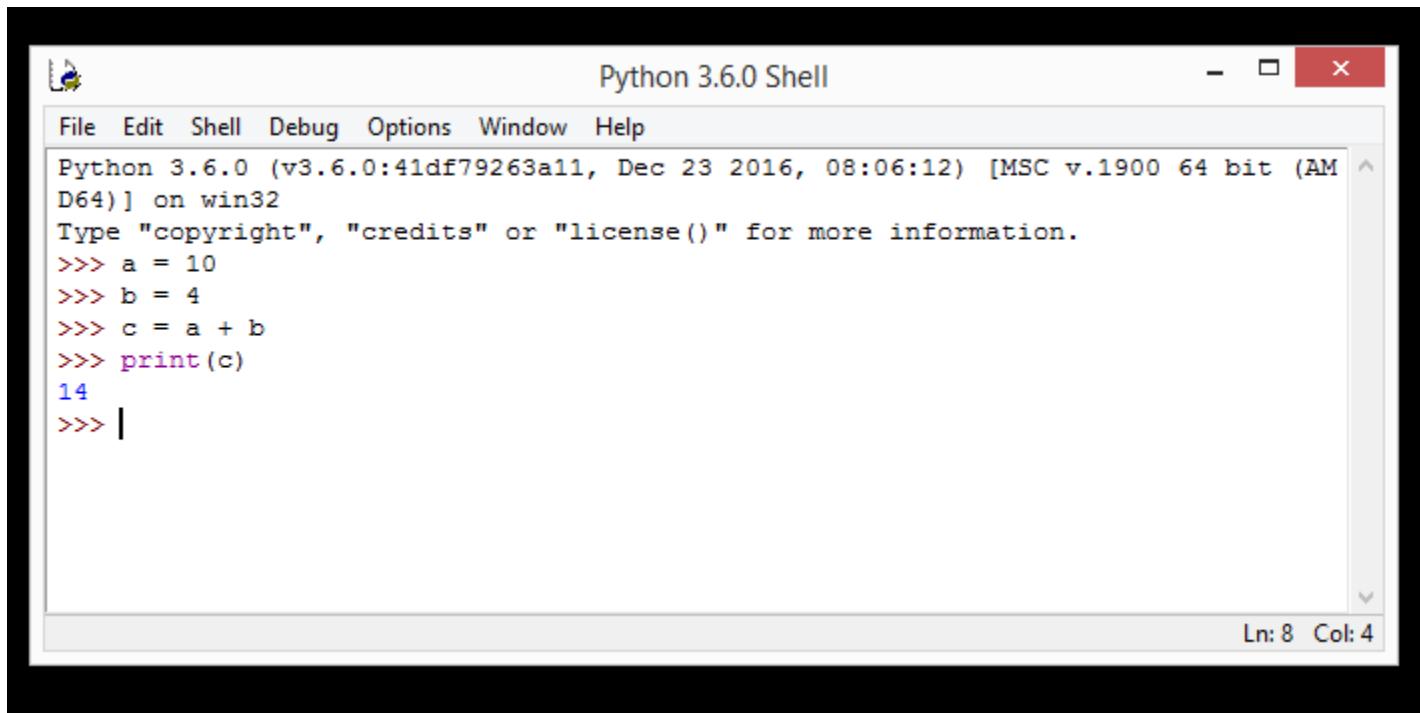
## Data abstraction
```

Annotations:

- An arrow points from the text "Console" to the Python Shell window.
- An arrow points from the text "Editor and Your program" to the code editor window.
- A callout box in the bottom-left corner contains the text "Let's go for some demo".

# You can

- Directly type into the console



A screenshot of the Python 3.6.0 Shell window. The title bar reads "Python 3.6.0 Shell". The menu bar includes File, Edit, Shell, Debug, Options, Window, and Help. The main window displays the following text:

```
Python 3.6.0 (v3.6.0:41df79263a11, Dec 23 2016, 08:06:12) [MSC v.1900 64 bit (AM  
D64)] on win32  
Type "copyright", "credits" or "license()" for more information.  
>>> a = 10  
>>> b = 4  
>>> c = a + b  
>>> print(c)  
14  
>>> |
```

The status bar at the bottom right shows "Ln: 8 Col: 4".

- In which, we **seldom** do this

# Or Run a file

The screenshot shows two windows. On the left is the Python 3.6.0 Shell window with the title bar "Python 3.6.0 Shell". The console output shows:

```
Python 3.6.0 (v3.6.0:41df79263a11, Dec 23 2016, 08:06:12)
D64) on win32
Type "copyright", "credits" or "license()" for more information
>>>
```

On the right is the IDLE editor window with the title bar "test1.py - C:\Users\dcschl\Desktop\test1.py (3.6.0)". The code in the editor is:

```
a = 3
b = 1
c = a + b
if c < 0:
    print('c < 0')
else:
    print('c > 0')
```

The screenshot shows the Python 3.6.0 Shell window with the title bar "Python 3.6.0 Shell". The console output shows:

```
Python 3.6.0 (v3.6.0:41df79263a11, Dec 23 2016, 08:06:12)
D64) on win32
Type "copyright", "credits" or "license()" for more information
>>>
=====
RESTART: C:\Users\dcschl\Desktop\test1.py
c > 0
>>>
```

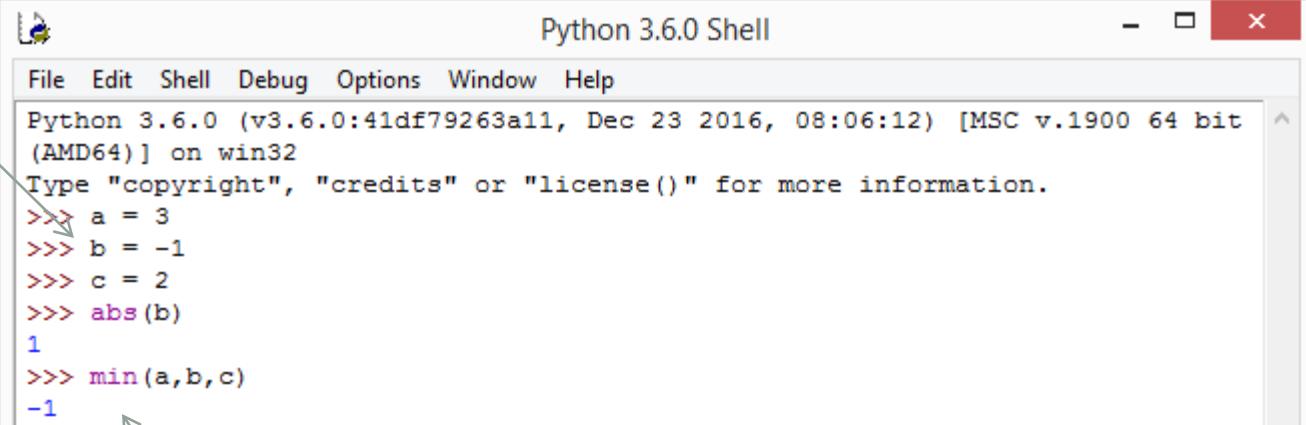
A red oval highlights the output "c > 0". A grey arrow points from this highlighted area to the word "Output" in the text below.

Output of your

IPNO CSNC 100

# Functions and Variables

a, b, c  
are  
variables



The screenshot shows the Python 3.6.0 Shell window. The title bar reads "Python 3.6.0 Shell". The menu bar includes File, Edit, Shell, Debug, Options, Window, and Help. The main window displays the following Python session:

```
Python 3.6.0 (v3.6.0:41df79263a11, Dec 23 2016, 08:06:12) [MSC v.1900 64 bit
(AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.

>>> a = 3
>>> b = -1
>>> c = 2
>>> abs(b)
1
>>> min(a,b,c)
-1
```

abs () and min ()  
are functions

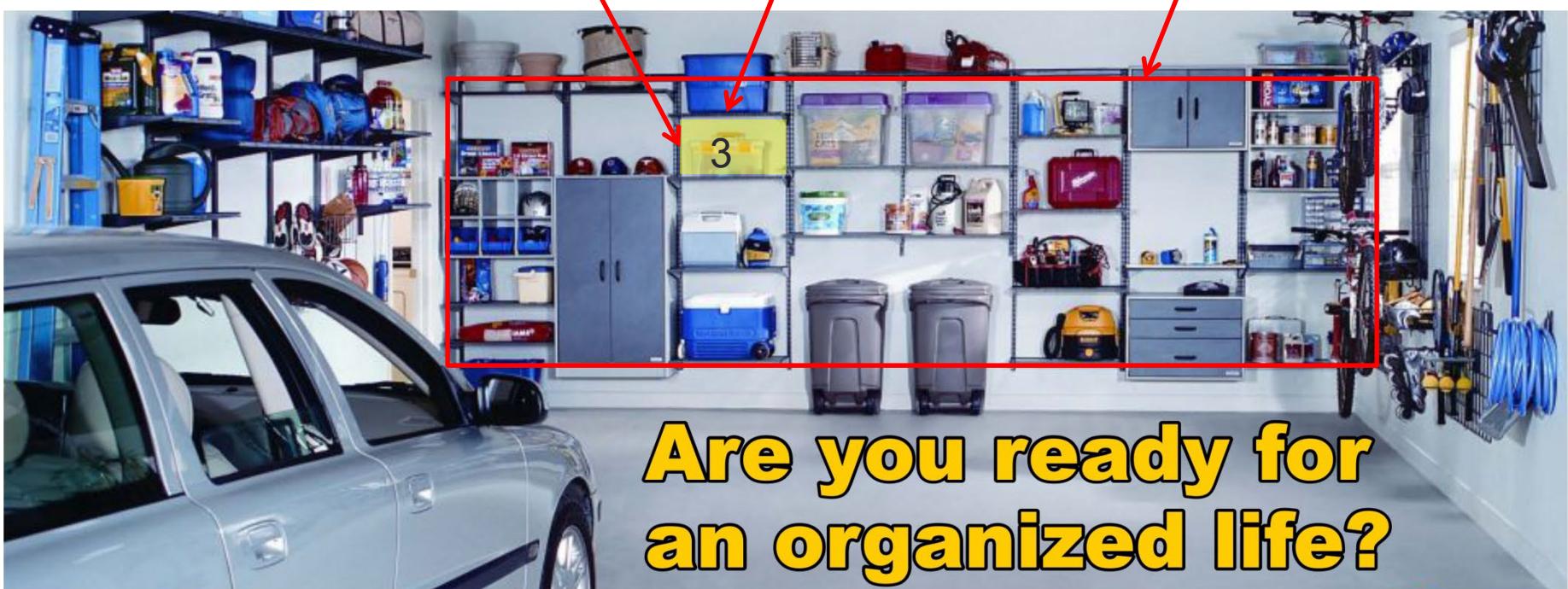
# Variables

- For storage of data

```
>>> x = 3
```

This space is allocated and labeled as “x”. And we store ‘3’ inside

Computer Memory



**Are you ready for  
an organized life?**

# Variable Naming

- Start with 'a'-'z' or 'A'-'Z' or '\_'
- Contain only alphanumeric characters or '\_'
- Case sensitive

x\_1 != x\_1

- Avoid reserved keywords e.g. if
- Python convention: lower case letters separated by '\_'
  - e.g. count\_change

# What Type Can a Variable Stores?



# Variable Type

```
>>> x = 3
```

```
>>> name = "Alan"
```

This space is allocated and labeled as “x”. And we store ‘3’ inside. And can store **integers** only

This space is allocated and labeled as “name”. And we store “Alan” inside. And can store **strings** only



# What Type Can a Variable Stores?

8, 45, 123 int

2.71828, 3.14159 , 1.0 float

True, False bool

"so cool"  
'so cool'

None

# The function type(...)

```
>>> type(123)  
<class 'int'>
```

```
>>> type('123')  
<class 'str'>
```

```
>>> type(None)  
<class 'None'>
```

# Type conversion

```
>>> str(123)  
'123'
```

```
>>> float('45.2')  
45.2
```

```
>>> int(23.8)  
23
```

```
>>> int('10 plus 20')  
ValueError!
```

# ASSIGNMENT OPERATION

---

What is “ $x = x + 1$ ”?

# Assignments

```
>>> x = 10  
>>> x = 2  
>>> x = 4  
>>> print(x)
```



- Assignment to a variable:
  - The operator “=” means differently than what we learn in our math
  - Storing a value into a variable
  - If that variable already contained something beforehand, the old value will be replaced by the new one

# Assignments

```
>>> abc = 18
```

```
>>> my_string = 'This is my string'
```

```
>>> x, y = 1, 2
```

Doesn't matter if it's quote  
or double quote

```
>>> a, b, c = 1, 2, 3  
>>> a, b, c = c, b, a  
>>> print(a, b, c)  
???
```



## Arithmetic: + - \* / \*\* // %

```
>>> a = 2 * 3
```

```
>>> a
```

```
6
```

```
>>> 2 ** 3
```

```
8
```

```
>>> 11 / 3
```

```
3.6666666666666665
```

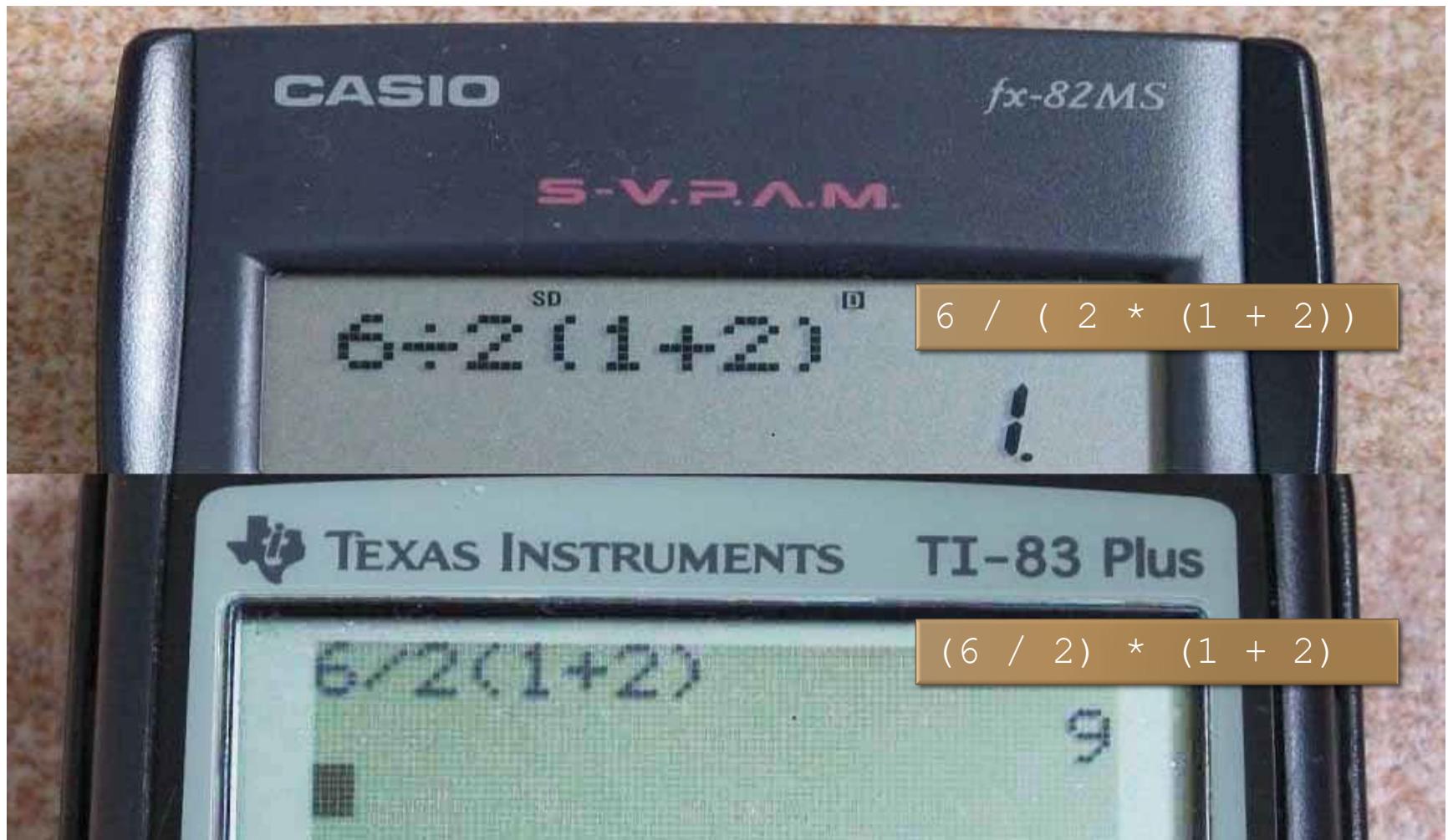
```
>>> 11 // 3
```

```
3
```

```
>>> 11 % 3
```

```
2
```

# Operator Precedence



# Python Operator Precedence

- $6 / 2 * (1+2)$
- $3 * (1+2)$
- $3 * (1+2)$
- $3 \star 3$
- 9

Operator	Description
<code>**</code>	Exponentiation (raise to the power)
<code>~ + -</code>	Complement, unary plus and minus (method names for the last two are <code>+@</code> and <code>-@</code> )
<code>* / % //</code>	Multiply, divide, modulo and floor division
<code>+ -</code>	Addition and subtraction
<code>&gt;&gt; &lt;&lt;</code>	Right and left bitwise shift
<code>&amp;</code>	Bitwise 'AND'>
<code>^  </code>	Bitwise exclusive 'OR' and regular 'OR'
<code>&lt;= &lt; &gt; &gt;=</code>	Comparison operators
<code>&lt;&gt; == !=</code>	Equality operators
<code>= %= /= //= -=</code> <code>+= *= **=</code>	Assignment operators
<code>is is not</code>	Identity operators
<code>in not in</code>	Membership operators
<code>not or and</code>	Logical operators

# What Type Can a Variable Stores?

8, 45, 123 int

2.71828, 3.14159 , 1.0 float

True, False bool

"so cool"  
'so cool'

None

# Boolean: Truth values

- Statements can be either **True** or **False**
- $2 > 1$  is True
- $5 < 3$  is False

# Operators

- Comparison:

```
>>> 1 <= 10
```

True

```
>>> 5 > 15
```

False

```
>>> 5 <= 5
```

True

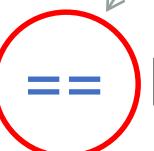
```
>>> 2 != 3  
True
```

```
>>> '1' == 1  
False
```

```
>>> False == False  
True
```

```
>>> True != True  
False
```

The very no. 1  
trap for  
programmers



# Operators

- Logic:

```
>>> True or False
```

```
True
```

```
>>> True and False
```

```
False
```

```
>>> not False
```

```
True
```

- a **or** b True if either **a** or **b** is True
- a **and** b True if both **a** and **b** are True
- **not** a True if a is **not** True

# Truth Tables

A	NOT A
True	False
False	True

A OR B		A	
		True	False
B	True	True	True
	False	True	False

A AND B		A	
		True	False
B	True	True	False
	False	False	False

# More about Truth Values

- Python has keywords `True` and `False`
- In Python 3.x, `True` and `False` will be equal to `1` and `0`
- Anything that is not `0` or `empty` will be evaluated as `True`
- Logic:

```
>>> True and 0  
0  
>>> not 'abc'  
False  
>>> 1 or 0  
1
```

# Strings

```
>>> s = 'ba'          >>> 'z' in t  
>>> t = 'ck'          False  
>>> s + t            >>> 'bananb' > t  
'back'               True  
>>> t = s + 'na' * 2 >>> 'banan' <= t  
>>> t                 True  
'banana'             >>> 'c' < t  
                           False
```

lexicographical ordering: first the first two letters are compared, and if they differ this determines the outcome of the comparison; if they are equal, the next two letters are compared, and so on, until either sequence is exhausted.

# Strings

```
>>> w = 'banana'          >>> s = (w+' ') * 2  
>>> s = w + w            >>> print(s)  
>>> print(s)              'banana banana '  
'bananabanana'  
>>> s = w * 3  
>>> print(s)  
'bananabananabana'
```

# Strings

- A String is a sequence of characters
- We can index a string, i.e.

```
>>> s = 'abcd'
```

```
>>> s[0]
```

```
'a'
```

```
>>> s[2]
```

```
'c'
```

- The index of the first character is 0

# String Slicing (Talk more in tutorials)

`s[start:stop:step]`

Non-inclusive

```
>>> s = 'abcdef'
```

```
>>> s[0:2]
```

```
'ab'
```

```
>>> s[1:2]
```

```
'b'
```

```
>>> s[:2]
```

```
'ab'
```

```
>>> s[1:5:3]
```

```
'be'
```

```
>>> s[::-2]
```

```
'ace'
```

```
>>> s[::-1]
```

```
???
```

Default  
start = 0  
stop = #letters  
step = 1

# Strings

```
c = "#"
```

```
s = " "
```

```
print(" ")                      *
print(" *")                     #
print(s*3 + c)                  ###
print(s*2 + c * 3)              #####
print(s + c * 5)                #
print(s*3 + c)                  ###
print(s*2 + c * 3)              #####
print(s + c * 5)                ######
print(c * 7)                    #
print(s*3 + c)
```

# ASCII Art

(c) .-. (c)  
/ . \_ \  
\_ \ ( Y ) / \_  
( \_ . - / ' - ' \ - . \_ )  
| | X | |  
( \_ . / ^ - ' \ - . )  
` - '

@~t@  
@~~t@ @@@@  
@~(t@ %^^^@  
@((tt@s^//^@  
@tt@s///@  
@ttC@s^/^@  
@CCC@tts^s@ @@@@G// / / / @@  
@O^CC@%ttst@@ /~~~~//@  
@O //(((((/^ ~~~~~//@  
@O //(@@@(((^ ~~~//(/ /@  
@O^ /( @ @(((^ ~~~// / /@  
@^ ^ / @ @ @(( / ^ ~~~/t(/ @  
@ ^ (( / / / ^ ~~~/t(( @ @  
@ = (( / / / ^ ((~~t/t(( @ @  
@% % ( ( ( ( % % @  
@ % s ( ( ( ( % % % @  
@ t @ @ 00% / / / / / @  
@ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @

# FUNCTIONS

---

Abstraction

# Back to this ASCII Art

```
c = "#"
s = " "
print(" ")
print(" *")
print(s*3 + c)
print(s*2 + c * 3)
print(s + c * 5)
print(s*3 + c)
print(s*2 + c * 3)
print(s + c * 5)
print(s*3 + c)
```

# What if I want this?

```
c = "#"  
s = " "  
  
print(" ")  
print(" *")  
print(s*3 + c)  
print(s*2 + c * 3)  
print(s + c * 5)  
  
print(s*3 + c)  
print(s*2 + c * 3)  
print(s + c * 5)  
  
print(s*3 + c)  
print(s*2 + c * 3)  
print(s + c * 5)  
  
print(s*3 + c)  
print(s*2 + c * 3)  
print(s + c * 5)  
  
print(s*3 + c)
```



```
*  
#  
###  
#####  
#  
###  
#####  
#  
###  
#####  
#  
###  
#####  
#
```

# FUNCTION ABSTRACTION

---

# What if I want this?

```
c = "#"
s = " "
print(" ")
print("    *")
print(s*3 + c)
print(s*2 + c * 3)
print(s + c * 5)
print(s*3 + c)
print(s*2 + c * 3)
print(s + c * 5)
print(s*3 + c)
print(s*2 + c * 3)
print(s + c * 5)
print(s*3 + c)
print(s*2 + c * 3)
print(s + c * 5)
print(s*3 + c)
```

Same

```
*  
#  
###  
#####  
#  
###  
#####  
#  
###  
#####  
#  
###  
#####  
#
```

# Function Abstraction

```
c = "#"  
s = " "  
  
print(" ")  
print(" *")  
print(s*3 + c)  
print(s*2 + c * 3)  
print(s + c * 5)  
  
print(s*3 + c)  
print(s*2 + c * 3)  
print(s + c * 5)  
  
print(s*3 + c)  
print(s*2 + c * 3)  
print(s + c * 5)  
  
print(s*3 + c)  
print(s*2 + c * 3)  
print(s + c * 5)  
  
print(s*3 + c)
```



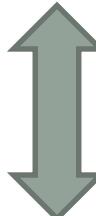
Give this part of the  
code a name:  
E.g.  
“drawTriangle”

\*  
#  
###  
#####  
#  
###  
#####  
#  
###  
#####  
#  
###  
#####  
#

# Function Abstraction

c = "#" \*  
s = " "  
  
print(" ") #  
print(" \*") #####  
  
drawTriangle() #####  
  
drawTriangle() #  
drawTriangle() ###  
drawTriangle() #####  
  
drawTriangle() #  
  
print(s\*3 + c) #  
print(s\*2 + c \* 3) ###  
print(s + c \* 5) #####  
  
#  
###  
#####  
  
#  
###  
#####  
  
#

By “drawTriangle()” we mean:



# Compare

```
c = "#"  
s = " "
```

```
def drawTriangle():  
    print(s*3 + c)  
    print(s*2 + c * 3)  
    print(s + c * 5)
```

```
print(" ")  
print(" *")  
drawTriangle()  
drawTriangle()  
drawTriangle()  
drawTriangle()  
print(s*3 + c)
```

```
c = "#"  
s = " "  
  
print(" ")  
print(" *")  
print(s*3 + c)  
print(s*2 + c * 3)  
print(s + c * 5)  
print(s*3 + c)  
print(s*2 + c * 3)  
print(s + c * 5)  
print(s*3 + c)  
print(s*2 + c * 3)  
print(s + c * 5)  
print(s*3 + c)  
print(s*2 + c * 3)  
print(s + c * 5)  
print(s*3 + c)
```

# Abstraction in Daily Life: Cooking Rice



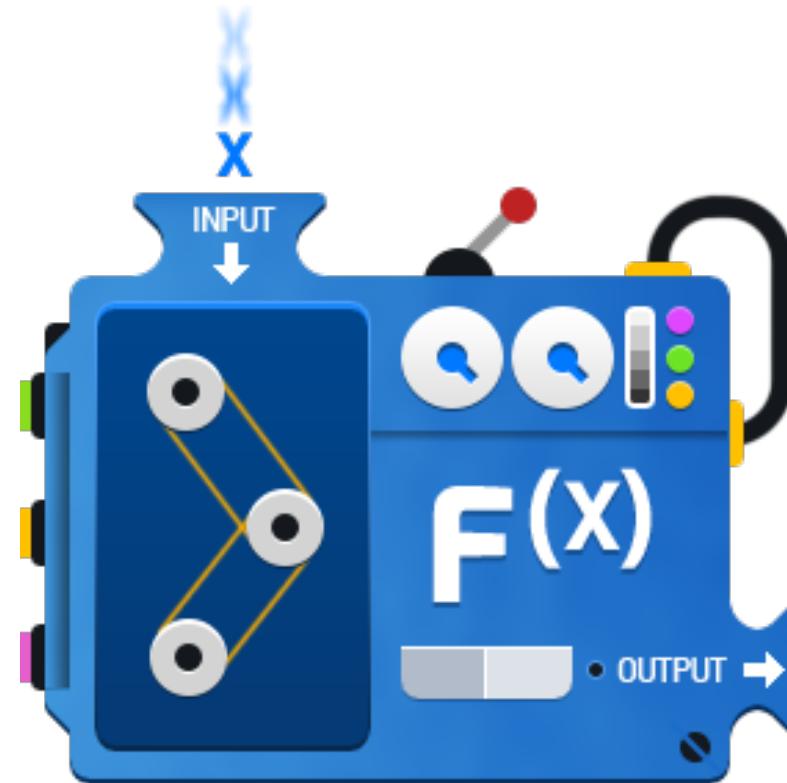
# Abstraction in Daily Life: Cooking Rice

- BBC
  - 1. Measure the rice into a cup and level the top
  - 2. Rinse the rice thoroughly in cold water
  - 3. Pour the rice into a pan over a low heat
  - 4. Add double the amount of water
  - 5. Bring to a boil
  - 6. Put a lid on and turn the heat down to as low as possible
  - 7. Cook for 10 mins and do not take the lid off
  - 8. Fluff the rice with a fork
- Abstraction:



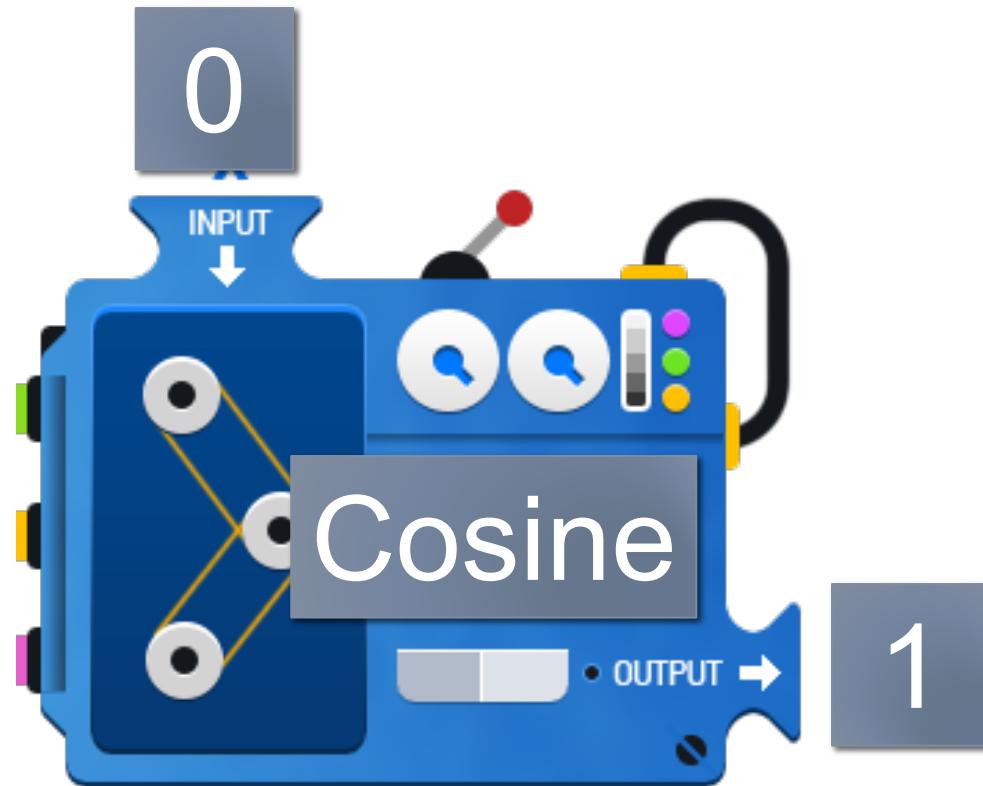
# Functions

- A function is like a black box
  - You put in something for the input
  - And it will produce a new thing for the output



# For example

- “Cosine” is a function
  - Input 0
  - Output 1



# Let's Write Our Own Function!

```
def square(x):  
    return x * x  
  
>>> square(3)  
9  
  
>>> square(square(2))  
16  
  
>>> square(3,4)  
???
```

# Let's Write Our Own Function!

Define  
(keyword)

Function name

Input  
(Argument)

```
def square(x):
```

```
    return x * x
```

Indentation

Output

# Another one....

```
def singHappyBirthdayTo(name):  
    print('Happy birthday To You!')  
    print('Happy birthday To You!')  
    print('Happy birthday to ' + name + '~')  
    print('Happy birthday to You!!!')
```



## Indentation

```
>>> singHappyBirthdayTo('Alan')  
Happy birthday To You!  
Happy birthday To You!  
Happy birthday to Alan~  
Happy birthday to You!!!
```

# What if

A function



```
def singHappyBirthdayTo(name):  
    print('Happy birthday To You! ')  
    print('Happy birthday To You! ')  
    print('Happy birthday to ' + name + '~~')  
  
print('Happy birthday to You!!!')
```



Execute an extra line **OUT**  
of the function

# Indentation Matters

```
square.py - C:/Users/dcschl/Google Drive/Courses
File Edit Format Run Options Window Help
def square(x):
    return x * x

def singHappyBirthdayTo(name):
    print('Happy birthday To You!')
    print('Happy birthday To You!')
    print('Happy birthday to ' + name + '~')
    print('Happy birthday to You!!!!')

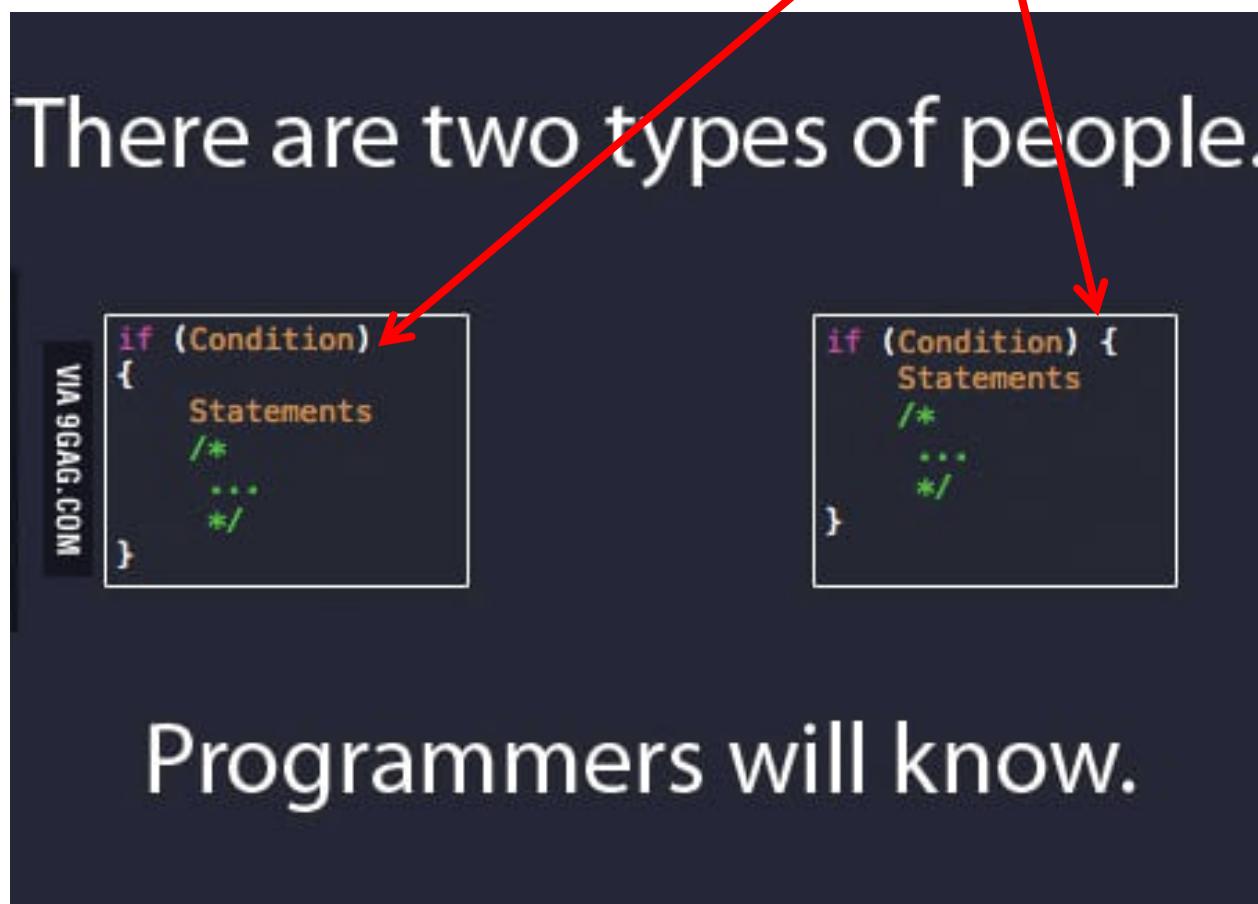
print(square(3))

singHappyBirthdayTo('Alan')
```

- This is a “Python thing”
  - Meaning, in other languages, usually spaces, tabs, new lines do not affect the code

# C

- For C language, it doesn't matter

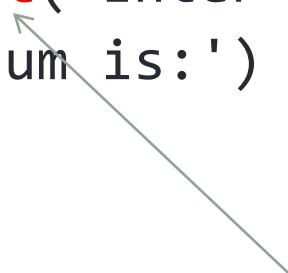


```
function register()
{
    if (!empty($_POST)) {
        $msg = '';
        if ($_POST['user_name']) {
            if ($_POST['user_password_new']) {
                if ($_POST['user_password_new'] === $_POST['user_password_repeat']) {
                    if (strlen($_POST['user_password_new']) > 5) {
                        if (strlen($_POST['user_name']) < 65 && strlen($_POST['user_name']) > 1) {
                            if (preg_match('/^a-zA-Z\d{2,64}$/', $_POST['user_name'])) {
                                $user = read_user($_POST['user_name']);
                                if (!isset($user['user_name'])) {
                                    if ($_POST['user_email']) {
                                        if (strlen($_POST['user_email']) < 65) {
                                            if (filter_var($_POST['user_email'], FILTER_VALIDATE_EMAIL)) {
                                                create_user();
                                                $_SESSION['msg'] = 'You are now registered so please login';
                                                header('Location: ' . $_SERVER['PHP_SELF']);
                                                exit();
                                            } else $msg = 'You must provide a valid email address';
                                        } else $msg = 'Email must be less than 64 characters';
                                    } else $msg = 'Email cannot be empty';
                                } else $msg = 'Username already exists';
                            } else $msg = 'Username must be only a-z, A-Z, 0-9';
                        } else $msg = 'Username must be between 2 and 64 characters';
                    } else $msg = 'Password must be at least 6 characters';
                } else $msg = 'Passwords do not match';
            } else $msg = 'Empty Password';
        } else $msg = 'Empty Username';
        $_SESSION['msg'] = $msg;
    }
    return register_form();
}
```



# Add Two Numbers

```
def addTwoNumbers() :  
    a = int(input('Enter an integer:'))  
    b = int(input('Enter another integer:'))  
    print('The sum is:')
```



```
print(a + b)
```

Keyboard input  
(Built-in function)

```
>>> addTwoNumbers()
```

```
Enter an integer:2
```

```
Enter another integer:3
```

```
The sum is:
```

```
5
```

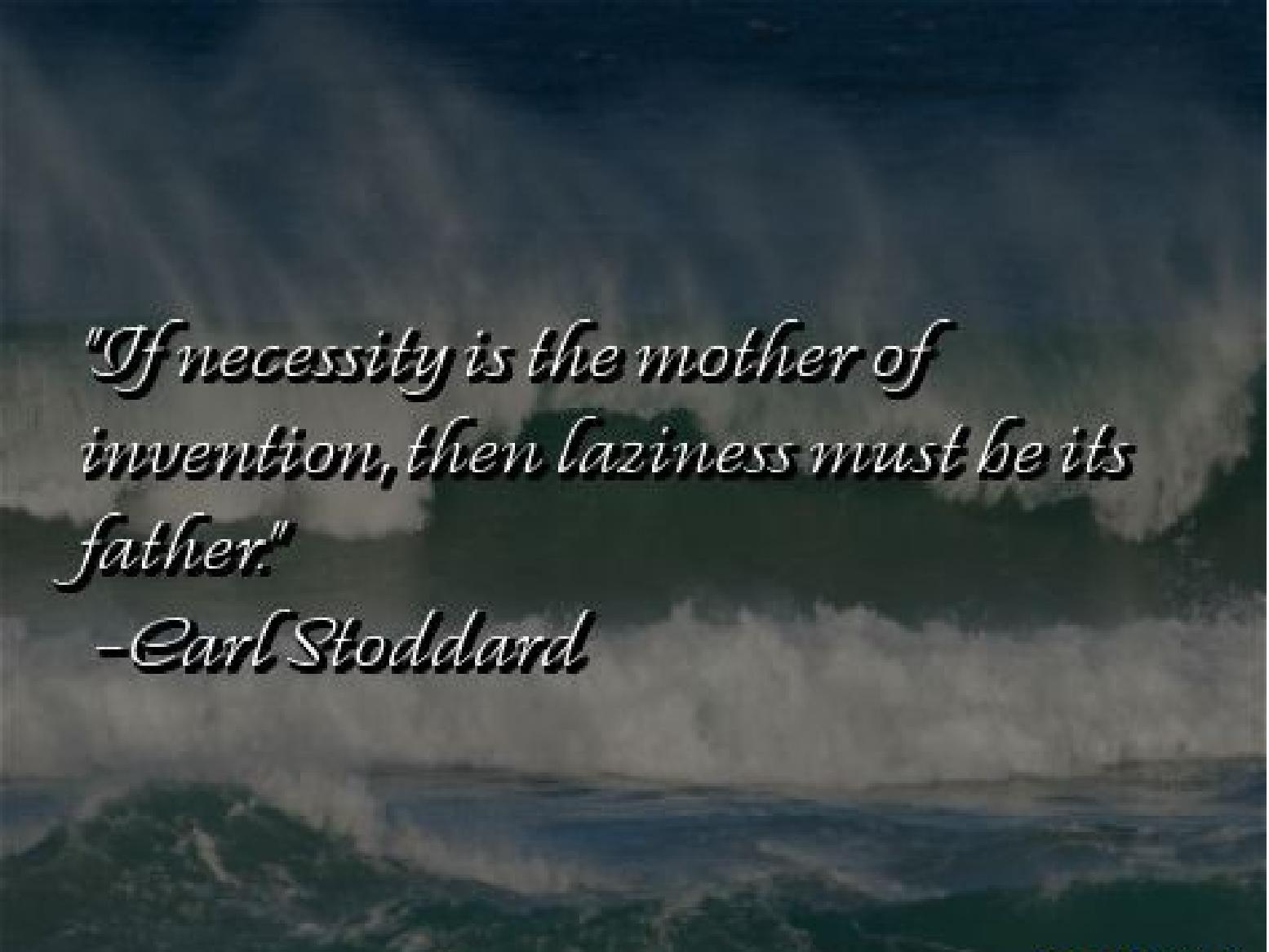
```
>>>
```

# Python Packages

- What if I want to compute  $\cos(\pi/2)$ ?
  - Do I need to write the code for this?

$$\cos(x) = \underbrace{\frac{1}{0!}}_{n=0} - \underbrace{\frac{x^2}{2!}}_{n=1} + \underbrace{\frac{x^4}{4!}}_{n=2} - \underbrace{\frac{x^6}{6!}}_{n=3} + \underbrace{\frac{x^8}{8!}}_{n=4} + \dots$$

- Or draw a graph
- Or implement some network features?
- Or other complicated tasks?



*"If necessity is the mother of invention, then laziness must be its father"*

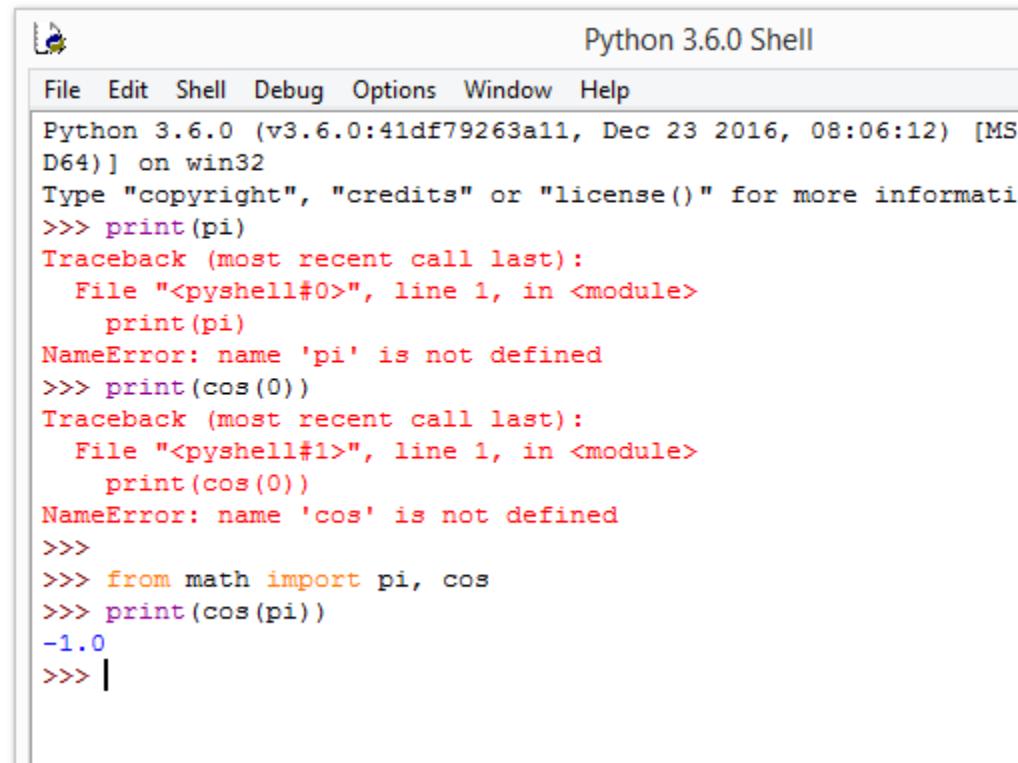
*-Earl Stoddard*

# Different Scenario with Diff. Equipment



# Python Packages (Ready Made Functions)

- Import the **function** `cos` and the **constant** `pi` from the `math` package
  - (Or library)

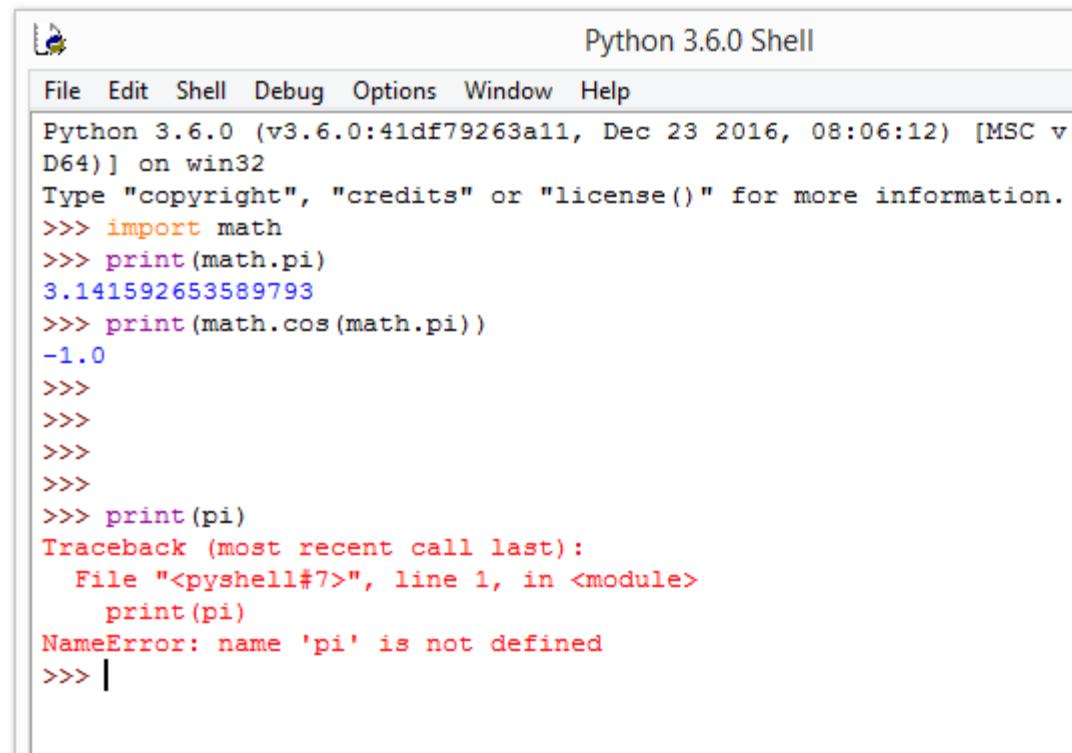


The screenshot shows a Python 3.6.0 Shell window. The menu bar includes File, Edit, Shell, Debug, Options, Window, and Help. The title bar says "Python 3.6.0 Shell". The shell area displays the following code and errors:

```
Python 3.6.0 (v3.6.0:41df79263a11, Dec 23 2016, 08:06:12) [MS
D64] on win32
Type "copyright", "credits" or "license()" for more information
>>> print(pi)
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    print(pi)
NameError: name 'pi' is not defined
>>> print(cos(0))
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    print(cos(0))
NameError: name 'cos' is not defined
>>>
>>> from math import pi, cos
>>> print(cos(pi))
-1.0
>>> |
```

# The Other way to Import cos

- You need at add the prefix “math.” before the function
- Otherwise..



Python 3.6.0 Shell

```
File Edit Shell Debug Options Window Help
Python 3.6.0 (v3.6.0:41df79263a11, Dec 23 2016, 08:06:12) [MSC v.
D64] on win32
Type "copyright", "credits" or "license()" for more information.
>>> import math
>>> print(math.pi)
3.141592653589793
>>> print(math.cos(math.pi))
-1.0
>>>
>>>
>>>
>>>
>>> print(pi)
Traceback (most recent call last):
  File "<pyshell#7>", line 1, in <module>
    print(pi)
NameError: name 'pi' is not defined
>>> |
```

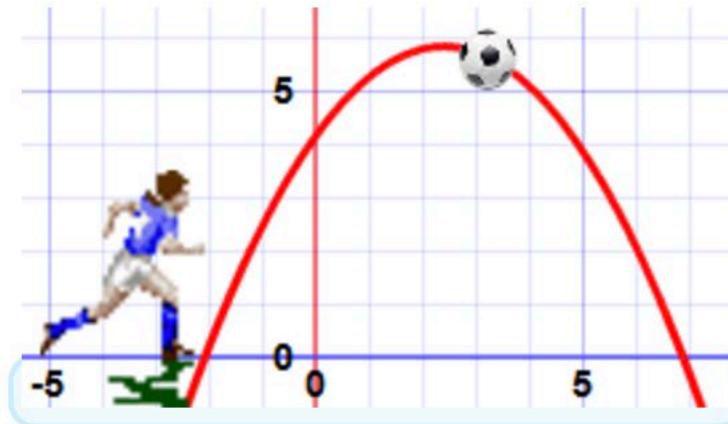
# Example: Solving a Quadratic Eqt.

An example of a **Quadratic Equation**:

$$5x^2 + 3x + 3 = 0$$

*this makes it Quadratic*

Quadratic Equations make nice curves, like this one:



# Example: Solving a Quadratic Eqt.

## Standard Form

The **Standard Form** of a Quadratic Equation looks like this:

$$ax^2 + bx + c = 0$$

- **a**, **b** and **c** are known values. **a** can't be 0.
- "x" is the **variable** or unknown (we don't know it yet).

# Example: Solving a Quadratic Eqt.

- Remember what we learned in high school...

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

- Let's try to implement it in Python



```
from math import sqrt

def solve_qe(a,b,c):
    delta = b**2 - 4*a*c
    ans1 = (-b + sqrt(delta))/(2*a)
    ans2 = (-b - sqrt(delta))/(2*a)
    print("The two solutions are " + str(ans1)
          + " and " + str(ans2))
```

```
>>> solve_qe(1,5,6)
The two solutions are -2.0 and -3.0
>>> solve_qe(1,4,4)
The two solutions are -2.0 and -2.0
>>>
```

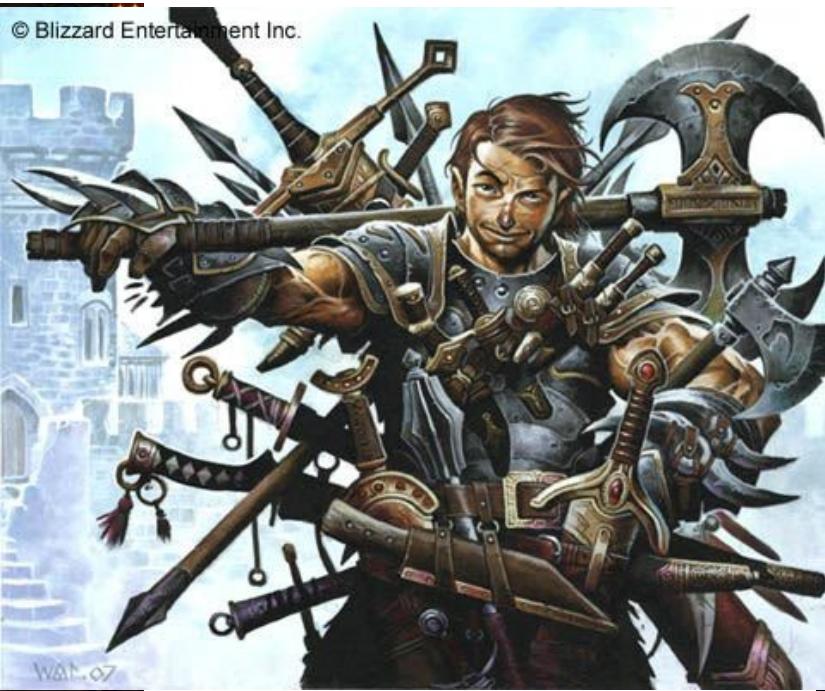
# Which way should I use?

- First, “from math import cos”
  - Make your code shorter
  - Save memory
  - But you need to know exactly which function you have to import
- Second, “import math”
  - You can be more relax to use functions in the whole library
  - But your code will be longer and require more memory

```
>>>
>>> from math import pi, cos
>>> print(cos(pi))
-1.0
>>> |
```

```
-->-->-->
>>> import math
>>> print(math.pi)
3.141592653589793
>>> print(math.cos(math.pi))
-1.0
>>>
```

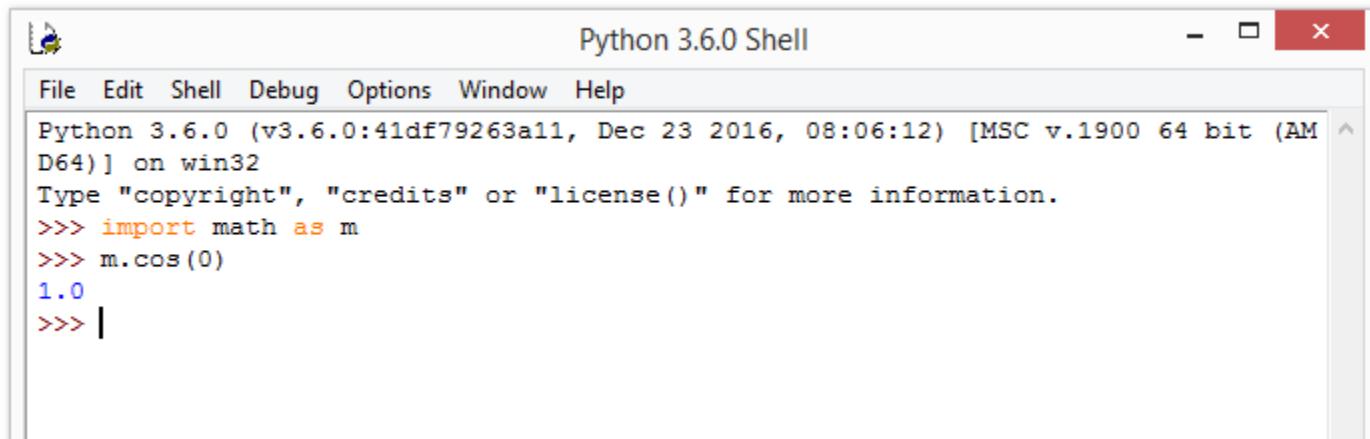
# You don't want to be like this in real life



Try NOT to import  
too many packages



# To Avoid Long Name



The screenshot shows a window titled "Python 3.6.0 Shell". The menu bar includes File, Edit, Shell, Debug, Options, Window, and Help. The main area displays the Python interpreter's welcome message and a command-line session:

```
Python 3.6.0 (v3.6.0:41df79263a11, Dec 23 2016, 08:06:12) [MSC v.1900 64 bit (AM  
D64)] on win32  
Type "copyright", "credits" or "license()" for more information.  
>>> import math as m  
>>> m.cos(0)  
1.0  
>>> |
```

# For More Packages

- Visit
  - <https://docs.python.org/>

The screenshot shows a web browser displaying the Python 3.6.2 Documentation page at <https://docs.python.org/>. The page title is "Python 3.6.2 documentation". On the left, there's a sidebar with links for "Download", "Docs for other versions" (listing Python 2.7, 3.5, 3.7, and Old versions), and "Other resources" (listing PEP Index, Beginner's Guide, Book List, and Audio/Visual Talks). A red oval highlights the "Library Reference" link in the main content area, which is described as "keep this under your pillow". The main content area also includes sections for "What's new in Python 3.6?", "Tutorial", "Language Reference", and "Python Setup and Usage". To the right of the main content, there are links for "Installing Python", "Distributing Python Modules", "Extending and", and "Python/C API".

Keep this under your pillow

Python 3.6.2 documentation

Welcome! This is the documentation for Python 3.6.2.

Parts of the documentation:

- What's new in Python 3.6?
- Tutorial
- Library Reference
- Language Reference
- Python Setup and Usage

Installing Python

Distributing Python Modules

Extending and

Python/C API

# Python Packages

The screenshot shows a web browser window titled "The Python Standard Lib". The address bar displays "Python Software Foundation [US] | https://docs.python.org/3/library/index.html". The page content lists various Python modules under categories:

- 8. Built-in array creation functions
  - 8.7. `array` — Efficient arrays of numeric values
  - 8.8. `weakref` — Weak references
  - 8.9. `types` — Dynamic type creation and names for built-in types
  - 8.10. `copy` — Shallow and deep copy operations
  - 8.11. `pprint` — Data pretty printer
  - 8.12. `reprlib` — Alternate `repr()` implementation
  - 8.13. `enum` — Support for enumerations
- 9. Numeric and Mathematical Modules
  - 9.1. `numbers` — Numeric abstract base classes
  - 9.2. `math` — Mathematical functions
  - 9.3. `cmath` — Mathematical functions for complex numbers
  - 9.4. `decimal` — Decimal fixed point and floating point arithmetic
  - 9.5. `fractions` — Rational numbers
  - 9.6. `random` — Generate pseudo-random numbers
  - 9.7. `statistics` — Mathematical statistics functions
- 10. Functional Programming Modules
  - 10.1. `itertools` — Functions creating iterators for efficient looping
  - 10.2. `functools` — Higher-order functions and operations on callable objects
  - 10.3. `operator` — Standard operators as functions

# Built-in Functions

- Functions that need **NOT** to be imported

Built-in Functions				
<code>abs()</code>	<code>dict()</code>	<code>help()</code>	<code>min()</code>	<code>setattr()</code>
<code>all()</code>	<code>dir()</code>	<code>hex()</code>	<code>next()</code>	<code>slice()</code>
<code>any()</code>	<code>divmod()</code>	<code>id()</code>	<code>object()</code>	<code>sorted()</code>
<code>ascii()</code>	<code>enumerate()</code>	<code>input()</code>	<code>oct()</code>	<code>staticmethod()</code>
<code>bin()</code>	<code>eval()</code>	<code>int()</code>	<code>open()</code>	<code>str()</code>
<code>bool()</code>	<code>exec()</code>	<code>isinstance()</code>	<code>ord()</code>	<code>sum()</code>
<code>bytearray()</code>	<code>filter()</code>	<code>issubclass()</code>	<code>pow()</code>	<code>super()</code>
<code>bytes()</code>	<code>float()</code>	<code>iter()</code>	<code>print()</code>	<code>tuple()</code>
<code>callable()</code>	<code>format()</code>	<code>len()</code>	<code>property()</code>	<code>type()</code>
<code>chr()</code>	<code>frozenset()</code>	<code>list()</code>	<code>range()</code>	<code>vars()</code>
<code>classmethod()</code>	<code>getattr()</code>	<code>locals()</code>	<code>repr()</code>	<code>zip()</code>
<code>compile()</code>	<code>globals()</code>	<code>map()</code>	<code>reversed()</code>	<code>__import__()</code>
<code>complex()</code>	<code>hasattr()</code>	<code>max()</code>	<code>round()</code>	
<code>delattr()</code>	<code>hash()</code>	<code>memoryview()</code>	<code>set()</code>	