TODAY

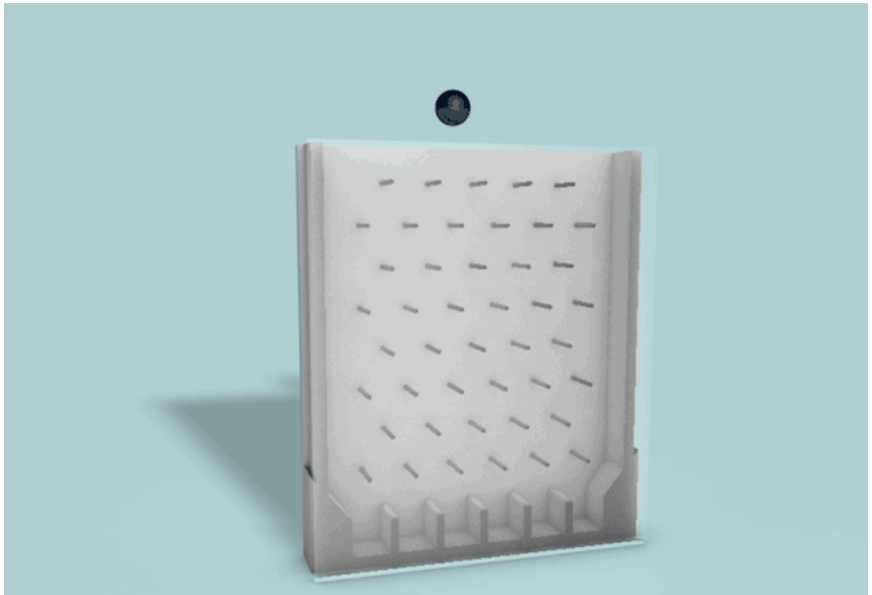
What is programming about?

Why Python?

Imagine

Running a Booth

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

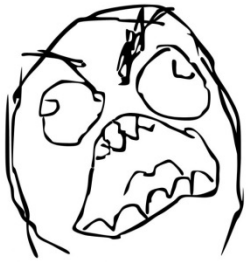


Let's see what we have for prizes

- 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

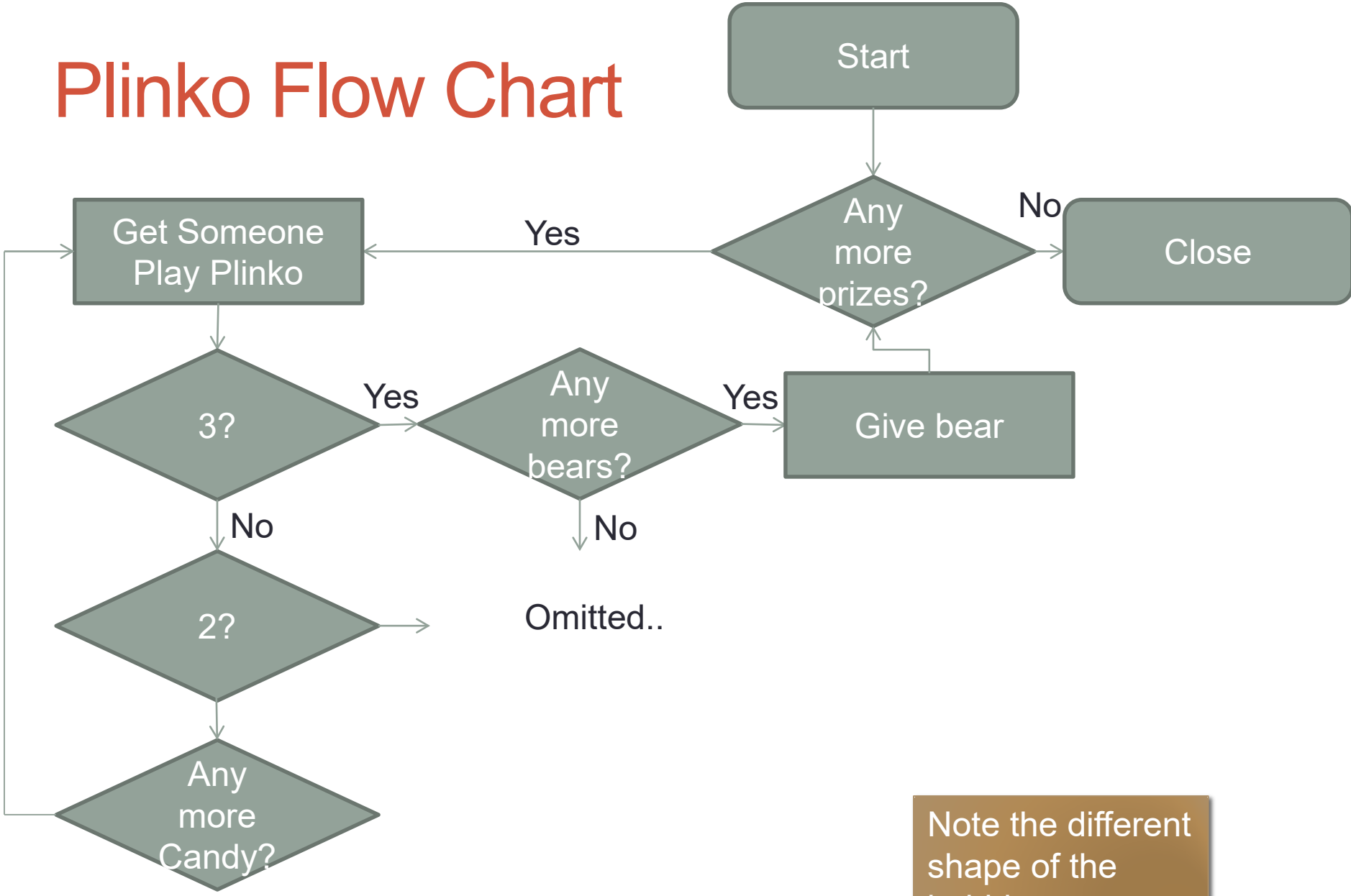
This is called **pseudocode**

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

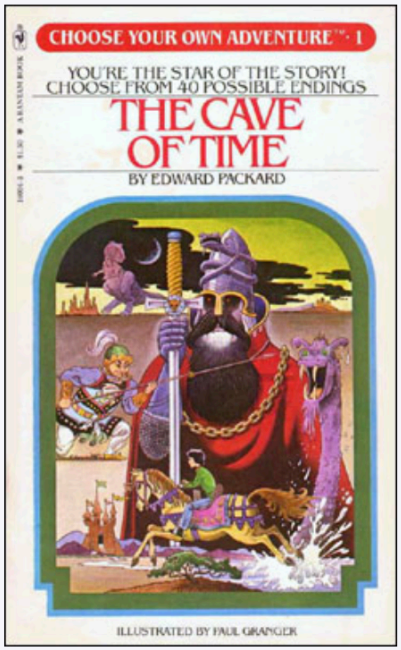
This article is about the trademarked book series. For the genre, see Gamebook. For the TV series, see Lawrence Leung's Choose Your Own Adventure.

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.[1]

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.[2] 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.[3]

Contents [hide]

Choose Your Own Adventure



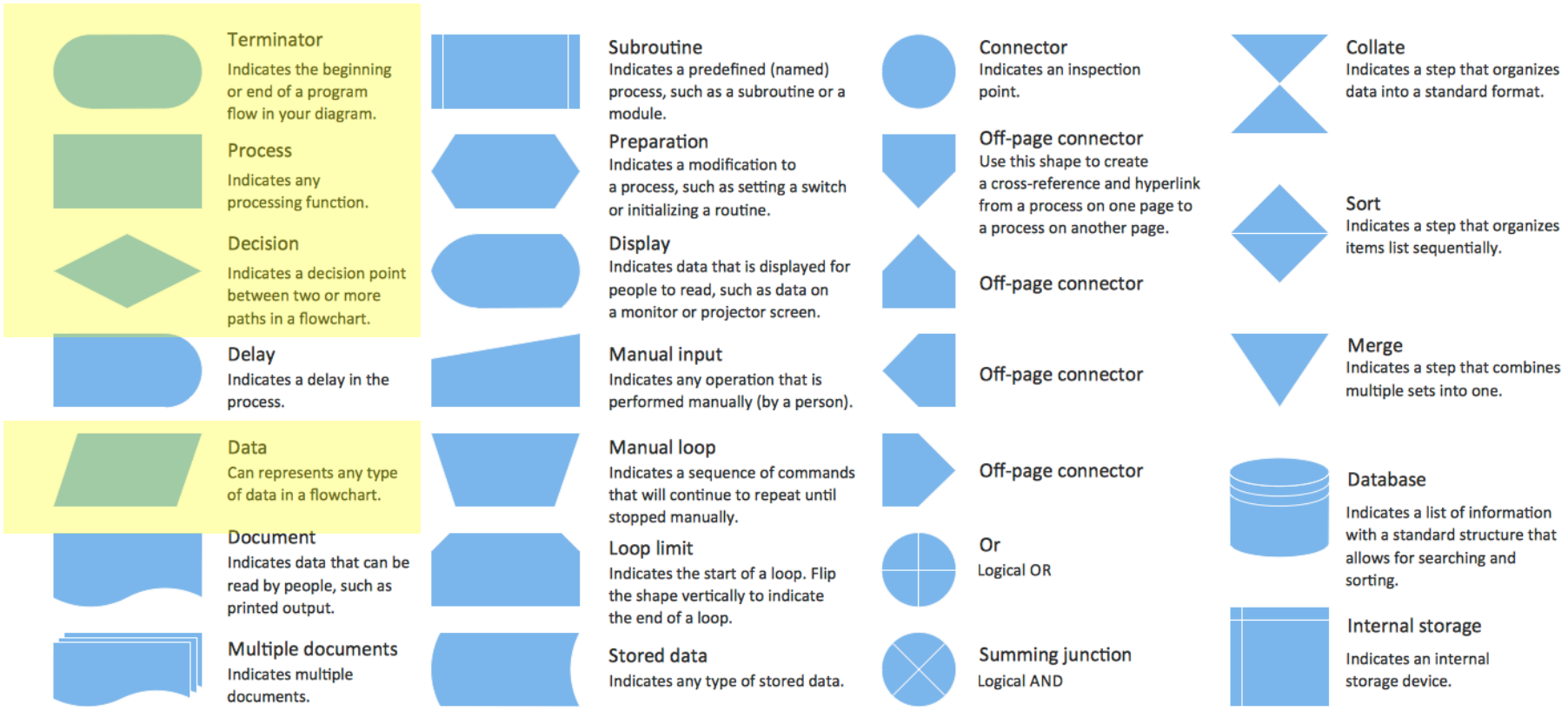
The Cave of Time by Edward Packard, the first book in the series

Cover artist	Paul Granger
Language	English

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

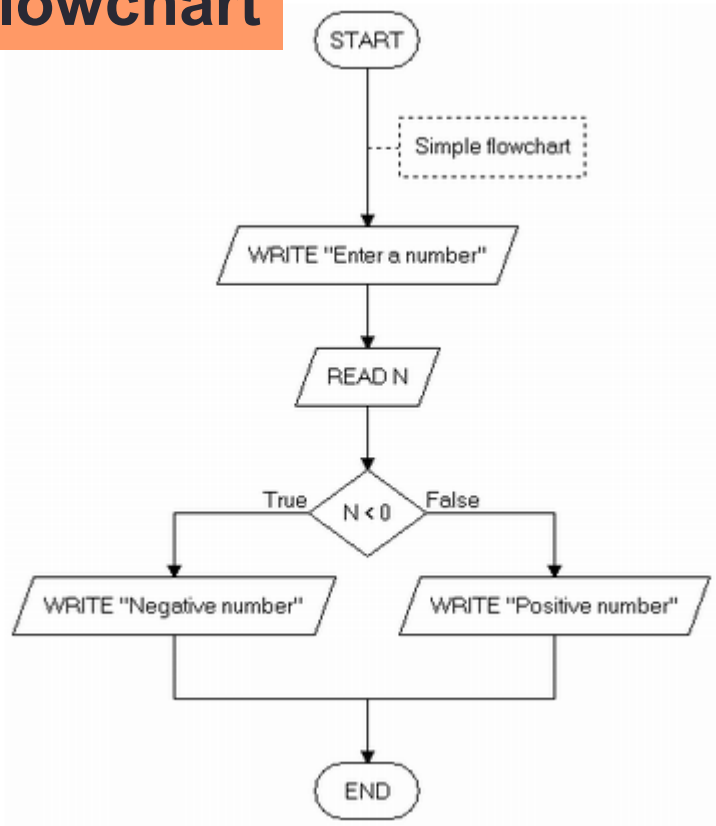
Flow Chart Bubble Types



Algorithm

- Ways of representing an algorithm:

Flowchart



Pseudocode

PSEUDOCODE

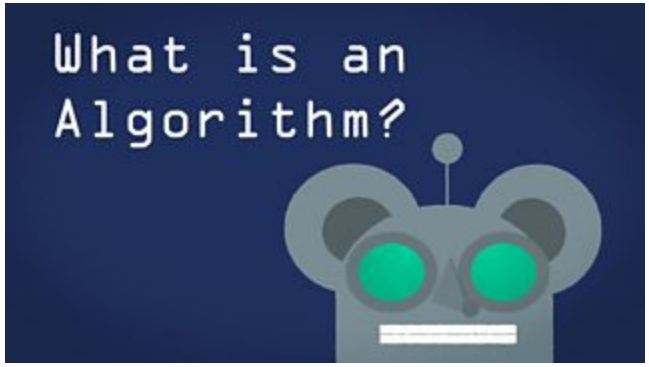
```
set total to zero

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
```

lynda.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



Algorithm *(noun.)*

Word used by programmers when...
they do not want to explain what they did.

Algorithm (1/3)

- An **algorithm** is a well-defined computational procedure consisting of *a set of instructions*, that takes some value or set of values as *input*, and produces some value or set of values as *output*.

'Algorithm' stems from 'Algoritmi', the Latin form of al-Khwārizmī, a Persian mathematician, astronomer and geographer.
Source: http://en.wikipedia.org/wiki/Algorithm

What is the difference between

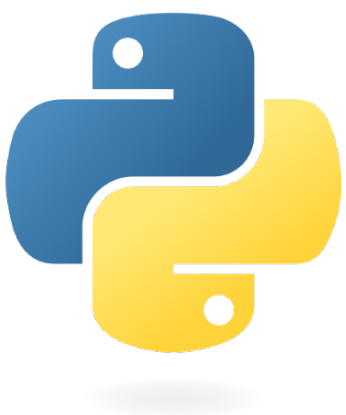
Algorithm vs Program

- 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



python™

Programming Languages History

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

ONE DIV ZERO

AN EXPLORATION OF SOFTWARE DEVELOPMENT.

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. Redditers 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

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!

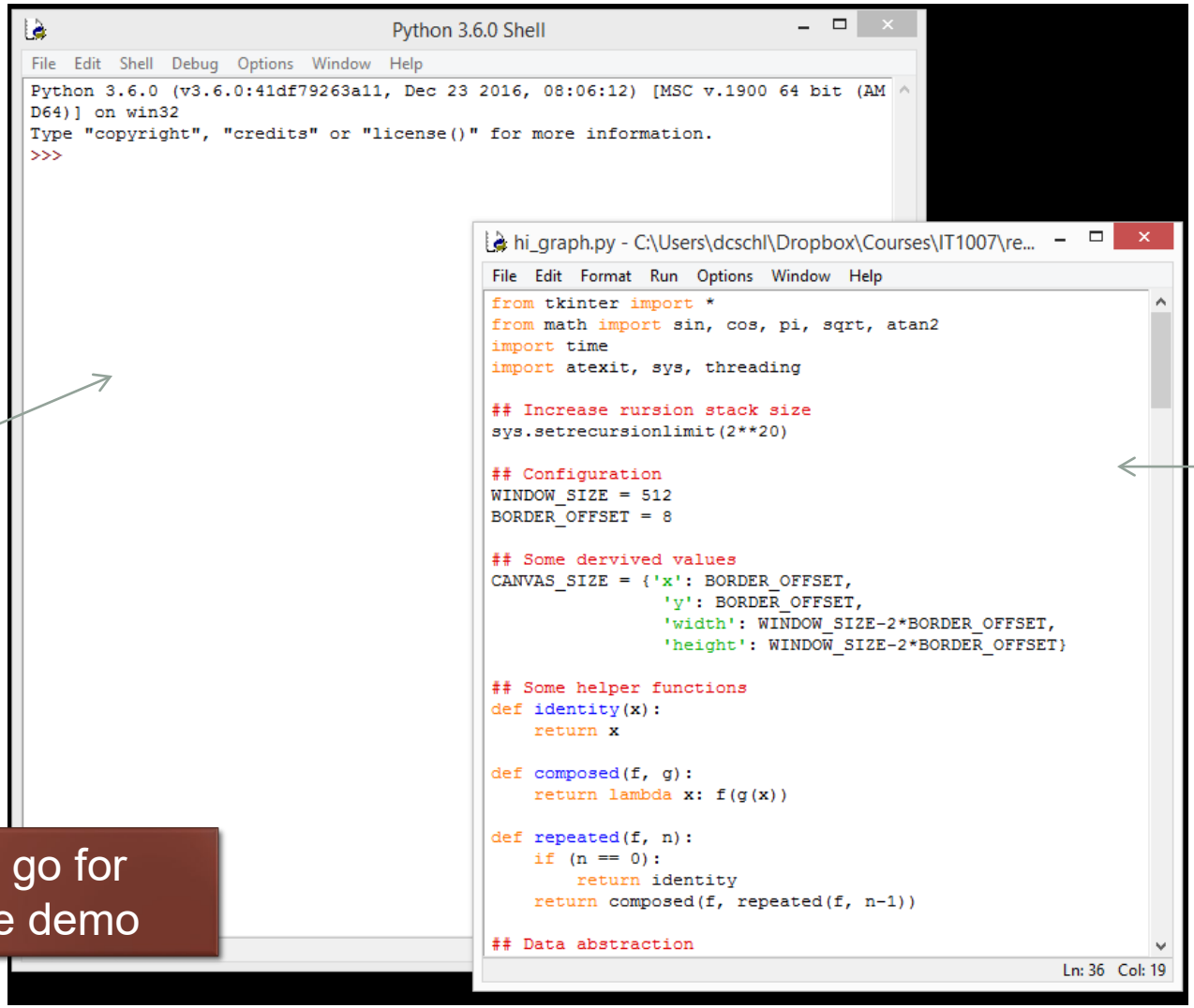
Automatic Vs. Manual Transmission:
Which is the best choice for you?

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



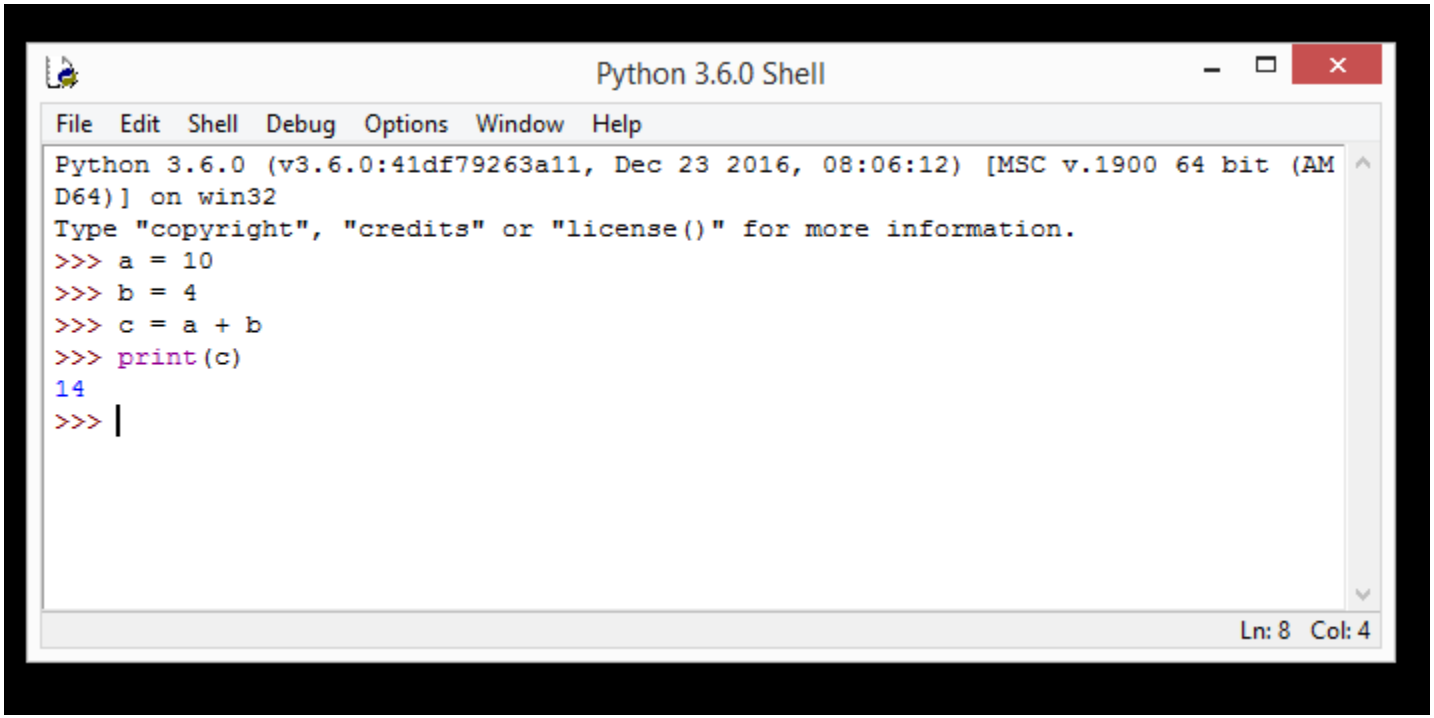
Console
- Input
- output

Editor and Your program

Let's go for some demo

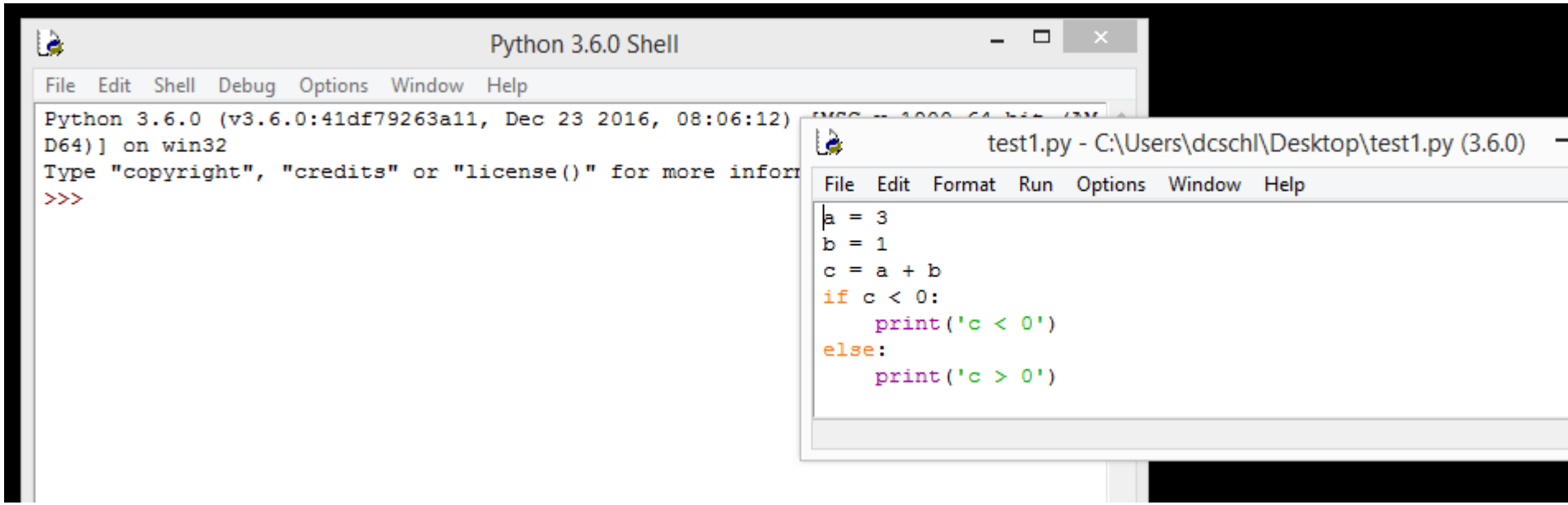
You can

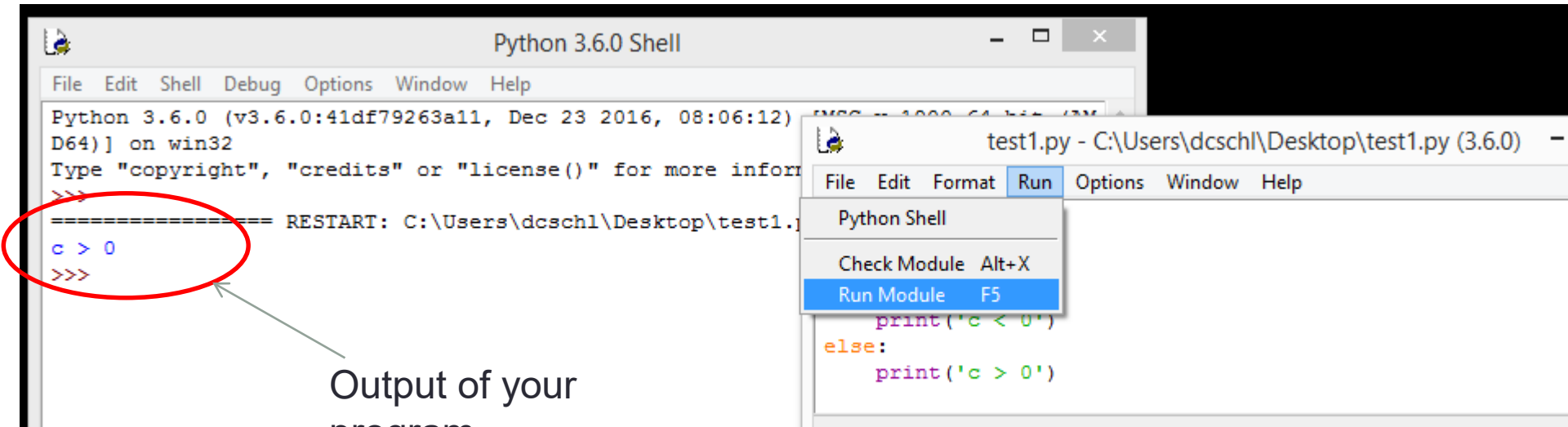
- Directly type into the console



- In which, we seldom do this

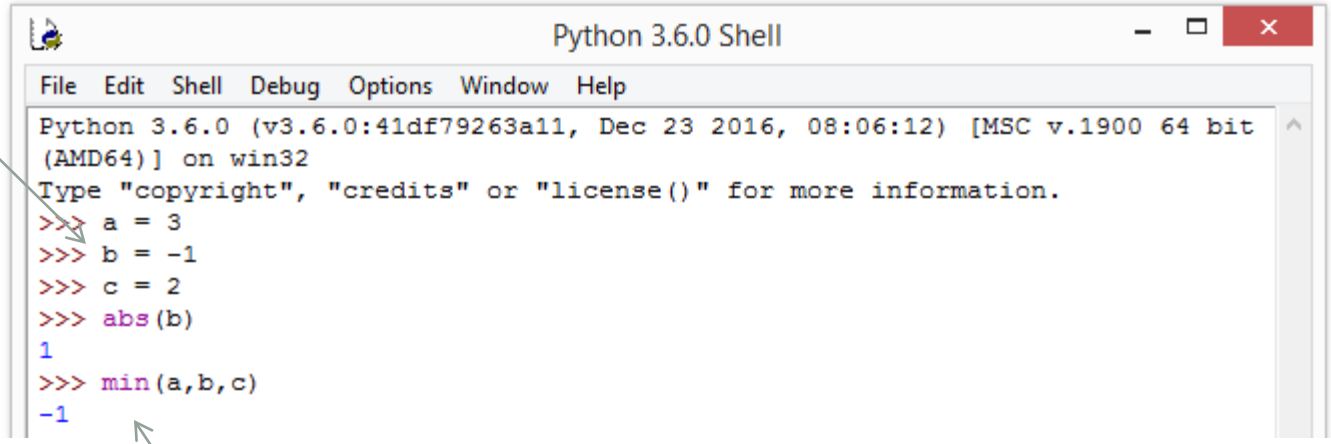
Or Run a file


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)
D64)] on win32
Type "copyright", "credits" or "license()" for more infor
>>>
test1.py - C:\Users\dcschl\Desktop\test1.py (3.6.0)
File Edit Format Run Options Window Help
a = 3
b = 1
c = a + b
if c < 0:
print('c < 0')
else:
print('c > 0')


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)
D64)] on win32
Type "copyright", "credits" or "license()" for more infor
>>>
========== RESTART: C:\Users\dcschl\Desktop\test1.
c > 0
>>>
test1.py - C:\Users\dcschl\Desktop\test1.py (3.6.0)
File Edit Format Run Options Window Help
Python Shell
Check Module Alt+X
Run Module F5
print('c < 0')
else:
print('c > 0')
Output of your program

Functions and Variables

`a, b, c` are variables



`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

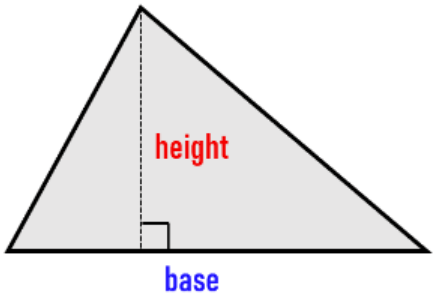
Calculating the Area of a Triangle

- b and h are variables!
- E.g. we can code in Python:

demo01.py - G:\My Drive\Courses\IT5001\Bootcamp\demo01.py (3.10.5)

File Edit Format Run Options Window Help

```
h = 10
b = 12
print(h*b/2)
```



$$A = \frac{1}{2}bh$$

b = base

h = height

Variable Naming

- A variable can have a short name (like x and y) or a more descriptive name (age, carname, total_volume).
- Rules for Python variables:
  - A variable name must start with a *letter* or the *underscore* character
  - A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
  - A variable name cannot start with a number
  - Variable names are case-sensitive (age, Age and AGE are three different variables)
  - A variable name cannot be any of the Python keywords.

Python Keywords

‹ Previous Next ›

Python has a set of keywords that are reserved words that cannot be used as variable names, function names, or any other identifiers:

Keyword	Description
and	A logical operator
as	To create an alias
assert	For debugging
break	To break out of a loop
class	To define a class
continue	To continue to the next iteration of a loop
def	To define a function
del	To delete an object
elif	Used in conditional statements, same as else if
else	Used in conditional statements
except	Used with exceptions, what to do when an exception occurs
False	Boolean value, result of comparison operations

Which are valid variable names for Python?

```
myvar = 1
my_var = 1
_my_var = 1
myVar = 1
MYVAR = 1
```

```
myvar2 = 1
2myvar = 1
my-var = 1
my var = 1
this_is_my_super_long_var = 1
```

Multi Words Variable Names

- Variable names with more than one word can be difficult to read. There are several techniques you can use to make them more readable:
- Camel Case (C/C++):

```
myBankBalance
```

- Snake Case (Python):

```
my_bank_balance
```

- Using the “wrong” convention will not cause any errors

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’	str
	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

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

```
>>> abc = 18
>>> my_string = 'This is my string'
>>> x, y = 1, 2
```

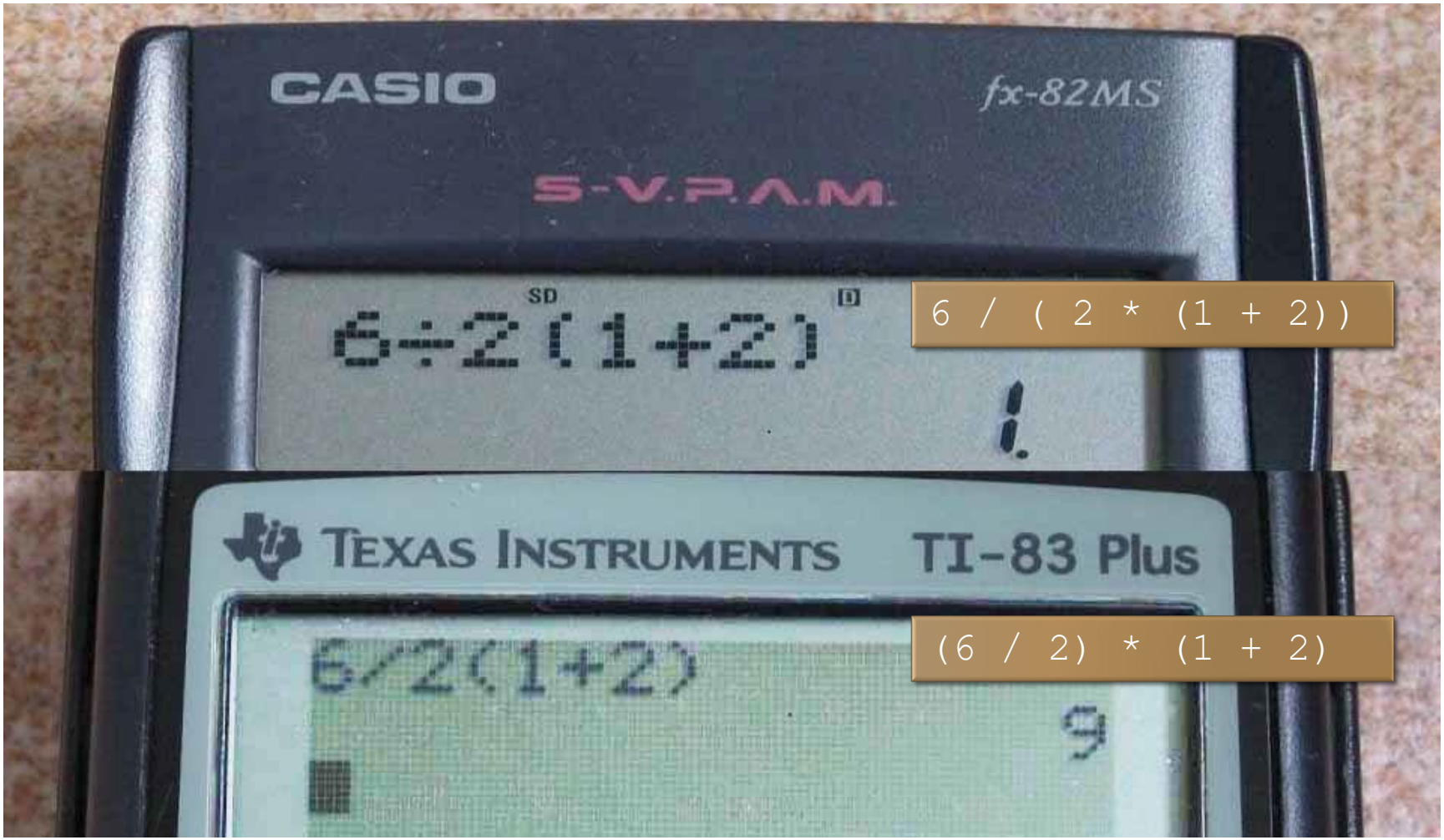
```
>>> a, b, c = 1, 2, 3
>>> a, b, c = c, b, a
>>> print(a, b, c)
???
```

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*3`
- `9`

Operator	Description
**	Exponentiation (raise to the power)
~ + -	Complement, unary plus and minus (method names for the last two are +@ and -@)
* / % //	Multiply, divide, modulo and floor division
+ -	Addition and subtraction
>> <<	Right and left bitwise shift
&	Bitwise 'AND'td>
^ |	Bitwise exclusive `OR' and regular `OR'
<= < > >=	Comparison operators
<> == !=	Equality operators
= %= /= //= -= += *= **=	Assignment operators
is is not	Identity operators
in not in	Membership operators
not or and	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’	str
	None

Boolean: Truth values

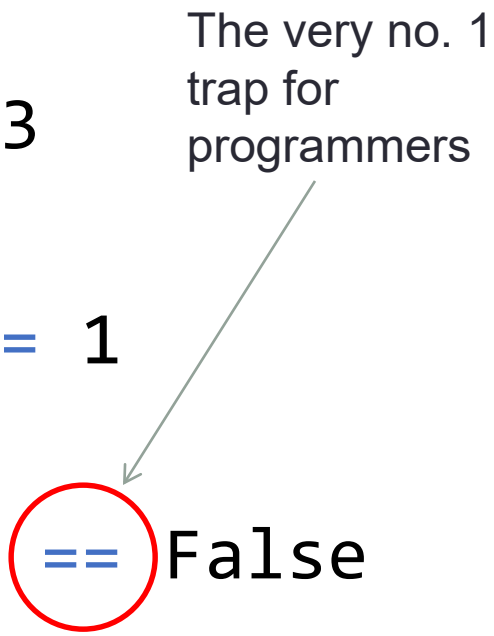
- Statements can be either True or False
- `2 > 1` is True
- `5 < 3` is False

Operators

- Comparison:

```
>>> 1 <= 10
True
>>> 5 > 15
False
>>> 5 <= 5
True
```

```
>>> 2 != 3
True
>>> '1' == 1
False
>>> False == False
True
>>> True != True
False
```



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'
>>> t = 'ck'
>>> s + t
'back'
>>> t = s + 'na' * 2
>>> t
'banana'
```

```
>>> 'z' in t
False
>>> 'bananb' > t
True
>>> 'banan' <= t
True
>>> '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 + w
>>> print(s)
‘bananabanana’
>>> s = w*3
>>> print(s)
‘bananabananabanana’
```

```
>>> s = (w+’ ‘)*2
>>> print(s)
‘banana banana ‘
```

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)

Non-inclusive

```
    s[start:stop:step]
```

Default
start = 0
stop = #letters
step = 1

```
>>> s = 'abcdef'
>>> s[0:2]
'ab'

>>> s[1:2]
'b'

>>> s[:2]
'ab'
```

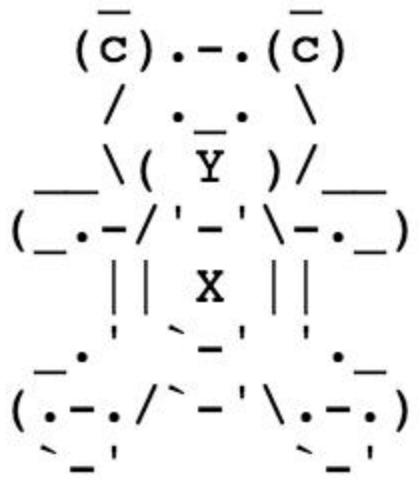
```
>>> s[1:5:3]
'be'

>>> s[::2]
'ace'

>>> s[::-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



FUNCTIONS

Abstraction

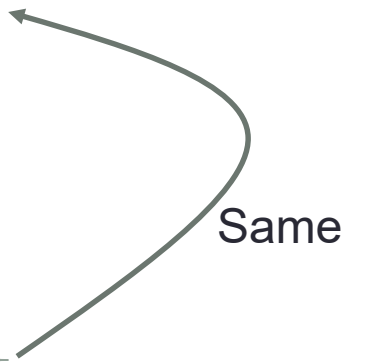
Back to this ASCII Art

```
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)
```

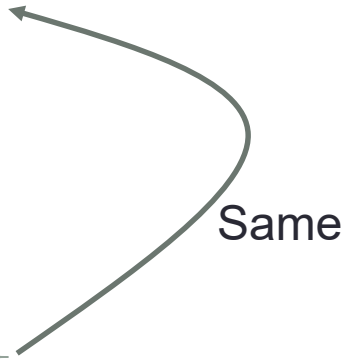



```
   *
   #
  ###
 #####
   #
  ###
 #####
   #
  ###
 #####
   #
  ###
 #####
   #
```

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)
```




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

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)
```

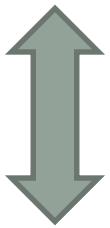
Give this part of the code a name:
E.g.
“drawTriangle”

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

Function Abstraction

```
c = "#"
s = " “

print(" ")
print("   *")

drawTriangle()


drawTriangle()


drawTriangle()


drawTriangle()

print(s*3 + c)
```

By “`drawTriangle()`” we mean:



```
print(s*3 + c)
print(s*2 + c * 3)
print(s + c * 5)
```

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

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

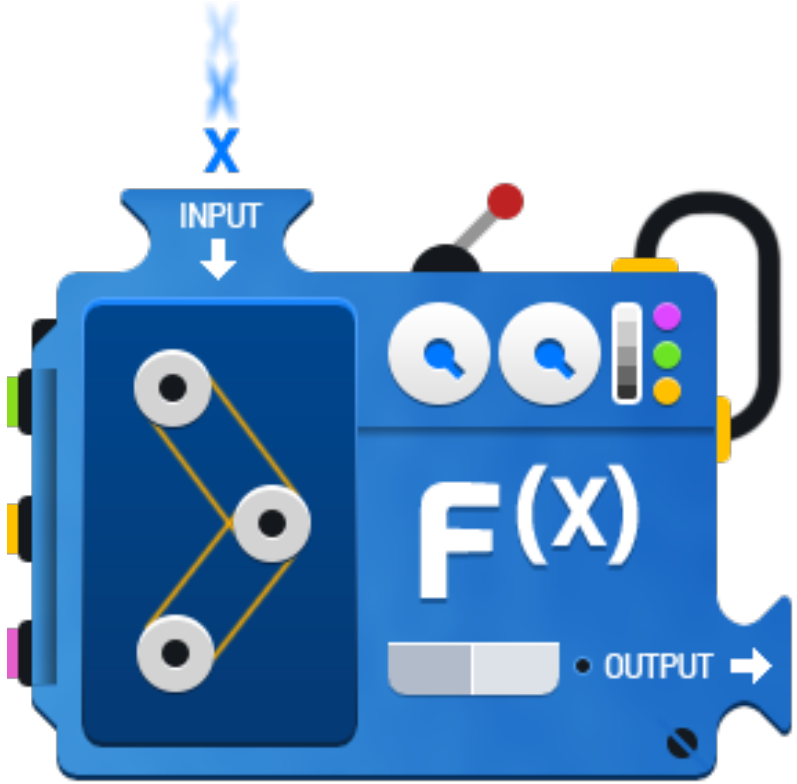
1. Measure the rice into a cup and level the top
2. Rinse the rice thoroughly in cold water
3. Pour the rice into a pan over a low heat
4. Add double the amount of water
5. Bring to a boil
6. Put a lid on and turn the heat down to as low as possible
7. Cook for 10 mins and do not take the lid off
8.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

- Abstraction:

https://www.bbcgoodfood.com/videos/techniques/how-cook-rice

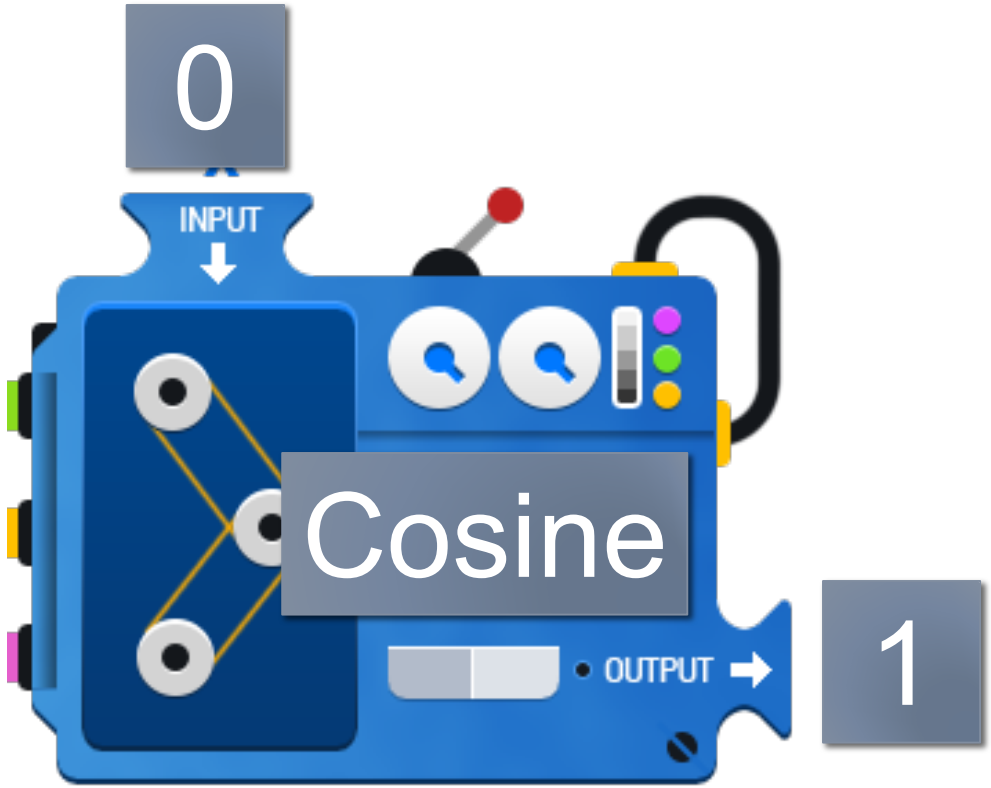
Functions

- 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

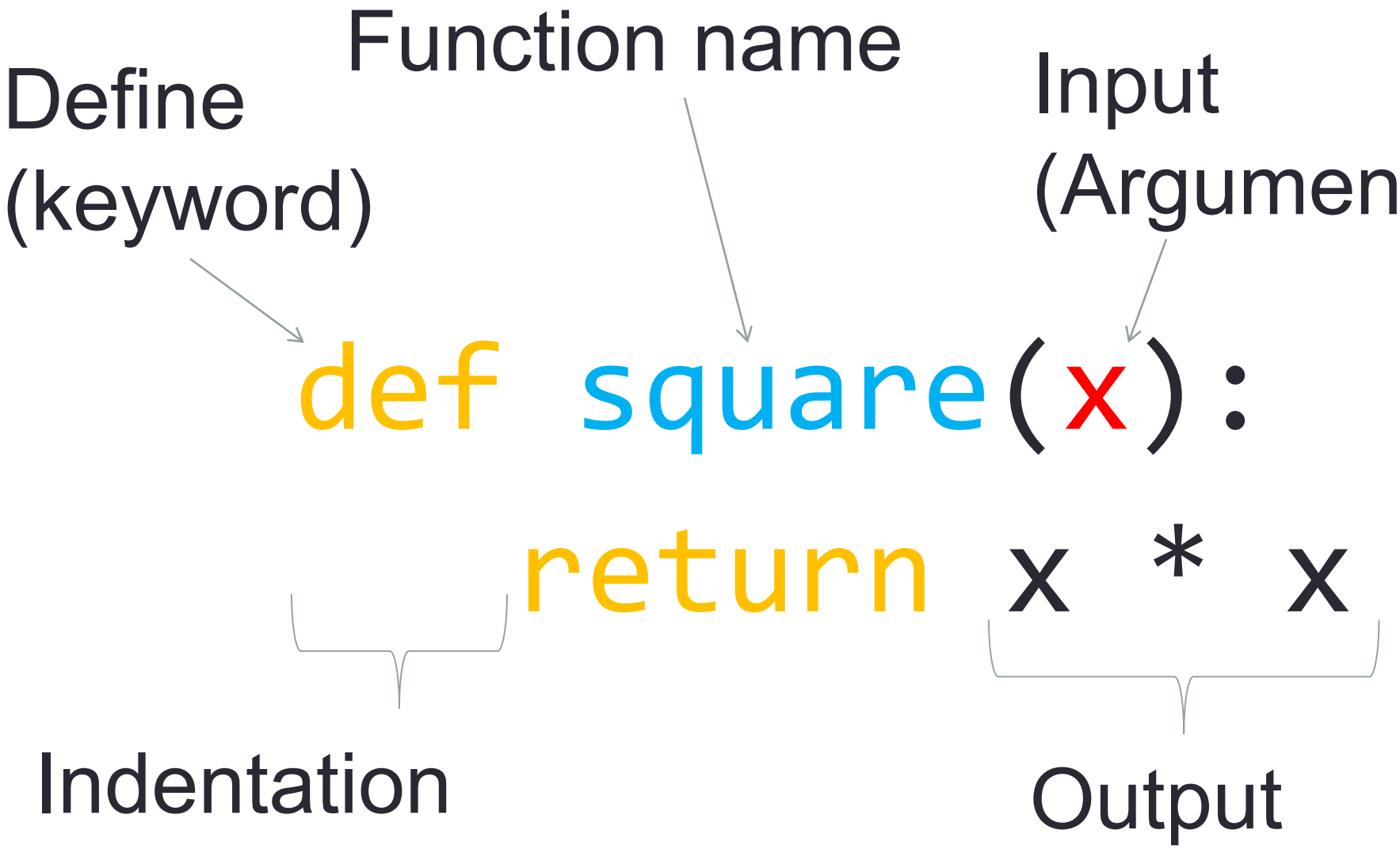


Let's Write Our Own Function!

```
def square(x):
    return x * x
```

```
>>> square(3)
9
>>> square(square(2))
16
>>> square(3,4)
???
```

Let's Write Our Own Function!



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!!!
```

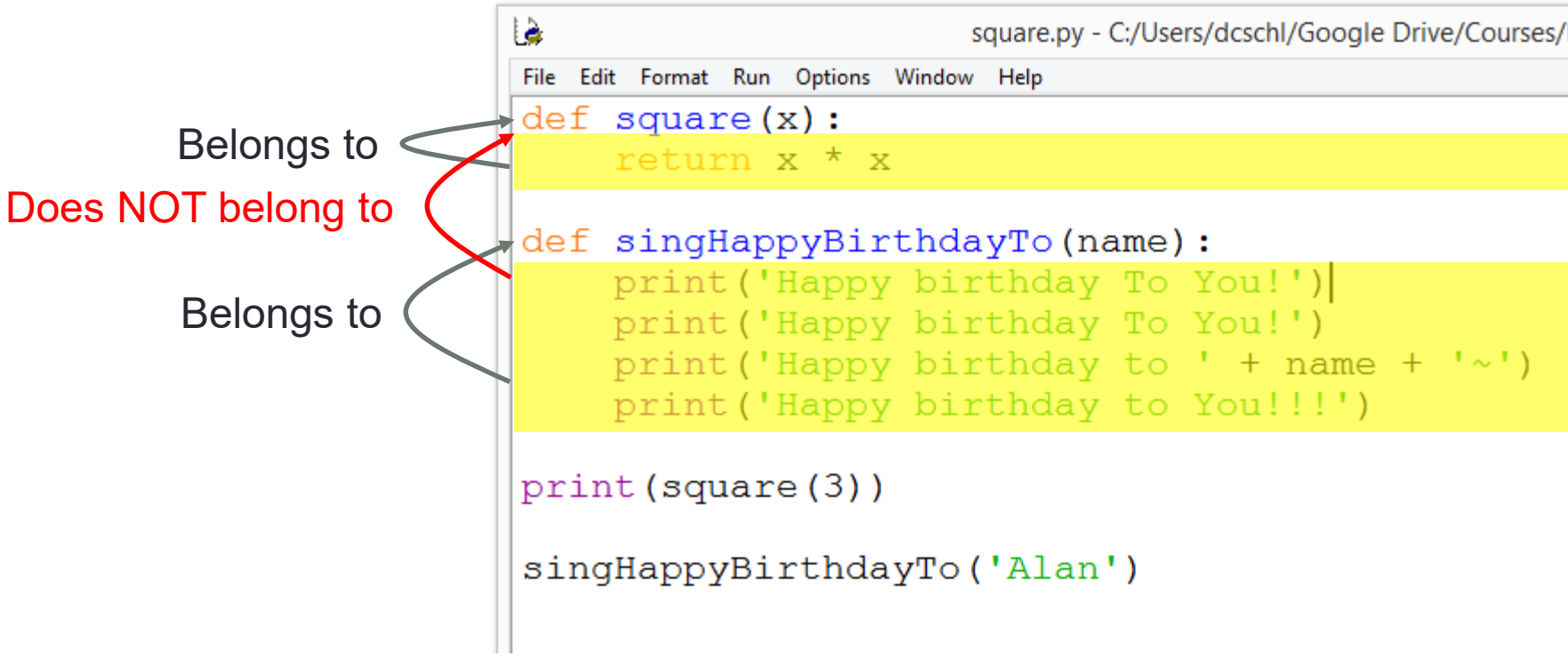
What if

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

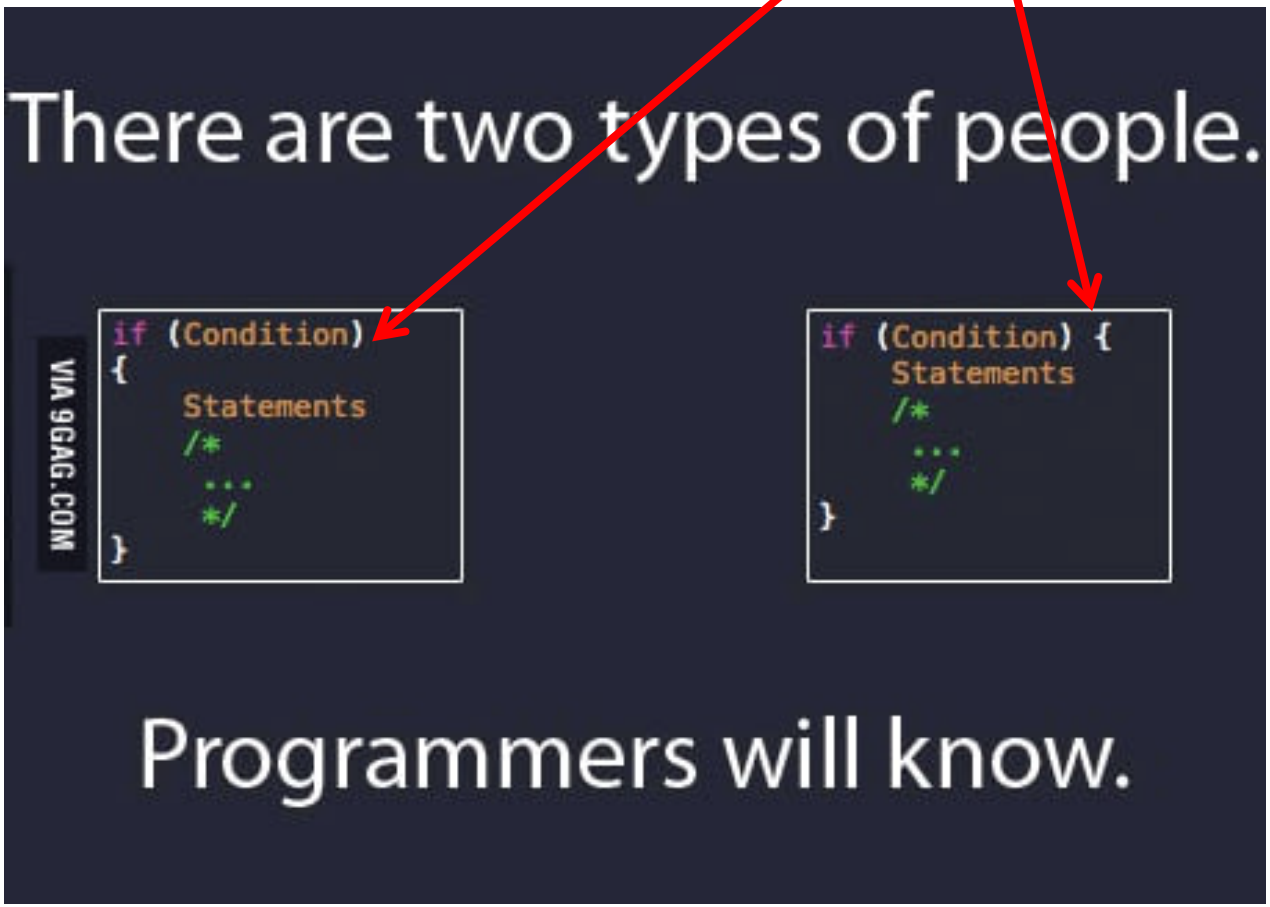
Indentation Matters



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

C

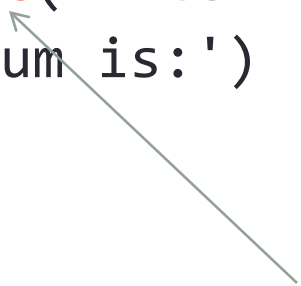
- For C language, it doesn't matter



```php
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-z\d]{2,64}$/i', $_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 add_two_numbers() :
    a = int(input('Enter an integer:'))
    b = int(input('Enter another integer:'))
    print('The sum is:')
    print(a + b)
```



Keyboard input
(Built-in function)

```
>>> add_two_numbers()
Enter an integer:2
Enter another integer:3
The sum is:
5
>>>
```

Python Packages

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

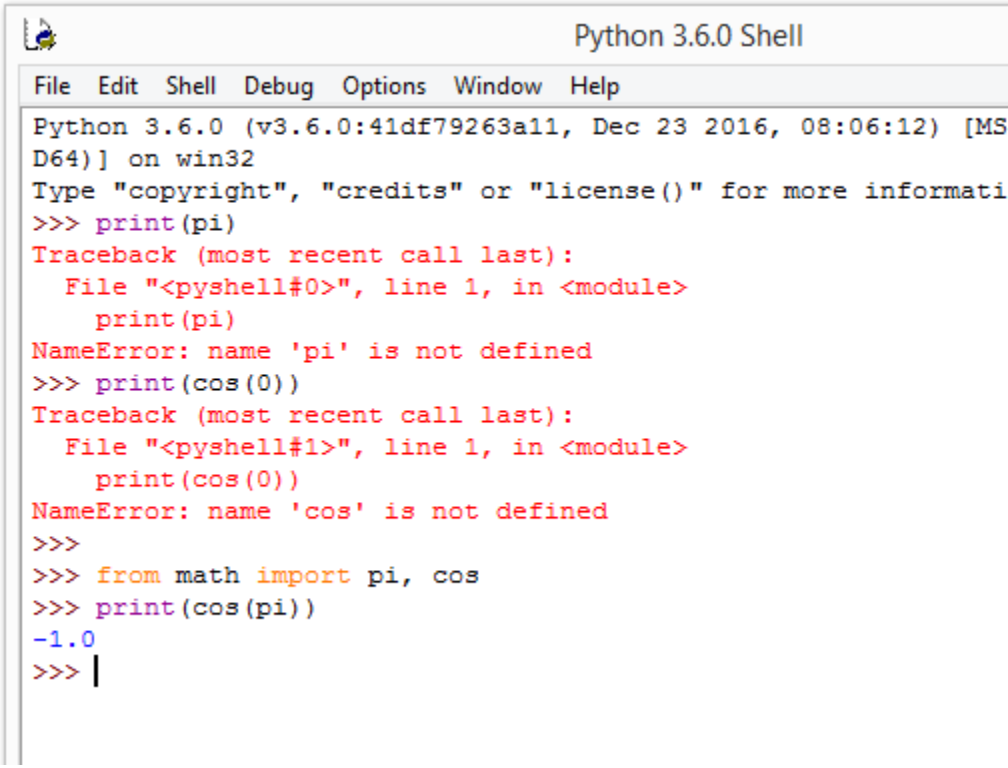
$$\cos(x) = \underbrace{\frac{1}{0!}}_{n=0} - \underbrace{\frac{x^2}{2!}}_{n=1} + \underbrace{\frac{x^4}{4!}}_{n=2} - \underbrace{\frac{x^6}{6!}}_{n=3} + \underbrace{\frac{x^8}{8!}}_{n=4} + \cdots$$

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

Different Scenario with Diff. Equipment

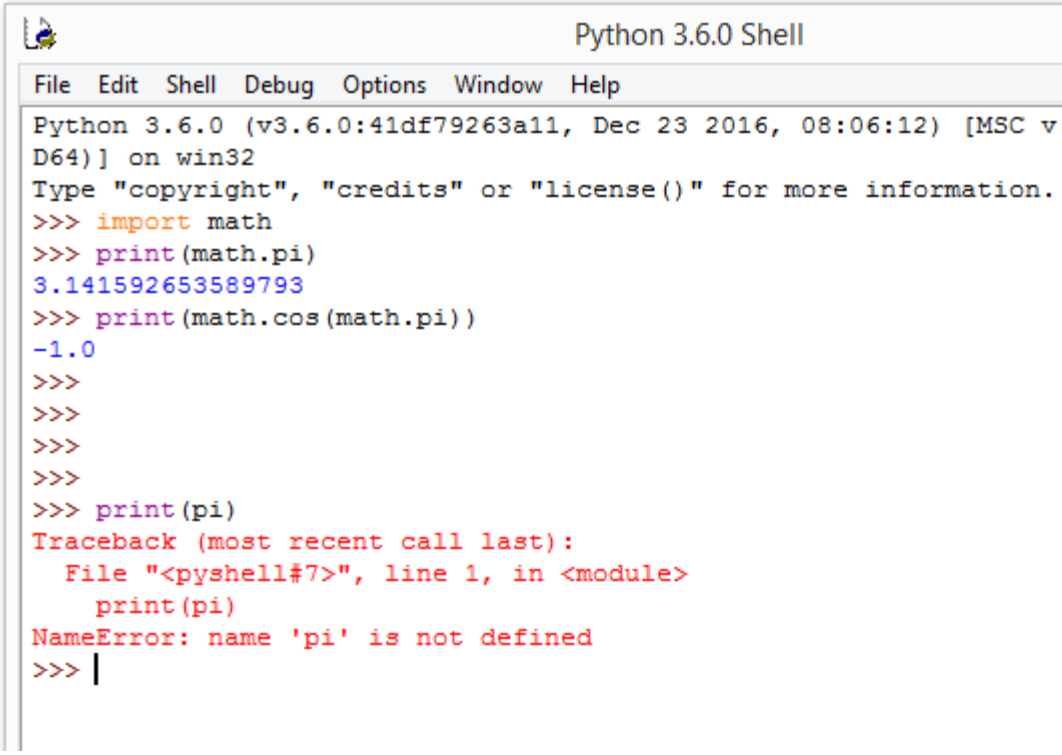
Python Packages (Ready Made Functions)

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



The Other way to Import cos

- You need at add the prefix "`math.`" before the function
- Otherwise..



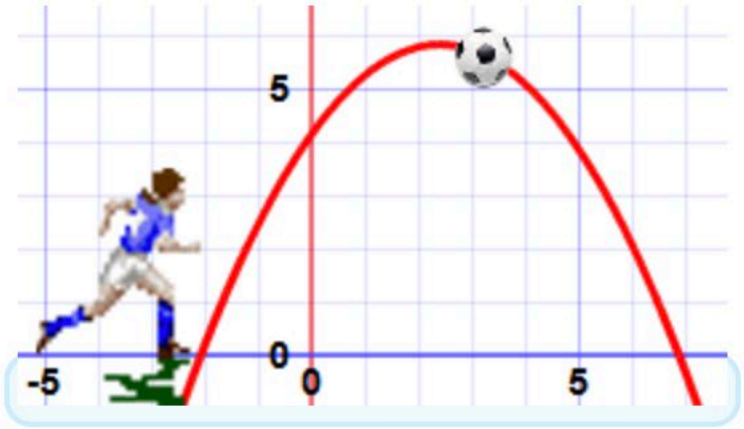
Example: Solving a Quadratic Eqt.

An example of a **Quadratic Equation**:

this makes it Quadratic

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

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

solve_qe1.py - /Volumes/GoogleDrive/My Drive/Courses/YSC2221/Lectur

```
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?

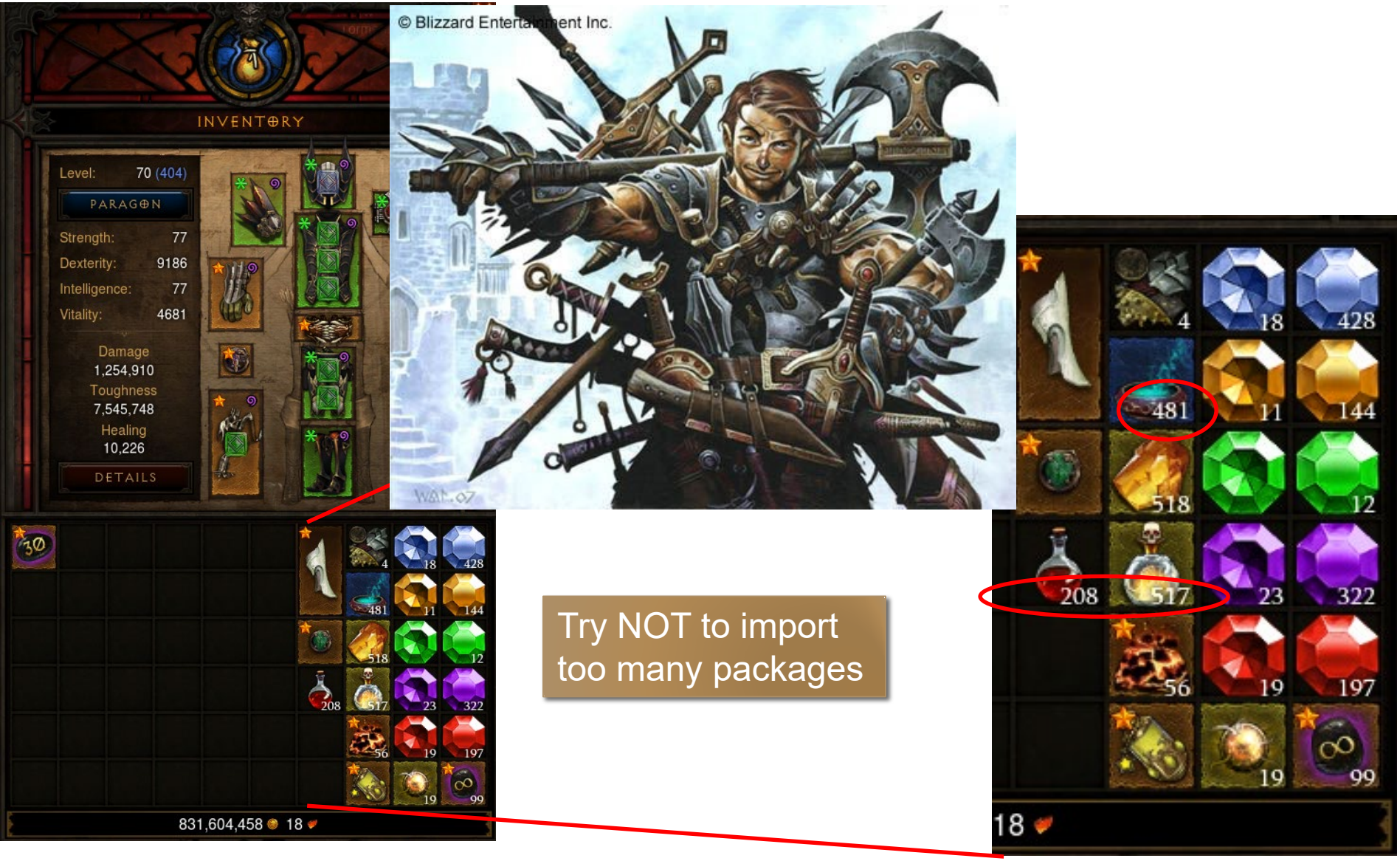
- First, “from math import cos”
  - Make your code shorter
  - Save memory
  - But you need to know exactly which function you have to import

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

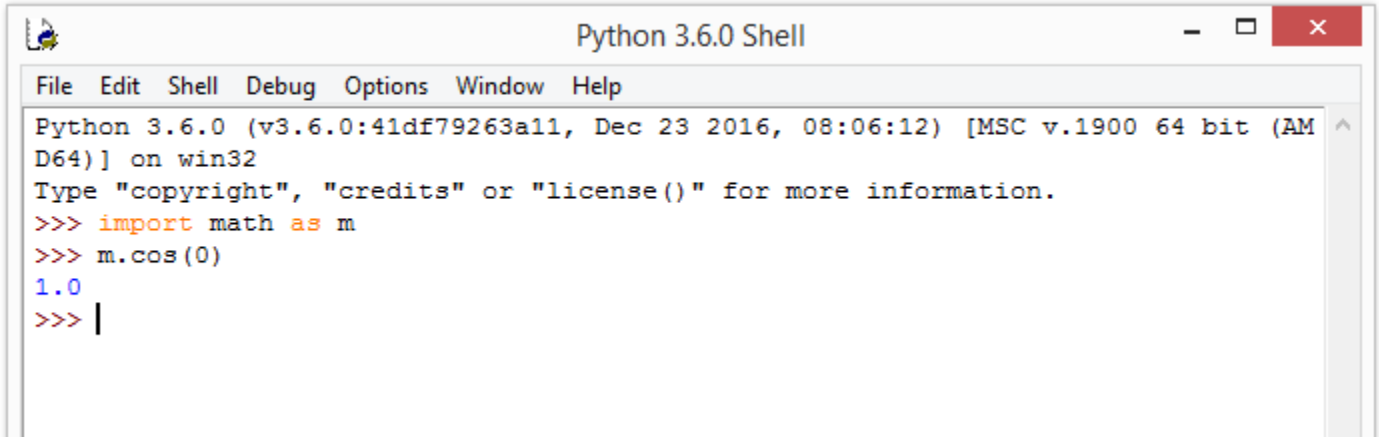
- Second, “import math”
  - You can be more relax to use functions in the whole library
  - But your code will be longer and require more memory

```
>>> 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

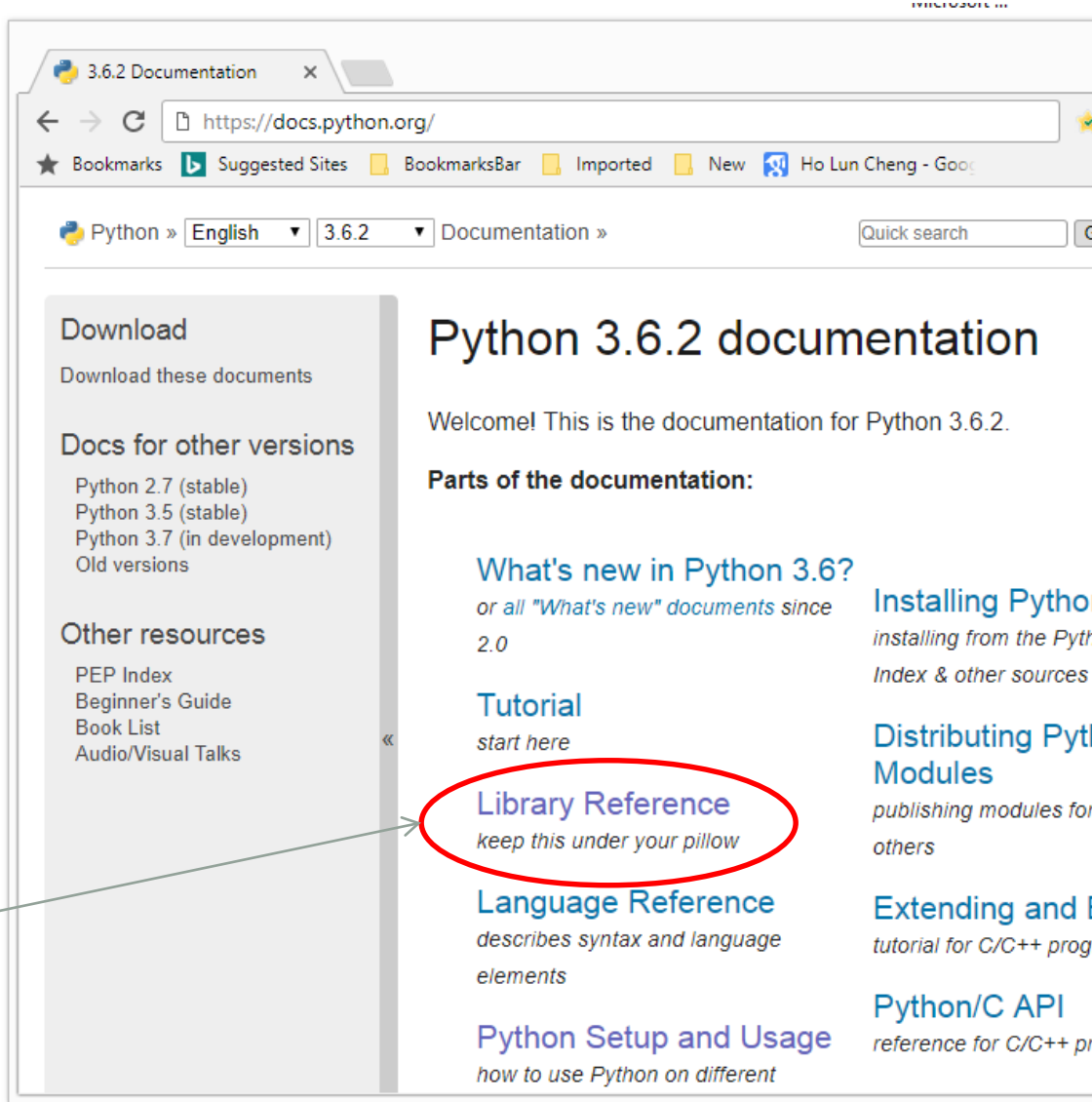


To Avoid Long Name



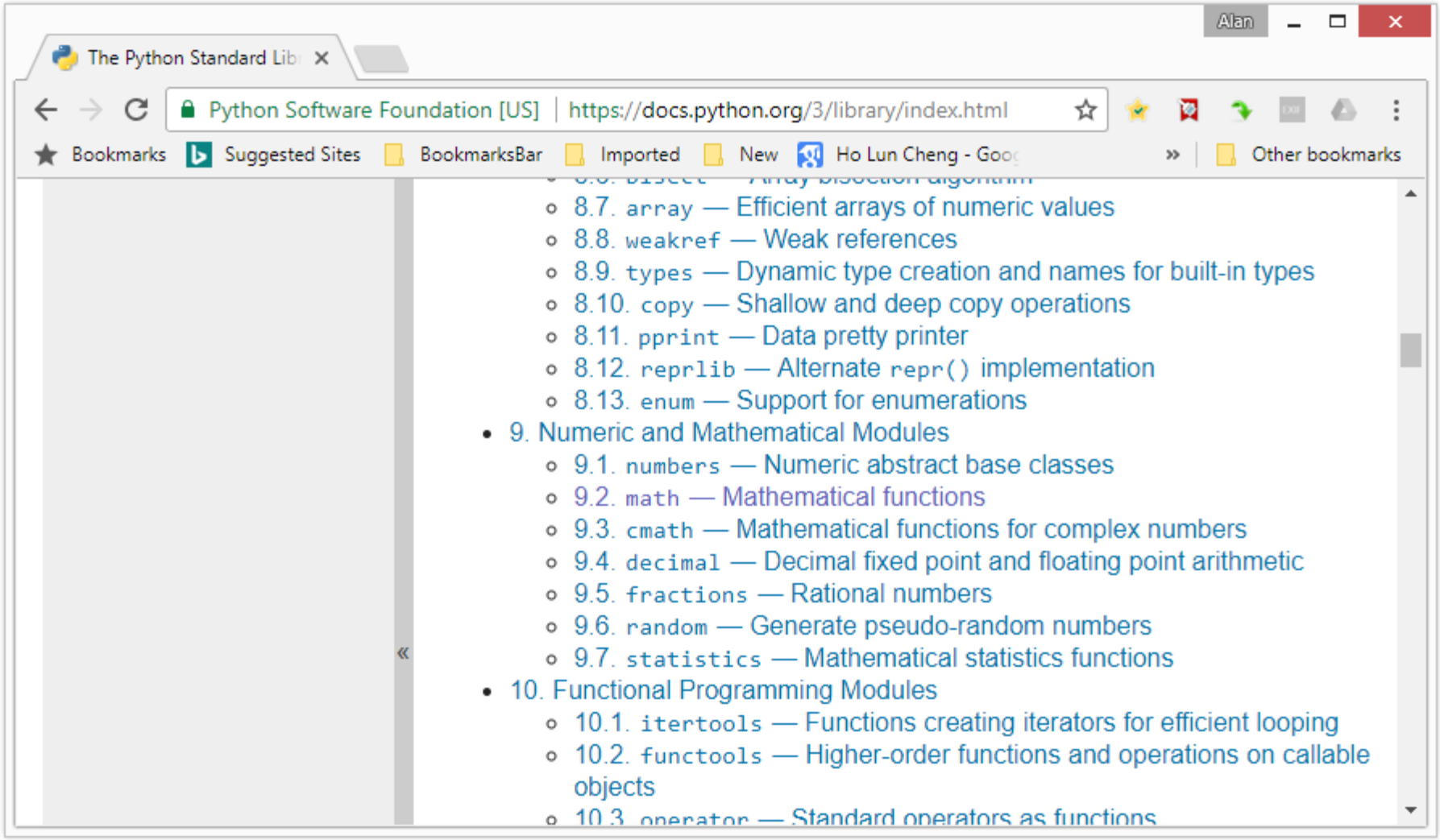
For More Packages

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



Keep this under your pillow

Python Packages



Built-in Functions

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

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