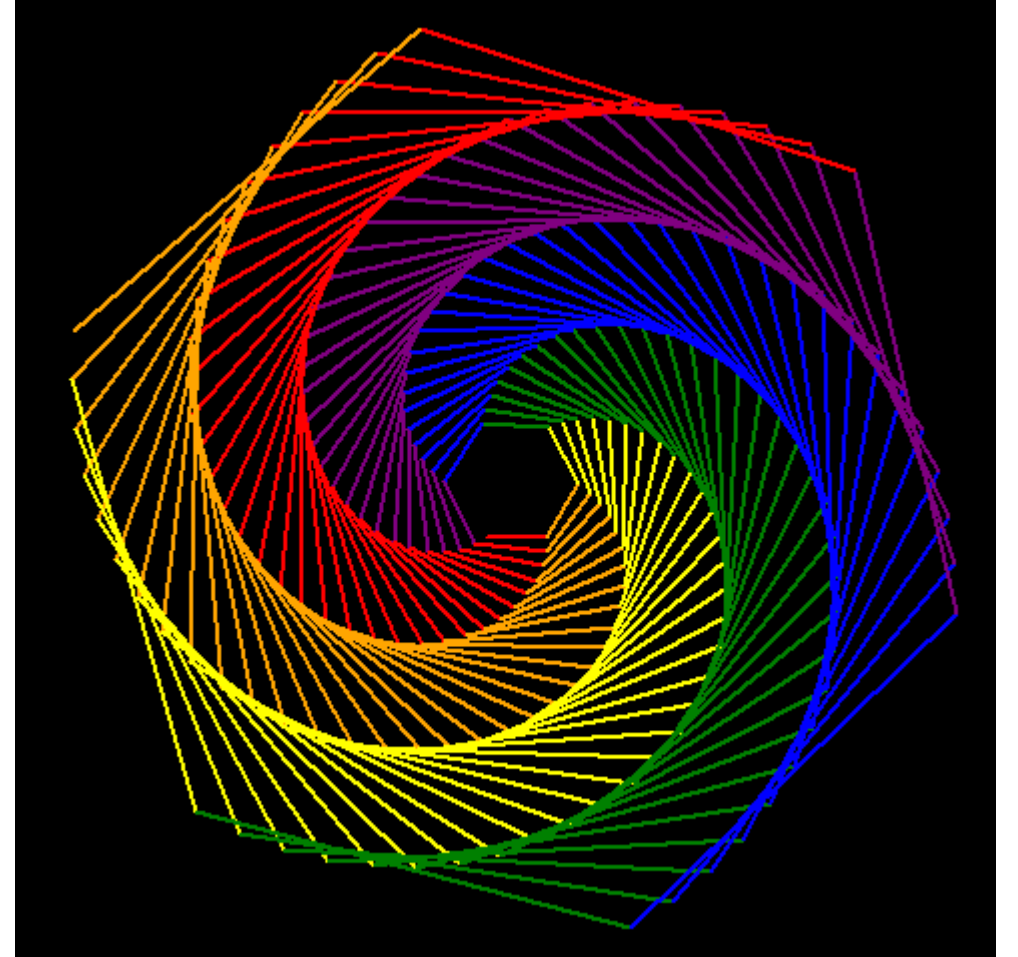


Turtle Graphics

Let's get familiar with Programming

Turtle Graphics

- “A picture is worth a thousand words”



Turtle Graphics

- “Turtle” is a Python feature like a drawing board, which lets us command a turtle to draw all over it!
- Imagine you have a little turtle on the screen and facing East
 - *picture*
- Then you command it to go forward 100 pixels
- Then turn left 90 degree and walk another 50 pixels
 - *picture*

Drawing a Square

- forward(100)
- right(90)
- forward(100)
- right(90)
- forward(100)
- right(90)
- forward(100)
- right(90)

Assignment 1

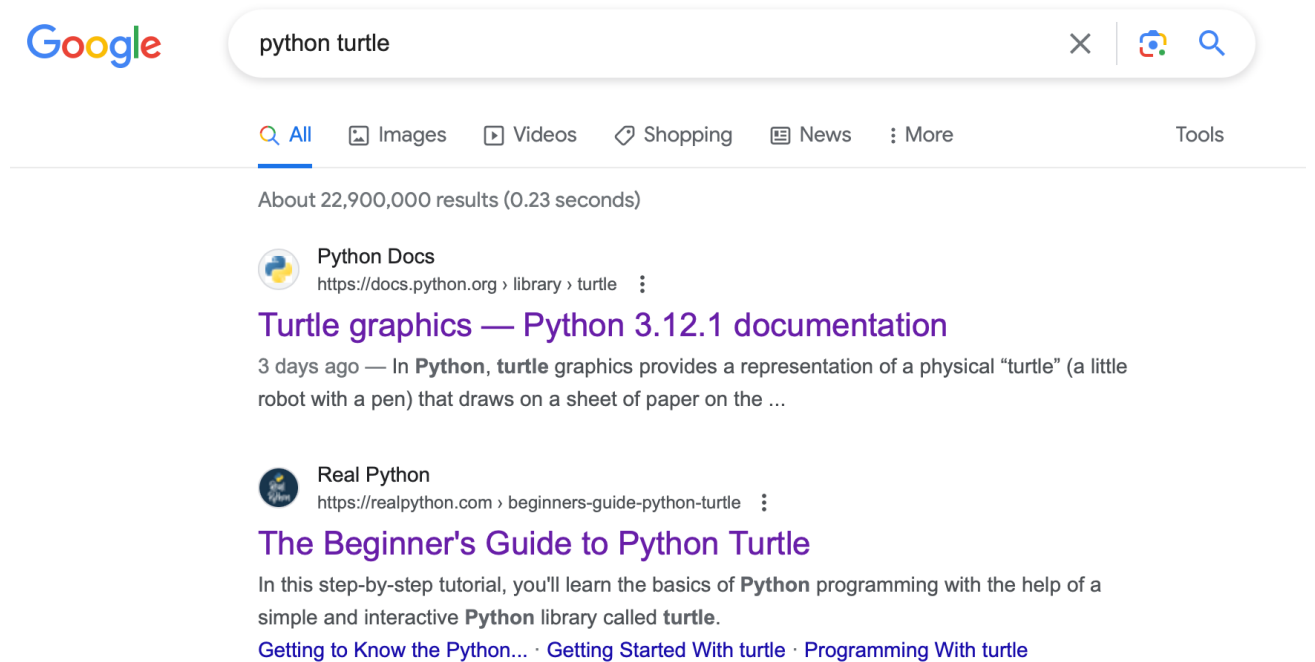
- Draw a triangle
- And share how you draw it
 - (You may be tempted to use for-loop now, but let's try without it now)

Assignment 2

- Draw a letter A
- *pictures of two A's)
- Share how you draw it
- Or did you draw other things?

How to learn programming in general?

- Google is your good friend
- E.g. how to get more commands in Turtle?
 - Google "turtle Python"

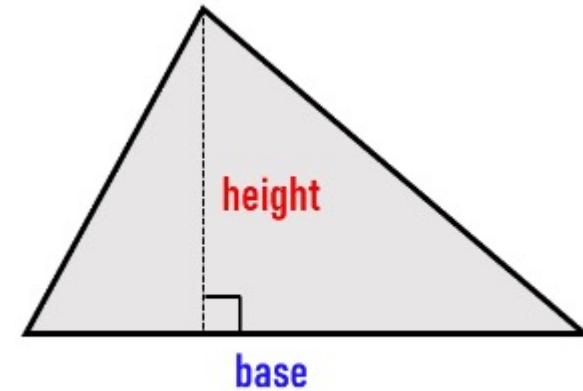


Variables

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 cannot start with a number
 - A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
 - 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
<u>and</u>	A logical operator
<u>as</u>	To create an alias
<u>assert</u>	For debugging
<u>break</u>	To break out of a loop
<u>class</u>	To define a class
<u>continue</u>	To continue to the next iteration of a loop
<u>def</u>	To define a function
<u>del</u>	To delete an object
<u>elif</u>	Used in conditional statements, same as else if
<u>else</u>	Used in conditional statements
<u>except</u>	Used with exceptions, what to do when an exception occurs
<u>False</u>	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`

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

Problem

- You put \$10000 into your fixed deposit bank account and the annual interest is 4%
 - How much do you have after one year
 - How much do you have after two years
 - How much do you have after three years?
- We need to use the balance result AFTER one year to calculate the next year

```
demo01.py - G:\My Drive\Courses\IT5001\Bootcamp\demo01.py (3.10.5)
File Edit Format Run Options Window Help

balance = 10000
rate = 0.04
balance_1y = balance * (1+rate)
print("After one year, the balance")
print(balance_1y)
balance_2y = balance_1y * (1+rate)
print("After two years, the balance")
print(balance_2y)
```

Calculating Balance

- In computing, a variable is a piece of memory that stores a value that can be *changed*.

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

File Edit Format Run Options Window Help

```
balance = 10000
rate = 0.04
balance_1y = balance * (1+rate)
print("After one year, the balance is:")
print(balance_1y)
balance_2y = balance_1y * (1+rate)
print("After two years, the balance is:")
print(balance_2y)
```

Variables Replaceability and Type

Variable Replaceability

- Normally what we learned in math in our secondary/primary schools
 - if $x = 10$, x is equal to 10 for the whole question/problem
- However it is not true in most programming languages
- What will the code in the right print?
 - The value of x is changed many times

demo.py - C:/Users/dcschl/Downloads/demo.py (3.10.5)

File Edit Format Run Options Window Help

```
x = 1
print(x)
x = 2
print(x)
x = 3
x = 4
print(x)
```

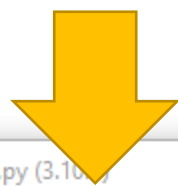
Problem

- You put \$10000 into your fixed deposit bank account and the annual interest is 4%
 - How much do you have after one year?
 - How much do you have after two years?
 - How much do you have after three years?
- After computing the balance after one year, the original “balance” is no longer needed
 - I could just “replace” the balance instead of creating a new `balance_1y`

```
demo01.py - G:\My Drive\Courses\IT5001\Bootcamp\demo01.py (3.10.5)
File Edit Format Run Options Window Help

balance = 10000
rate = 0.04
balance_1y = balance * (1+rate)
print("After one year, the balance")
print(balance_1y)
balance_2y = balance_1y * (1+rate)
print("After two years, the balance")
print(balance_2y)
```

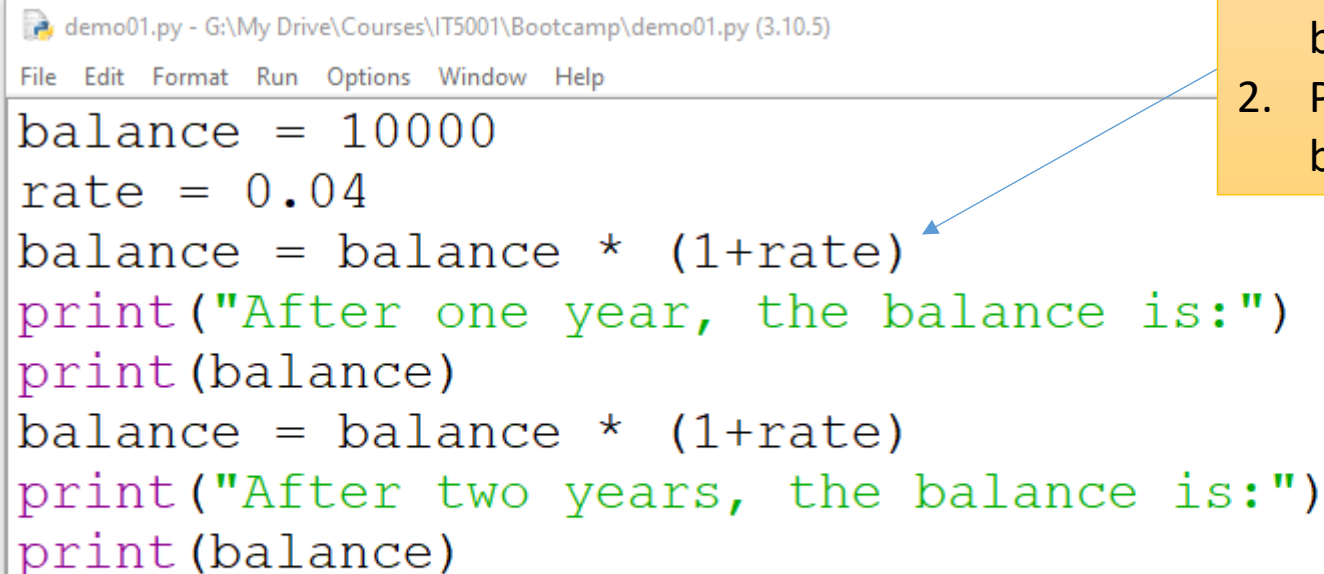
```
demo01.py - G:\My Drive\Courses\IT5001\Bootcamp\demo01.py (3.10.5)
File Edit Format Run Options Window Help
balance = 10000
rate = 0.04
balance_1y = balance * (1+rate)
print("After one year, the balance is:")
print(balance_1y)
balance_2y = balance_1y * (1+rate)
print("After two years, the balance is:")
print(balance_2y)
```



```
demo01.py - G:\My Drive\Courses\IT5001\Bootcamp\demo01.py (3.10.5)
File Edit Format Run Options Window Help
balance = 10000
rate = 0.04
balance = balance * (1+rate)
print("After one year, the balance is:")
print(balance)
balance = balance * (1+rate)
print("After two years, the balance is:")
print(balance)
```

Assignment Operator “=”

- The “=” *operator* in Python is different from math
 - In math, the LHS and RHS of an equal side are equal
- But in Python “a = b” meaning
 - replace the value of a by b



```
demo01.py - G:\My Drive\Courses\IT5001\Bootcamp\demo01.py (3.10.5)
File Edit Format Run Options Window Help
balance = 10000
rate = 0.04
balance = balance * (1+rate)
print("After one year, the balance is:")
print(balance)
balance = balance * (1+rate)
print("After two years, the balance is:")
print(balance)
```

1. compute the value of $\text{balance} * (1 + \text{rate})$
2. Put that value into balance

Turtle Graphics: Drawing a Square

```
forward(100)
right(90)
forward(100)
right(90)
forward(100)
right(90)
forward(100)
right(90)
```

**What if you want
to draw a square
with side 200?
300? 400?**

Turtle Graphics: Drawing a Square

```
forward(200)
right(90)
forward(200)
right(90)
forward(200)
right(90)
forward(200)
right(90)
```

```
forward(300)
right(90)
forward(300)
right(90)
forward(300)
right(90)
forward(100)
right(90)
```

**What if you want
to draw a square
with side 200?
300? 400?**

Turtle Graphics: Drawing a Square

```
side_length = 200  
forward(side_length)  
right(90)  
forward(side_length)  
right(90)  
forward(side_length)  
right(90)  
forward(side_length)  
right(90)
```

What if you want
to draw a square
with side 200?
300? 400?


Variable Type

Types of People

Character introduction


A SIMPLE THINKING ABOUT BLOODTYPE

Bloodtype A Kind, Delicate, Warm




A tries to live in certain life pattern that A exclusively made. A cares about people around. A never gets accustomed to rapid changes or interpersonal conflicts. When starting a new thing, A becomes a perfectionist. A is very good at controlling its emotion and has strong endurance.

Bloodtype B Frank, Creative, Free




B is free enough to express its opinion out of the convention and also creative with full of ideas, so B is always pursuing something intriguing. B is open, capricious, indifferent and optimistic about future. 'Romantic' is not the matching word with B but warmhearted.

Bloodtype O Active, Curious, Friendly



O is a realist and an idealist at the same time. O is goal-oriented, therefore O gets disoriented easily without a vivid goal. O has strong concentration power and wants to be in a group. O is straightforward and self-assertive with its own principal. O is emotional and competitive, but doesn't stay anger long.

Bloodtype AB Talented, Mysterious, Wise



AB is coexisting with a calm A character and a capricious B character. AB is a talented straightforward critic and negotiator with discreet and realistic prudence. Sometimes AB is idealistic and emotional. AB wants to keep its own place and hates to be intruded.

LEEON SMART

The Best Book Genres,
According to Your Zodiac Sign

THE READER'S DIGEST VERSION


Aries
Action adventures,
page-turning thrillers


Taurus
Cozy romances,
feel-good beach reads


Gemini
LGBTQ novels,
slice-of-life stories


Cancer
Women's fiction, family
dramas, tearjerkers


Leo
Young adult fiction,
mythic adventures


Virgo
Historical fiction,
feminist books



Libra
Biographies,
legal dramas


Scorpio
Mysteries, Gothic
horror, true crime


Sagittarius
Graphic novels,
book series


Capricorn
Classic novels,
foundational literature


Aquarius
Science fiction,
dystopian stories


Pisces
Fantasy adventures,
mythological retellings

Variable Types

- Five primitive data types in Python:
- The five primitive data types in Python
 - Integers
 - Float
 - Strings
 - Boolean
 - None
- You can check the type of a variable by type()

```
a = 1
b = 0.1
c = "hello"
d = True
e = None
```

```
>>> b = 0.1
>>> type(b)
<class 'float'>
```

Difference between Integers and Float

Integers

- Whole numbers
- Exact representation
- Almost have no upper limit in Python*

Float

- Floats are number with decimal points
- Approximation (IEEE 754-1985)
 - Try computing $1 - 0.9$?
- Have upper and lower limits
 - E.g. max = 1.7976931348623157e+308
 - ($\sim 2^{1024}$)
 - May varies

Arithmetic: + - * / ** // %

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

```
>>> a
```

```
6
```

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

```
8
```

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

```
3.6666666666666665
```

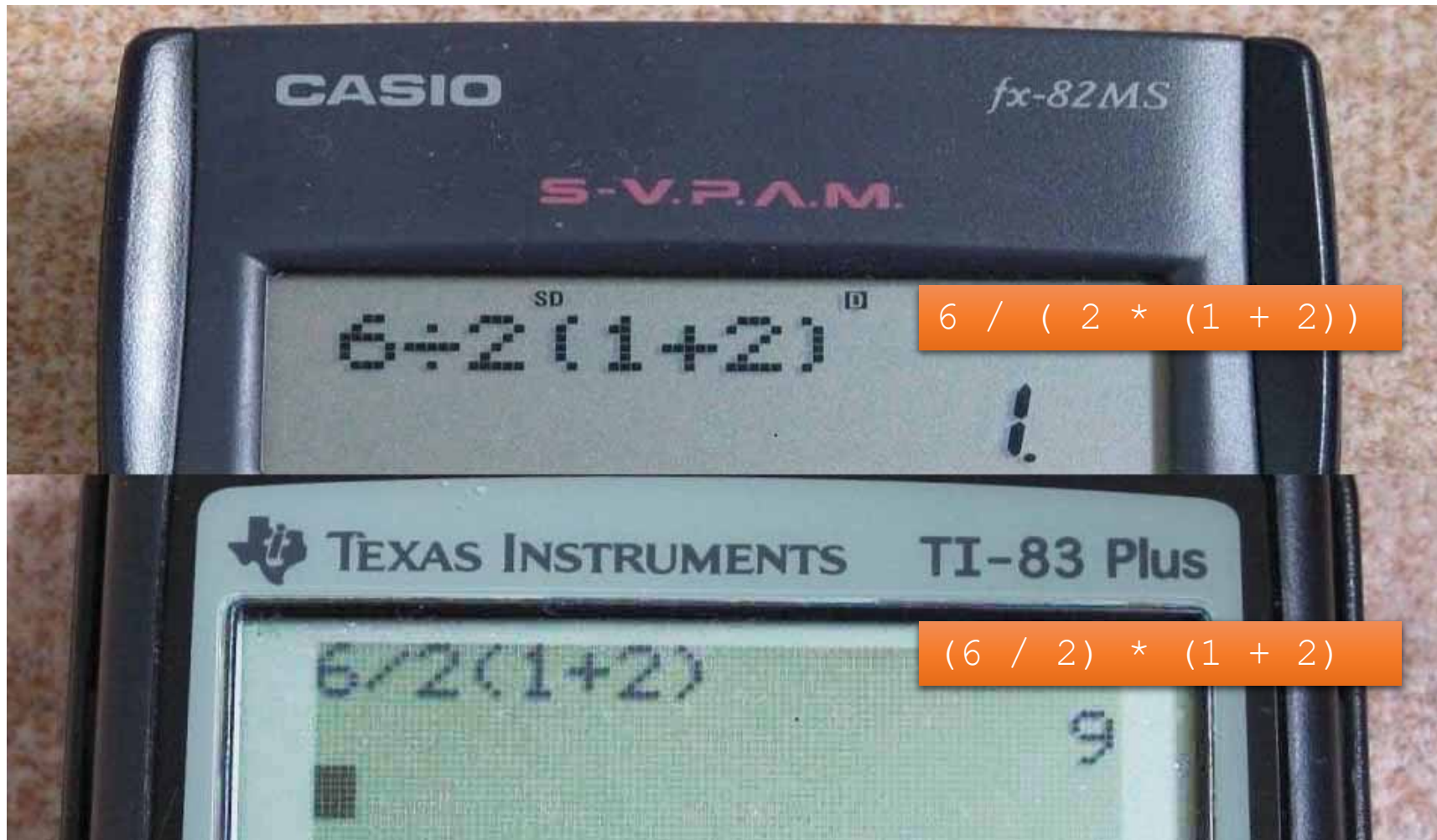
```
>>> 11 // 3
```

```
3
```

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

```
2
```

Operator Precedence



Python Operator Precedence

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

Operator	Description
<code>(expressions...),</code> <code>[expressions...], {key: value...}, {expressions...}</code>	Binding or parenthesized expression, list display, dictionary display, set display
<code>x[index], x[index:index], x(arguments...),</code> <code>x.attribute</code>	Subscription, slicing, call, attribute reference
<code>await x</code>	Await expression
<code>**</code>	Exponentiation [5]
<code>+x, -x, ~x</code>	Positive, negative, bitwise NOT
<code>*, @, /, //, %</code>	Multiplication, matrix multiplication, division, floor division, remainder [6]
<code>+, -</code>	Addition and subtraction
<code><<, >></code>	Shifts
<code>&</code>	Bitwise AND
<code>^</code>	Bitwise XOR
<code> </code>	Bitwise OR
<code>in, not in, is, is not, <, <=, >, >=, !=, ==</code>	Comparisons, including membership tests and identity tests
<code>not x</code>	Boolean NOT
<code>and</code>	Boolean AND
<code>or</code>	Boolean OR
<code>if - else</code>	Conditional expression
<code>lambda</code>	Lambda expression
<code>:=</code>	Assignment expression

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

- 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



Default
start = 0
stop = #letters
step = 1

s[start:stop:step]

```
>>> 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

$$\begin{array}{c}
 (\bar{c}) \cdot - \cdot (\bar{c}) \\
 / \quad \cdot \quad \backslash \\
 \backslash (\bar{Y}) / \\
 (\cdot - / ' - ' \backslash - \cdot) \\
 | \quad | \quad X \quad | \quad | \\
 ' \quad \backslash \quad - \quad / \quad ' \\
 (\cdot - \cdot / \backslash - ' \backslash \cdot - \cdot) \\
 \backslash \quad ' \quad \quad \quad \backslash \quad '
 \end{array}$$

```

@~t@
@~t@  @@@
@~(t@ %^^^@
@((tt@s^^^@
@((tt@s///@
@ttCs^^^@
@CC@tts^s@ @@@G////@
@O~CC@%ttst@@ /((~//@
@O  ///(((^ ~~~~~//@
@O  //(@@@((^ ~//(((/@
@O^ /( @@@((^ ~~~~//@
@^^^/( @@@((^ ~((~/t(/@
@^^  (((((/^ (~/t/((@@
@  ^  (((((/^ ((~/t/((@@
@ s  ^^^^ (t~%ttt(C@@
      @%%  (((((%%@
      @ %s  (((((%%@
      @ t  @ @ OO%%%%%%%%@
      @@@@@ @@@@@@@@@@@@@@

```

Hard Code

- The amount of balance is “hard coding”
- Meaning, you cannot change the value without changing the code
- More appropriate way is to ask the user each time

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

File Edit Format Run Options Window Help

```
balance = 10000
rate = 0.04
balance_1y = balance * (1+rate)
print("After one year, the balance is:")
print(balance_1y)
balance_2y = balance_1y * (1+rate)
print("After two years, the balance is:")
print(balance_2y)
```

input()

- You can use the input function to get an input from the user
- e.g.

```
balance = input()
```

- However, my code won't work like this, why?

```
print("What is your balance?")  
balance = input()  
rate = 0.04  
print("Your balance after one year is:")  
print(balance*(1+rate))
```

Let me try to figure it out

```
print("What is your balance?")
balance = input()
print("The type of balance is:")
print(type(balance))
rate = 0.04
print("Your balance after one year is:")
print(balance*(1+rate))
```

- Output:

```
What is your balance?
1000
The type of balance is:
<class 'str'>
Your balance after one year is:
```

Type conversion

- After

```
balance = input()
```

- The variable balance is a string (type)
- And it cannot be multiplied by a float number
- But we can convert the string into an integer by the function int()
- Demo

Correct code

```
print("What is your balance?")  
balance = int(input())  
rate = 0.04  
print("Your balance after one year is:")  
print(balance*(1+rate))  
|
```


Boolean

- If I ask you now “Do you have more than \$100 cash with you now?”
- Which one is your answer?
 - A. Yes
 - B. No
 - C. 999
 - D. Cow
 - E. What
 - F. I am not going to treat you a meal
 - G. (Silent)

Boolean

- In Python

```
>>> my_cash = 50
>>> my_cash > 100
False
```

- Cash == 50, how can $50 > 100$?!
- In Python, the “>” operator is also different from math
 - In math “ $a > b$ ” is saying that a must be larger than b
- However, in most programming language, you need to evaluate the answer, saying “yes” or “no”
 - However, the real wording is “True” or “False”

Comparison Operators

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

```
True
```

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

```
False
```

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

```
True
```

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

```
True
```

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

```
False
```

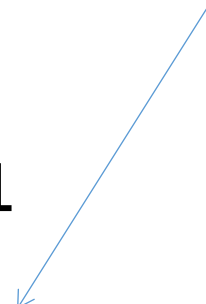
```
>>> False == False
```

```
True
```

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

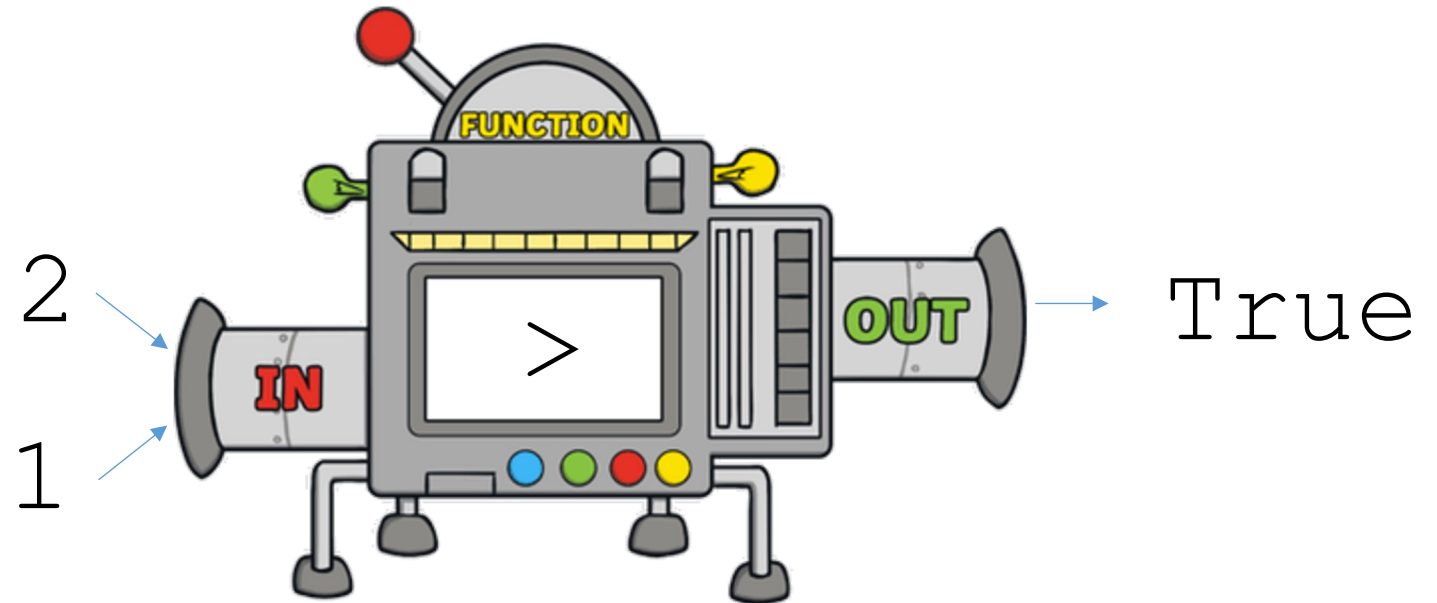
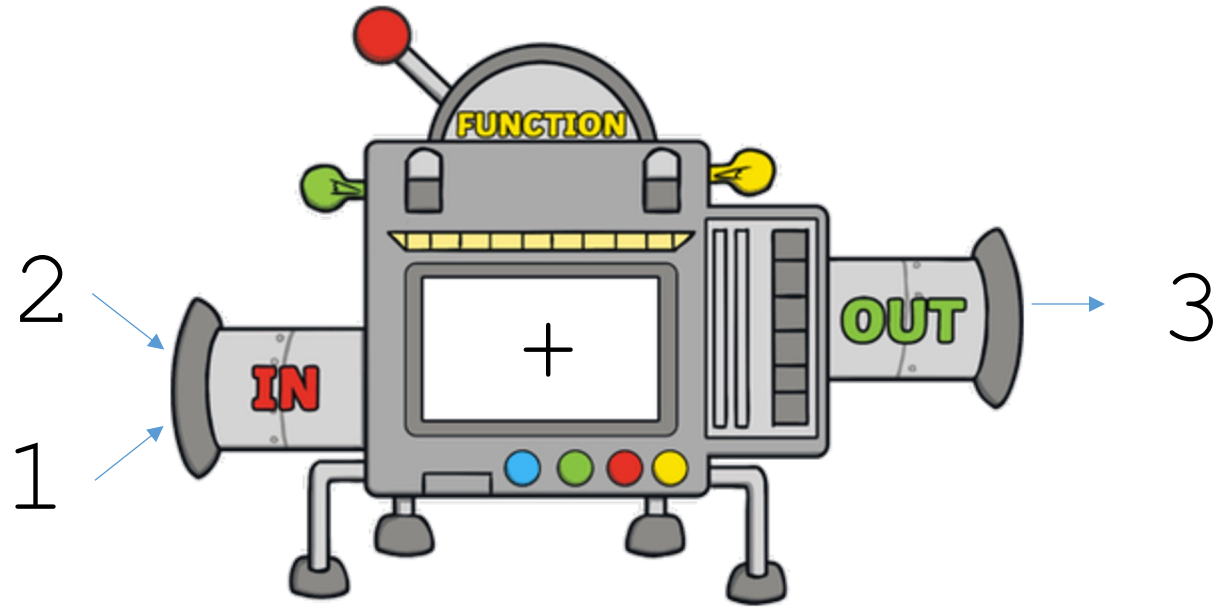
```
False
```

The very no. 1
trap for
programmers



Mentality change

- In most programming languages, you should think of “ $2 > 1$ ” in a way similar to “ $2 + 1$ ”



Old Enough To Drive?

```
>>> age = 20
```

```
>>> age > 18
```

```
True
```

```
>>> age = 14
```

```
>>> age > 18
```

```
False
```

Operators

- Logic:

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

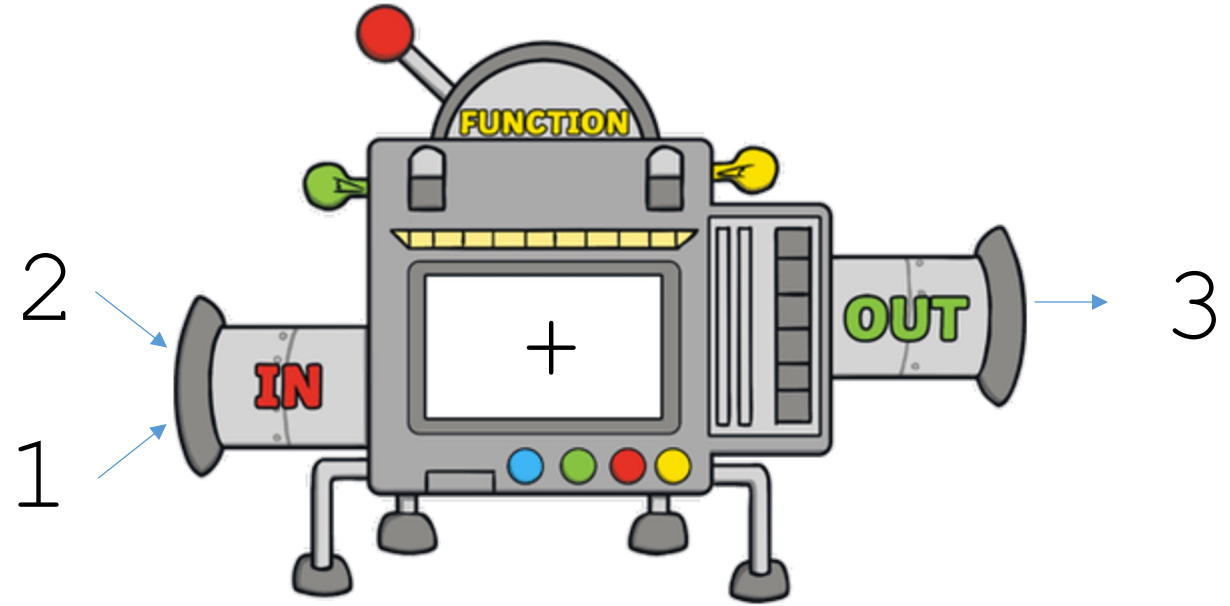
```
>>> True and False  
False
```

```
>>> 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

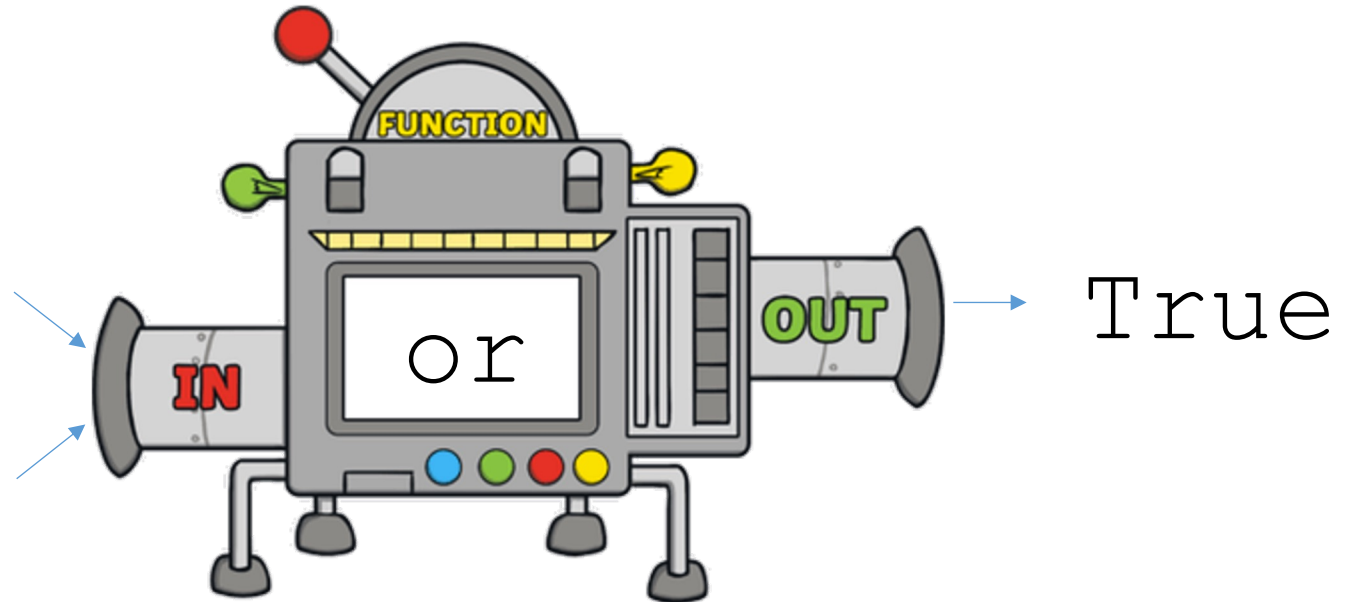
Mentality change

- In most programming languages, you should think of “True or False” in a way similar to “ $2 + 1$ ”



True

False



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

Dating Criteria

- E.g. If I would like to find a partner with a dating app
 - Cannot be too young or too old
- Older than 23yo but younger than 28yo

```
>>> age = 26
>>> age > 23 and age < 28
True
>>> age = 21
>>> age > 23 and age < 28
False
>>> age = 30
>>> age > 23 and age < 28
False
```