

CSE 101

# Computer Science Principle

## Lecture 03: Variables & Functions

---

March. 2023

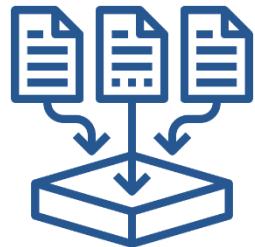
Prof. Francois Rameau

[francois.rameau@sunykorea.ac.kr](mailto:francois.rameau@sunykorea.ac.kr)



# What is Computer Science?

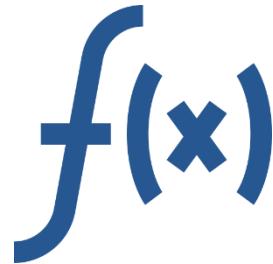
The previous class depicted a very high level of what we can do with programming. Today, we will start **CODING!**



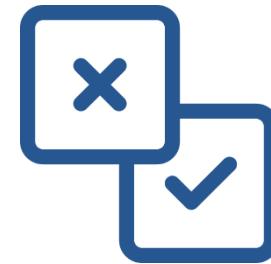
Variable



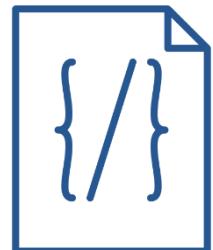
Arithmetic



Function



Boolean



Syntax

A very interesting dense and important class to learn the fundamentals of programming!

# - Expression and Arithmetic -

# Expression

## ! Terminology alert !

**Expression:** In programming, an expression is a value, or anything that executes and ends up being a value



## Let's see a few examples to understand better

### Example 1

"Hello, world!"

- It has a value
- In this case, it's a **string** (a sequence of characters)

### Example 2

5

- numbers are also expression
- 5 is an integer
- Recall that an integer is zero, or a positive or negative whole number with no fractional part

### Example 3

12.36

- **Floating-point** is a format that computers use to represent **real** numbers
- Recall that a real number is zero, or a positive or negative number that might have a fractional part

# Expression

## ! Terminology alert !

**Expression:** In programming, an expression is a value, or anything that executes and ends up being a value



## Let's see a few examples to understand better

### Example 4

`3 + 4`

- This expression is evaluated to 7
- This represents an addition
- Some of the simplest expressions in Python involve these arithmetic expressions

### Example 2

`len("hello world")`

- This expression calls the `len` function with the argument "hello world" and return the length of the string, which is 11.
- It returns a value so ... again ... it is an expression

### Example 3

`a>b`

- This expression evaluates to a boolean value, which is True if the value of `a` is greater than the value of `b`, and False otherwise.

# Expression

## ! Terminology alert !

**Expression:** In programming, an expression is a value, or anything that executes and ends up being a value



## At least three kinds of data in Python



Strings



Numbers



Boolean values

In computer programming, there are a wide variety of types of data because there are a wide variety of problems that computers can help solve

# Arithmetic in python

Let's consider this expression

$$2 + 9$$

Each numbers are  
called **operand**

The plus (+) is an **operator** – this is telling what  
to do with the input (in this case, add)

It is an **addition**

# Arithmetic in python

Python proposes by default a large panel of operators

Add

+

Subtract

-

Multiply

\*

Divide

/

Integer divide

//

(divides and rounds down to the nearest whole number)

Remainder

%

(gives the remainder of an integer division, also called "modulo" )

Exponential

\*\*

Note: this syntax is valid on Python but might differ on different languages

# Arithmetic in python

## Some examples

### Add

$11 + 5 \rightarrow 16$

### Subtract

$11 - 5 \rightarrow 6$

### Multiply

$11 * 5 \rightarrow 55$

### Divide

$11 / 5 \rightarrow 2.2$

### Integer divide

$11 // 5 \rightarrow 2$

### Remainder

$11 \% 5 \rightarrow 1$

(This example shows integer division.  
Any remainder is discarded)

### Exponential

$\ast \ast$

- The computer divides 11 by 5 and returns the remainder (which is 1), not the quotient (which is 2).
- The % is performing the "modulo operation"

# Arithmetic in python

## What happen when you have complex operations?

What's the answer?

$$8 \div 2(2 + 2) = ?$$

You can see how people vote. [Learn more](#)

1

57%

16

38%

0

3%

No idea

2%

**2,131 votes** • Poll closed

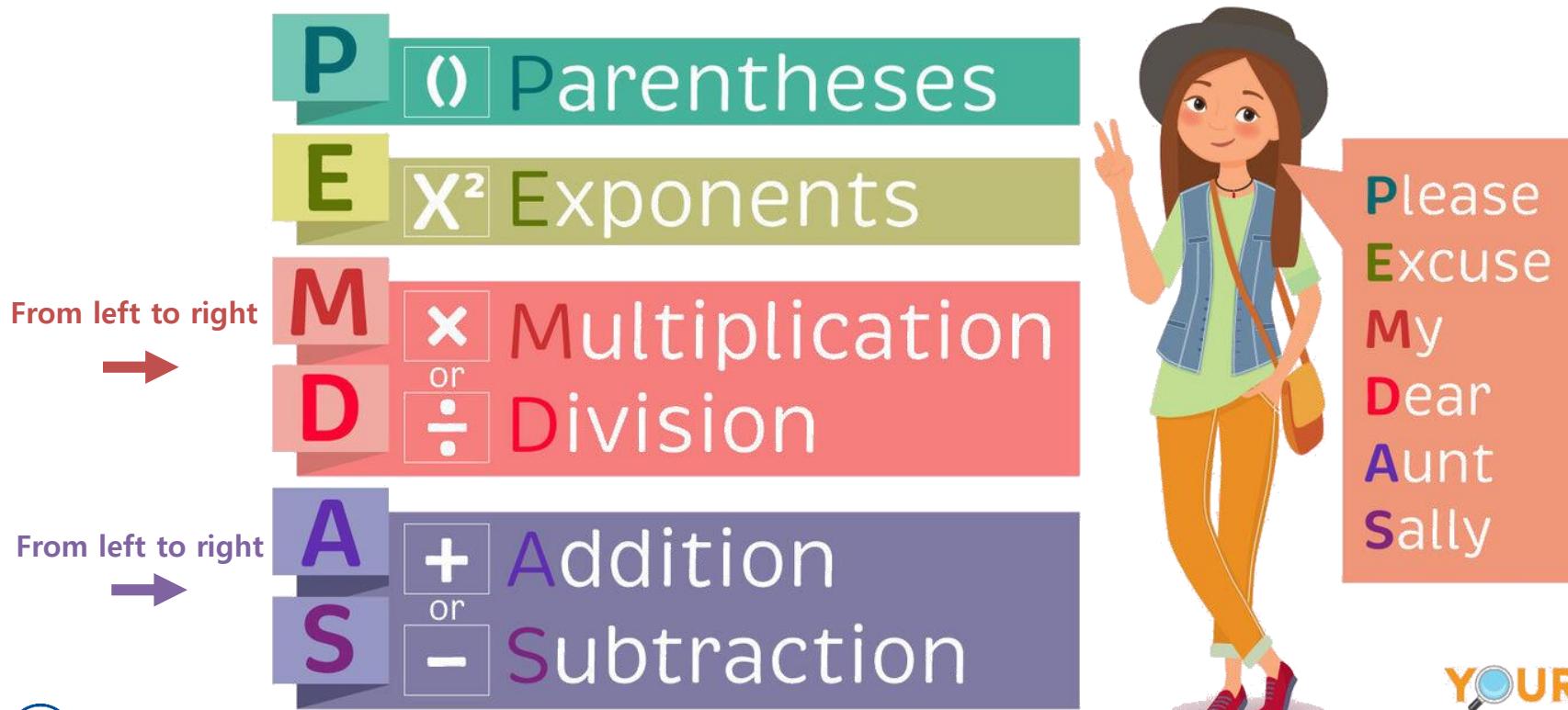


61 • 90 comments

# Arithmetic in python

What happen when you have complex operations?

$$2 * (3+2)-4/2 = ??$$



# Arithmetic in python



## What about the square root?

The `**` operator does exponentiation or raises a number to a power

For example: `2 ** 5` → 32 because  $2^5 = 32$

Recall raising a number to the power of  $\frac{1}{2}$  is the same as taking a square root

So `16 ** 0.5` would be the same as  $\sqrt{16}$  which is 4

# Arithmetic in python

## Operators on INT and FLOAT

$a + b \rightarrow \text{sum}$

$a - b \rightarrow \text{Difference}$

$a * b \rightarrow \text{Product}$

$a / b \rightarrow \text{Division}$



- If both are ints, result is int
- If either or both are floats, result is float
- Result is float

# - Variables -

# Variables

## ! Terminology alert !

**Variable:** In programming, a variable is a symbolic name that represents a value or a memory location where a value is stored.



It is the same concept as in mathematics  
Do you need an example?

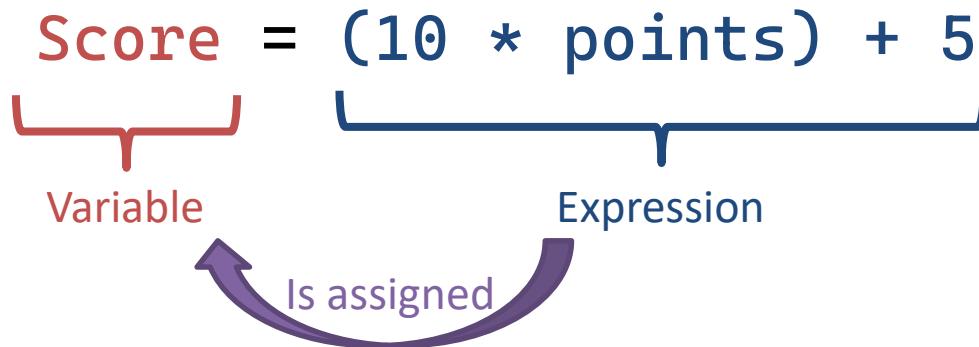
`billTotal = 10.50`

The variable **named billTotal** is **assigned** the value 10.50  
Now if we later refer to the **billTotal**, it will be 10.50

- In a program, variables can store a person's age, GPA, name, or almost any other kind of information
- The value is temporarily stored in the main memory of the computer while the program is running
- A variable is a kind of **identifier** because it identifies (names) something in the source code

# Assignment statement

When we give a value to a variable, we are writing an **assignment statement**. An assignment statement consists of a variable name, the equals sign, and a value or expression.



Other examples:

float

`length = 3.5`

integer

`total = 2 + 5`

string

`name= “Susan”`

# Assignment statement

After assigning a value to a variable, you can change the value of the variable with another assignment statement

```
total = 5  
... other code here ...  
total = 17 + 6  
... other code here ...
```

Variables can also appear on the right-hand side (RHS) of an assignment statement

```
next_year = this_year + 5  
total_earnings = income - taxes
```

# Assignment statement

In python variable do not have a particular fixed type associated to them

```
total = 5  
... other code here ...  
total = 6.0  
... other code here ...  
total = "six hundred"
```

The type of a variable can change over time in Python. But it is often not possible in other programming languages

## Check data type of a variable

```
type(variable)
```

*There are only two hard things in Computer  
Science: cache invalidation and naming things.*

-- Phil Karlton

# Variable: naming things!

It is important to choose variable names that are informative and helpful

Look at this code and try to guess what it is used for??

```
public relScore(m1: Mov, m2: Mov): number {
    const GW = 0.4;
    const YW = 0.1;
    const DW = 0.2;
    const WW = 0.3;

    let s = 0;
    if (m1.gen == m2.gen) {
        s += GW;
    }

    if (m1.yr == m2.yr) {
        s += YW;
    }

    if (m1.dir == m2.dir) {
        s += DW;
    }

    for (const w in m1.writers) {
        if (m2.writers.indexOf(w) !== -1) {
            s += WW;
            break;
        }
    }

    return s;
}

public movOnPg(m: Mov[], pg: number): Mov[] {
    const pp = 12;
    const s = (pg - 1) * pp;
    return m.slice(s, s + pg);
}
```

What about now?

```
public static movieRelationScore(movie1: Movie, movie2: Movie): number {
    const GENRE_WEIGHT = 0.4;
    const YEAR_WEIGHT = 0.1;
    const DIRECTOR_WEIGHT = 0.2;
    const WRITER_WEIGHT = 0.3;

    let score = 0;
    if (movie1.genre == movie2.genre) {
        score += GENRE_WEIGHT;
    }

    if (movie1.year == movie2.year) {
        score += YEAR_WEIGHT;
    }

    if (movie1.director == movie2.director) {
        score += DIRECTOR_WEIGHT;
    }

    for (const writer in movie1.writers) {
        if (movie2.writers.indexOf(writer) !== -1) {
            score += WRITER_WEIGHT;
            break;
        }
    }

    return score;
}

public static moviesOnPage(movies: Movie[], page: number): Movie[] {
    const amountPerPage = 12;
    const start = (page - 1) * amountPerPage;
    return movies.slice(start, start + page);
}
```

# Variable: naming things!

A python variable may contain:



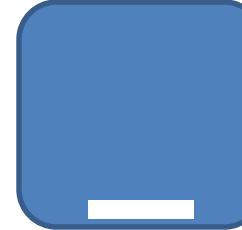
Lowercase  
letters



Uppercase  
letters



Digits



Underscore

Example of valid variable's name:

```
a_Weird_Variable_1 = 25
```

The first character must be a letter or an underscore:



```
1_wrong_variable_name = 1
```



```
_correct_variable_name = 1
```



# Variable: naming things!

Python is case sensitive!

First\_Name



first\_Name



FIRST\_NAME

A few keywords already have a pre-defined meaning and cannot be used as variables:

A few examples:

if

for

and

A variable's name cannot contain space, use underscore instead

# Variable: naming things!

Theoretically, you can name variables, functions, classes, etc.  
the way you like. BUT try to be consistent!

camelCase



oneHump

snake\_case



danger\_noodle

PascalCase



ThePhilosophy

kebab-case



tomato-meat-oignon

To be sure to do the best choice you can follow the naming  
convention proposed by PEP-8

# Variable: naming things!

## Naming convention proposed by PEP-8



Type	Naming Convention	Examples	Type	Naming Convention	Examples
Function	Use a lowercase word or words. Separate words by underscores to improve readability.	function, my_function	Constant	Use an uppercase single letter, word, or words. Separate words with underscores to improve readability.	CONSTANT, MY_CONSTANT, MY_LONG_CONSTANT
Variable	Use a lowercase single letter, word, or words. Separate words with underscores to improve readability.	x, var, my_variable	Module	Use a short, lowercase word or words. Separate words with underscores to improve readability.	module.py, my_module.py
Class	Start each word with a capital letter. Do not separate words with underscores. This style is called <a href="#">camel case or pascal case</a> .	Model, MyClass	Package	Use a short, lowercase word or words. Do not separate words with underscores.	package, mypackage
Method	Use a lowercase word or words. Separate words with underscores to improve readability.	class_method, method			

Real Python

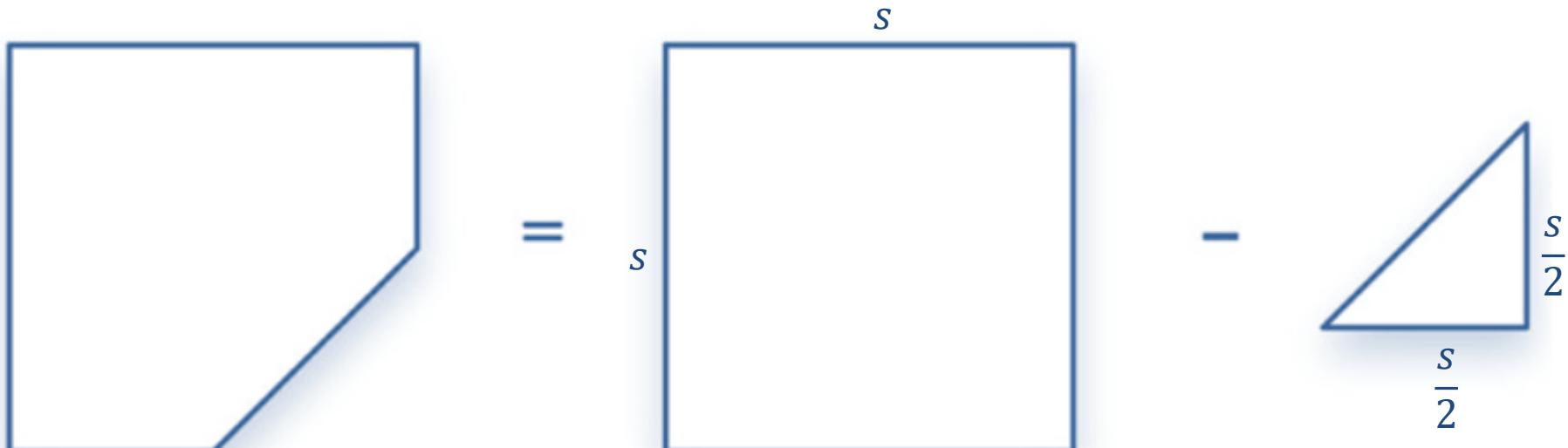
# **Variables & Arithmetic**

**- Example 1 -**

## **The broken countertop**

## Example: Area calculation

We want to compute the **area** of a square countertop with one corner cut off, as shown here



$$a_{result} = a_{square} - a_{triangle}$$

$$a_{square} = s^2$$

$$a_{triangle} = \frac{(s/2)(s/2)}{2}$$

Note: we assume that the triangular cut-out begins halfway along each edge

## Example: Area calculation

If it is 100 cm long on each side, we can write a statement like this:

```
area = 100**2 - 50*50/2
```

**Note that this code has a few issues:**

- 1. It's just a formula with no explanation of what the numbers mean
- 2. The code works only for countertops exactly 100 cm long. What if we had countertops of other sizes?



## Example: Area calculation (comment)

To address the first problem, we can add comments in the code  
to explain what is happening

```
# area = area of square - area of triangle  
# area of triangle is 1/2 base*height  
area = 100**2 - 50*50/2
```

The lines beginning with the `#` symbol are called **comments**

- Comments are notes that the programmer writes to explain what the program does
- Comments do not affect the input or output of the program or anything about how it runs

# Example: Area calculation (comment)

## Multiple ways to comment

### Block comment

```
# Compute the area of a square with a chopped corner  
# as follow, area = area of square - area of triangle  
# Note that the area of the triangle is 1/2 base*height  
area = 100**2 - 50*50/2
```

### Inline comment

```
x = 5 # This is an inline comment
```

### A few recommendations (PEP-8):

- Limit the line length of comments and docstrings to 72 characters.
- Use complete sentences, starting with a capital letter.
- Make sure to update comments if you change your code.
- Start comments with a # and a single space, like block comments.

# Example: Area calculation (comment)

## Documentation string

```
"""Solve quadratic equation via the quadratic formula.
```

A quadratic equation has the following form:

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

There are always two solutions to a quadratic equation:  $x_1$  &  $x_2$ .

```
"""
```

$$x_1 = \frac{(-b + \sqrt{b^2 - 4ac})}{2a}$$

$$x_2 = \frac{(-b - \sqrt{b^2 - 4ac})}{2a}$$

```
return x_1, x_2
```

- Surround docstrings with three double quotes on either side, as in  
    `"""This is a docstring"""`.
- Write them for all public modules, functions, classes, and methods.
- Put the `"""` that ends a multiline docstring on a line by itself

## Example: Area calculation

Now let's address the other issue: make it generalizable and work with other countertop sizes

```
side = 100
square = side**2
triangle = (side / 2)**2 / 2
area = square - triangle
```

To compute the area for a countertop of a different size (e.g. 200), simply change the first line:

```
side = 200
```

Do we still need comments to understand the code?  
Probably not! It is a **self-documenting code**

The spacing in between variables, numbers, and operator is optional, but is included here to make the formulas easier to read

# Example: Area calculation (input)



With the previous solution, the side value has to be set in the code.  
Instead, can we propose that the user input this value?

## Input statement

Do this by writing an **input** statement – An input statement reads a **string** from the keyboard. As part of an input statement, the programmer must give a **prompt** message telling the user what to enter. For example:

```
name = input("What is your name? ")
```

The person's name will be assigned to the name variable. You could also say that we are saving the person's name in the name variable

## Example: Area calculation (input)



In the case of the area calculation, the user should enter a number, not a string ... we need to do a **cast**

**Definition:** In programming, a **cast** (also known as type conversion or type casting) is a way to explicitly convert a value from one data type to another.

## Type casting in Python

Python supports several built-in functions for type casting, including:

- `int()`: Converts a value to an integer.
- `float()`: Converts a value to a floating-point number.
- `str()`: Converts a value to a string.
- `bool()`: Converts a value to a Boolean.

```
# Explicit type casting
num_str = "42"
num_int = int(num_str)

# Implicit type casting
result = 3 + 4.5
```

# Example: Area calculation (input)



Back to our area calculation case, let's apply what we have learned

- To collect a floating-point number, use:

```
side = float(input("Enter side length: "))
```

- If we wanted to only allow integer numbers as input, we would use:

```
side = int(input("Enter side length: "))
```

- The type chosen – **int** vs. **float** – depends on the application  
For our program, read in a float so the user could enter a fraction of a centimeter if desired
- The last step is how to display the final result on the computer screen

# Example: Area calculation (print)



We will use the `print()` function to output the results in the terminal!

## `print()`

The syntax to print a basic message is simply this:

```
print("your text here")
```

Any text printed with additional print commands will appear on a new line. Other type of data can be printed

```
test_variable = 4.232  
print(test_variable)
```

# Example: Area calculation (print)



- To print a number, it must first be converted into a string, like so:

```
print("The area is " + str(area))
```

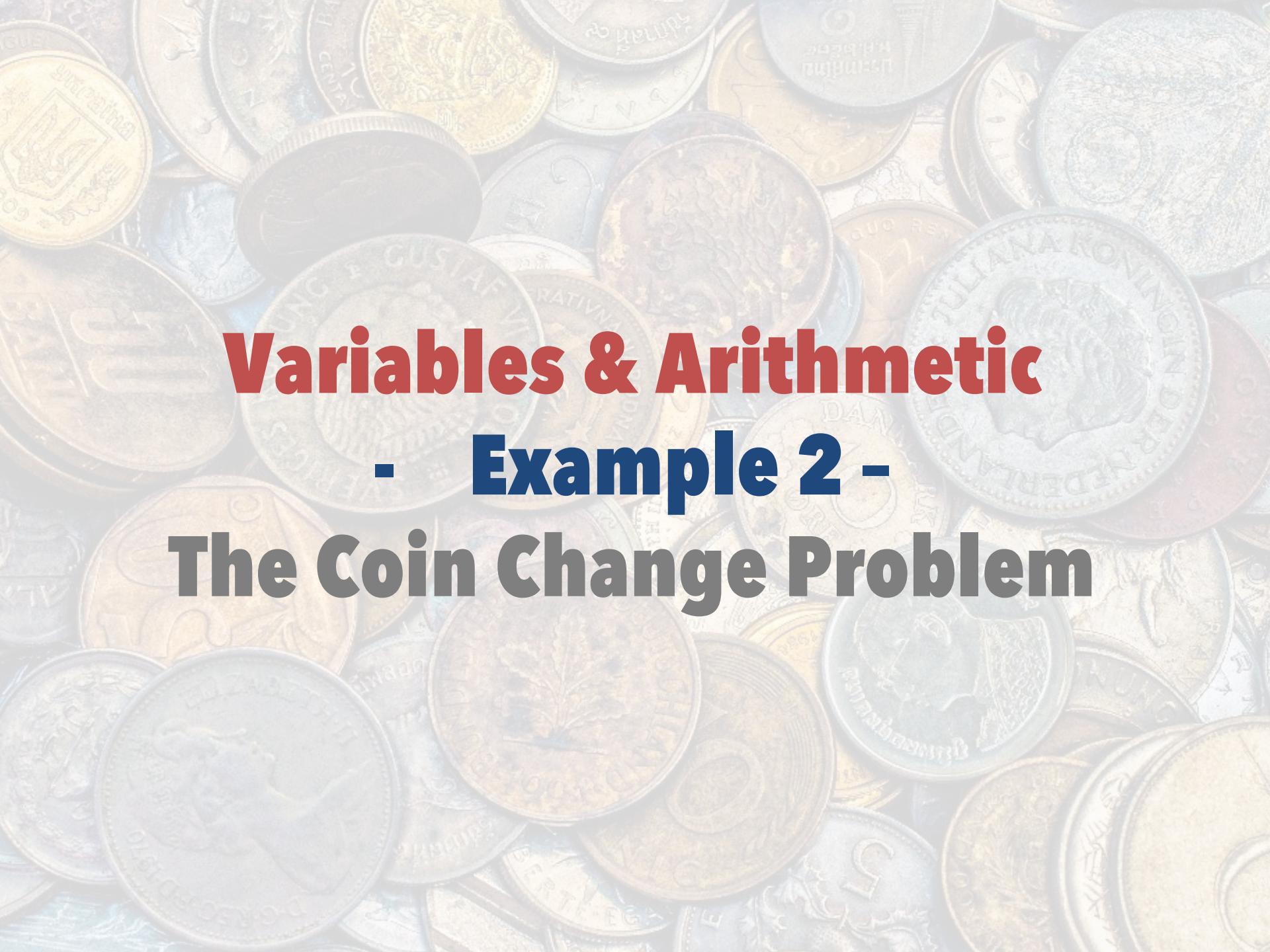
- When used in this fashion, the + symbol performs **string concatenation**
  - This simply means Python will join the two strings together into one

## Example: countertop.py

```
# This program prints the area of a
# countertop formed by cutting the
# corner off a square piece of material

side = float(input("Enter side length: "))
square = side**2
triangle = (side/2)**2 / 2
area = square - triangle
print("The area is " + str(area))
```





# **Variables & Arithmetic**

## **- Example 2 -**

## **The Coin Change Problem**

# The coin change problem

Here is an example that uses the modulo (remainder) operator with integer division

## Problem

Given a total number of cents, the computer should print how many dimes, nickels, and pennies are needed to make that change while minimizing the number of coins



- The code will use several variables
- It will also need the **str** command to print variables containing numbers to the screen
- Recall that **str** converts a number to a string so that it can be concatenated with other strings

# The coin change problem

```
cents = int(input("Enter the number of cents: "))

dimes = cents // 10
cents = cents % 10
nickels = cents // 5
cents = cents % 5
pennies = cents

print("That number of cents is equal to " +
      str(dimes) + " dimes, " + str(nickels) +
      " nickels and " + str(pennies) + " pennies.")
```

# **Variables & Arithmetic**

**- Example 3 -**

## **Print multiline text**

# Escape sequences

Escape sequences in programming languages like Python allow printing characters (symbols) on the screen that perform special functions

Code	Result/Output	Description
\'	Single Quote	Add single quote with in a String
\\"	Backslash	Insert single Back Slash
\n	New Line	\n takes the cursor to first position of a new line
\r	Carriage Return	\r takes the cursor to the first position of the same line
\t	Tab	\t add spaces of 8 characters
\b	Backspace	\b takes the cursor one position backward
\f	Form Feed	Form Feed is page breaking ASCII control character
\ooo	Octal value	Octal value
\xhh	Hex value	Hex value

# Escape sequences

```
print('There was an old man with a beard\n\\
Who said, \"It\\'s just how I feared!\"\n\\
\\tTwo owls and a hen\n\\
\\tFour larks and a wren\n\\
Have all built their nests in my beard.')
```

## Output:

```
There was an old man with a beard
Who said, "It's just how I feared!"
    Two owls and a hen
    Four larks and a wren
Have all built their nests in my beard.
```

# **- Functions -**

# Additional arithmetic in Python

We have seen a few simple examples of how to use basic arithmetic.  
But many more are available if we **import** the `math` module:

```
import math
```

Then, you can for instance access the value of the constant  $\pi$

```
test = math.pi * 2
```

A **Python module** is a file consisting of Python source code (we will talk about library later, no worries)

# Functions

Python's math module contains code related to mathematical functions

## Constants

$\pi, e$ , etc.

## Functions

$\cos x, \log_2$   
etc.

In *programming*, a **function** is a name given to a set of statements  
that perform a well-defined task

## Example

the **input** function  
prints the value  
entered by the user

Parentheses contain  
parameters

Prints the value

print() is also a function

print("hello world")

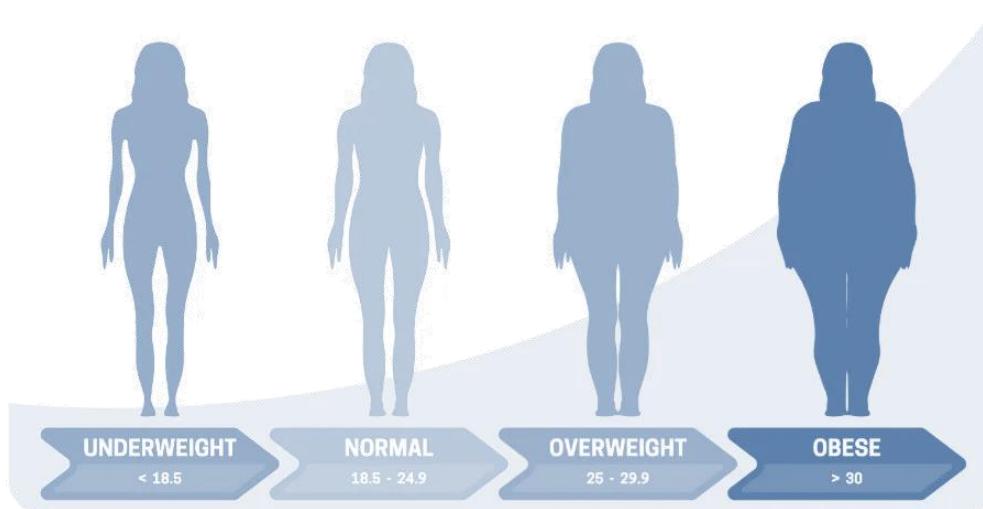
Function  
Name

Parameters separated by  
commas

multiple arguments!

# Functions: Example BMI

**BODY MASS INDEX (BMI) =  $\frac{\text{WEIGHT Kg}}{\text{HEIGHT m}^2}$**



## Problem

- We want to create a program that calculates and prints a person's BMI based on entered values.
- However, we want to ensure the BMI is printed with 3 digit decimal precision (e.g. 19.421). By default it will print 15 decimal digit, e.g. 19.421004314691235

# Functions

- To print a number to a designed number of digits, use the **format** function
- Suppose there is a variable **total** to be printed with two decimal places. Here is how to do it:

```
print("Total: " + "{:.2f}".format(total))
```

- If we wanted four digits, we would write **{:.4f}** instead
- Note that when using the **format** method, do not also use **str** to print a number (also in this case, the concatenation is not needed)

# bmi.py

```
weight = float(input('Enter weight in pounds: '))
feet = float(input('Enter feet portion of height: '))
inches = float(input('Enter inches portion of height: '))

total_inches = feet * 12 + inches

bmi = (weight * 703) / total_inches ** 2

print('Your BMI is ' + str(bmi))
print('Your BMI is ' + '{:.3f}'.format(bmi))
```

## Notes:

- The first print statement gives the BMI with full accuracy. The second print statement rounds to 3 decimal places.
- The blank lines are present to make the code more readable. They do not affect program execution in any way.

# Other common functions

Some examples:

```
type(45)  
int(34.56)  
float(45)  
str(3421)  
len('apple')  
round(2.32)  
abs(-45)  
pow(2, 3)  
help(pow)
```

```
import math  
math  
math.log(10)  
math.log10(10)  
math.log10(1e6)  
radians = 0.7  
math.sin(radians)  
math.sqrt(3)  
...
```

```
import random  
random.random()  
random.randint(0,100)
```

Try these on a Python Console or as part of a program

# Function composition

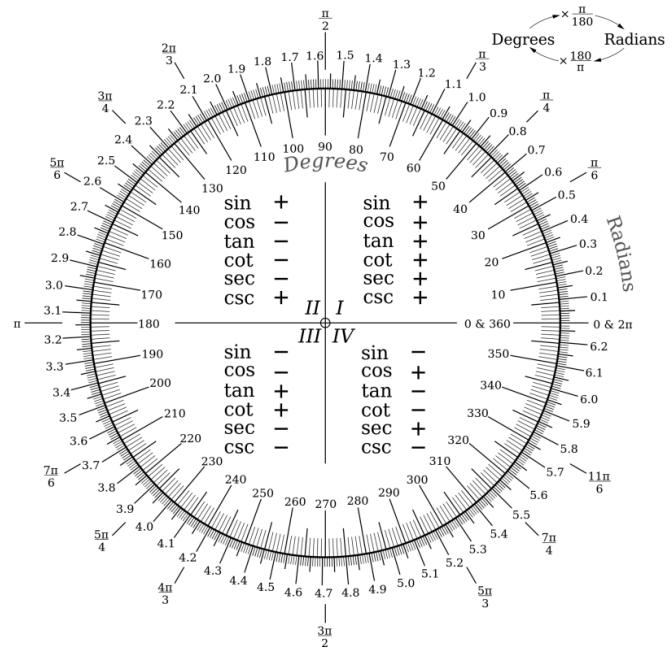
We can compose functions as is done in mathematics, e.g.,  $f(g(x, y))$

```
import math
radians = 0.7
print(math.degrees(radians))

math.radians(math.degrees(radians))

radians = 0.3
math.acos(math.cos(radians))

pow(abs(-3), round(5.6))
```



# Write your own functions!

## ! Terminology alert !

**Function:** In programming, a function is a block of code that performs a specific task, takes input and returns output.



Now that you know how to use functions, make your own!

## But why?



### 1. Modularity:

Functions break down a complex program into smaller, more manageable parts.



### 4. Code reuse:

Functions can be reused across different parts of the program or in other programs, saving time and effort.



### 2. Abstraction:

Functions hide implementation details, making it easier to reason about the program.



### 5. Testing:

Functions can be tested in isolation, making it easier to identify and fix bugs.

# DONT REPEAT YOURSELF



First rule of coding

# Create a new function

## A function in maths: A 2 steps process

1. Define a function, once

$$f(x, y) = x \times y + 1$$

2. Apply/Use/Invoke/Call the function, as many times as desired

$$f(2, 3) = 2 \times 3 + 1 = 7$$

## A function in programming: again 2 steps

1. Define a function, once

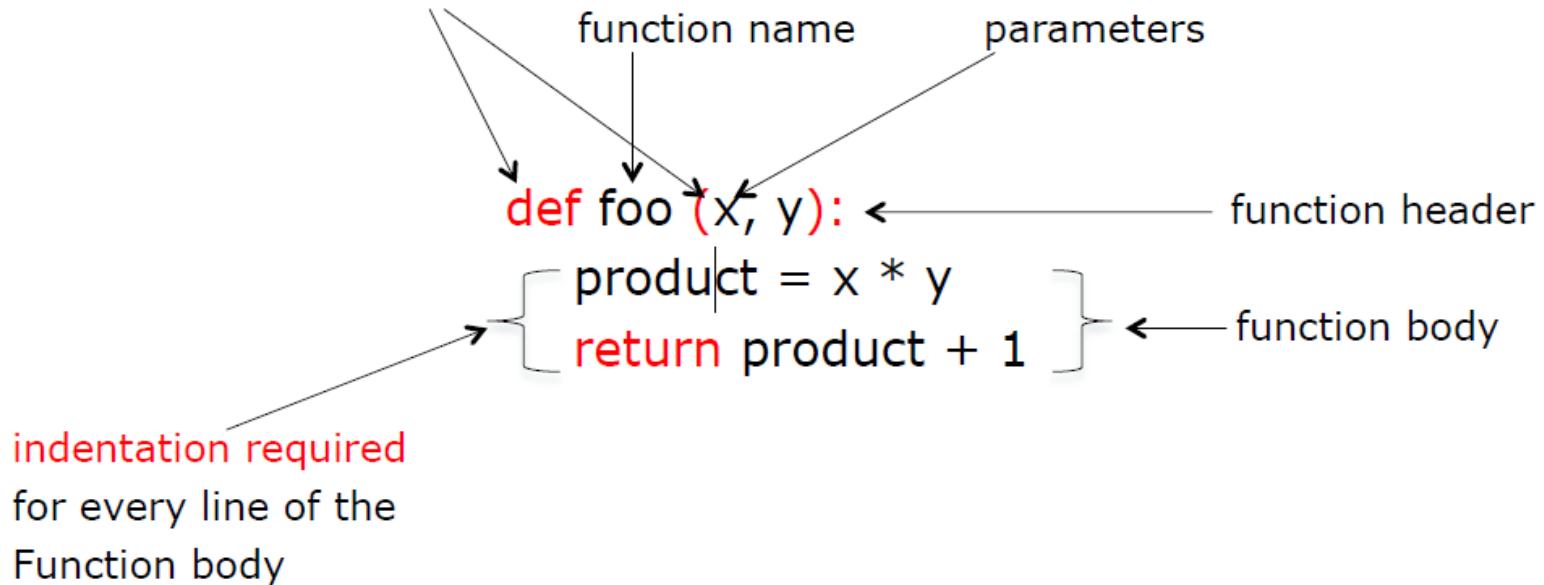
```
def f(x, y):  
    return x * y + 1  
indentation
```

2. Apply/Use/Invoke/Call the function, as many times as desire

```
f(2, 3)
```

# Defining/calling a function

required elements



foo(2, 3)

() after a function name means a **function call**, in this case with two arguments

# Parameters and arguments

A function can have zero or more parameters

Functions may be defined with *formal parameters*

```
def multAdd(a, b, c):
```

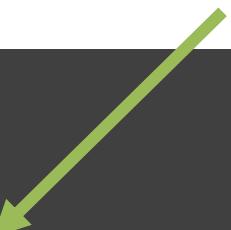
```
    return a * b + c
```

```
print(multAdd(1, 2, 3))
```

```
print(multAdd(2.1, 3.4, 4.3))
```

```
print(multAdd(abs(pow(2,3)), 3.2 + 2.3, 45.34))
```

Functions can be then called with *actual arguments*. How many? As many as the function needs!



# Program: flow of execution

## Code

```
def message():
    print(1)
    message1()
    print(2)
def message1():
    print('a')
    message2()
    print('b')
def message2():
    print('middle')

message()
```

Note: this is a *program* with three functions and it starts with a call to *message*.

## Output:

```
1
a
middle
b
2
```

# Void vs fruitful functions

## Fruitful functions

A **fruitful function** is a function that **returns a value**

```
# fruitful function
def square(n):
    return n * n

print(square(3))
```

What will get printed when this code runs?

## Void functions

A **void function** is a function that **does not returns a value**

```
# void function
def announce(message):
    print(message)

announce('hello!')
```

What would happen if you tried to print your announce call?

```
print(announce('hello!'))
```

Since announce does not have a return statement, it returns **None**. So the statement `print(announce('hello!'))` will print **None**.

# bmi\_v2.py

```
# Function definition
def bmi(w, h):
    return (w * 703) / (h ** 2)

# main is to use the function defined above.
def main():
    weight = float(input('Enter weight in pounds: '))
    feet = float(input('Enter feet portion of height: '))
    inches = float(input('Enter inches portion of height: '))

    total_inches = feet * 12 + inches
    my_bmi = bmi(weight, total_inches)
    print('Your BMI is ' + '{:.3f}'.format(my_bmi))

# This sets up a call to the function main.
main()
```

Note how the program is organized

# Why functions? Abstraction

- One of the most important concepts in computer science is **abstraction**
  - Give a **name** to a group of statements and use it.
  - E.g. `bmi(...)`
- From the outside, the details are hidden
  - Only care that calling this function will do a desired computation
- Functions thus allow complex problems to be solved by dividing it into smaller, more manageable sub-problems
  - This process is called **problem decomposition** (also **functional decomposition**)

## bmi\_v3.py

Here is an alternative way of implementing the bmi function

It illustrates proper indentation and relies on **two local variables, numerator and denominator**

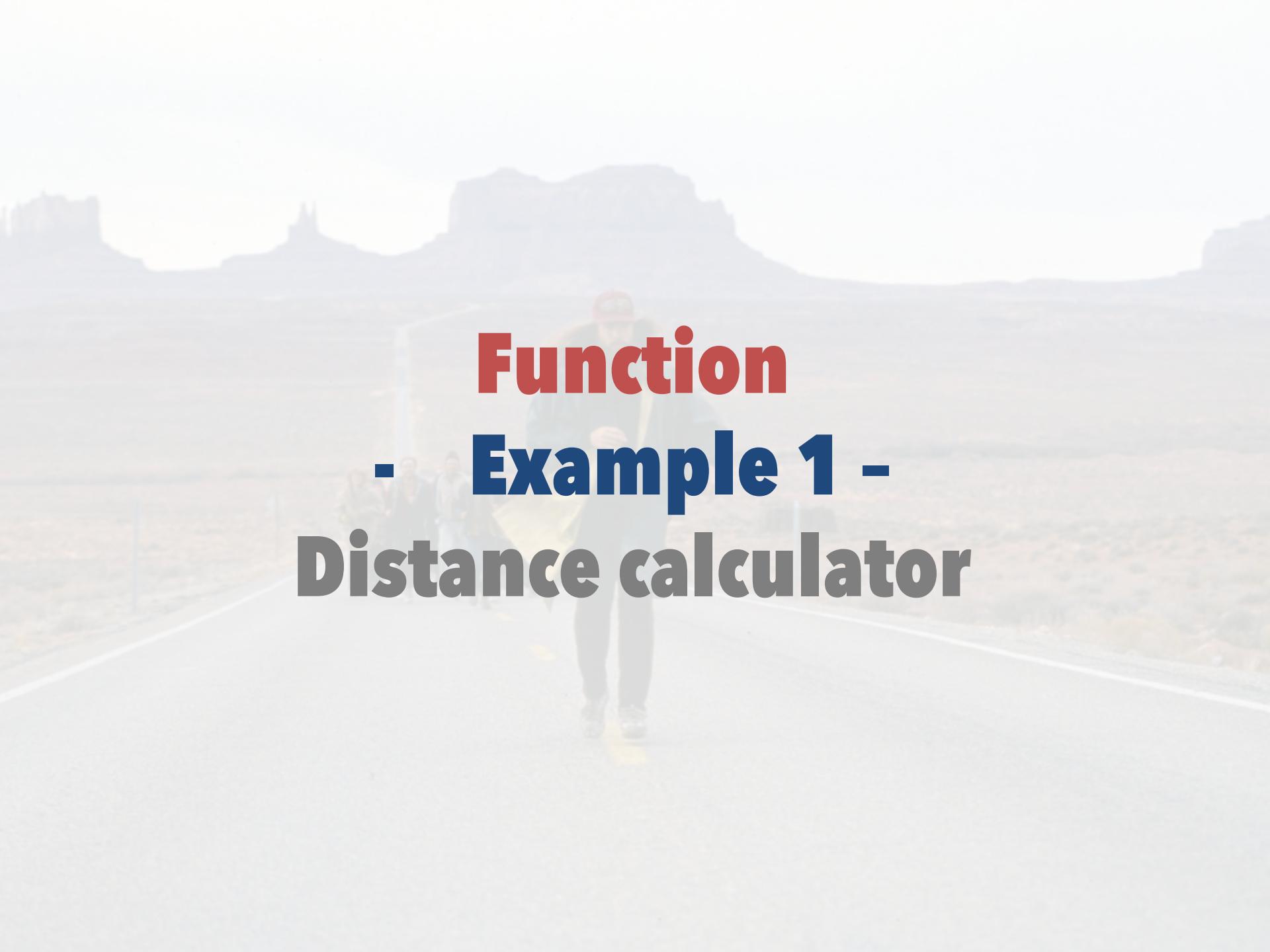
```
def bmi(w, h):
    numerator = w * 703
    denominator = h ** 2
    return numerator / denominator

def main():
    weight = float(input('Enter weight in pounds: '))
    feet = float(input('Enter feet portion of height: '))
    inches = float(input('Enter inches portion of height: '))

    total_inches = feet * 12 + inches
    my_bmi = bmi(weight, total_inches)
    print('Your BMI is ' + '{:.3f}'.format(my_bmi))

main()
```

Note: A **local variable** is a variable accessible only inside the function where it is created

A blurred background image of a person walking away from the camera on a road through a desert landscape with mountains in the background.

# **Function**

- Example 1 -**

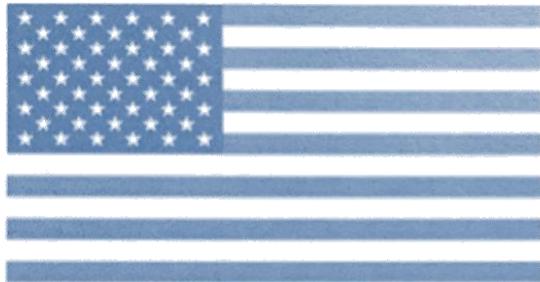
# **Distance calculator**

# Example: distance calculator

## Problem:

A distance traveled is provided in miles, yards, and feet (e.g. 3 miles, 68 yards, 16 feet). We need this to be converted to total inches traveled and print the result.

This requires some unit conversions



**1 foot = 12 inch**

**1 yard = 3 feet = 36 inches**

**1 mile = 1760 yards = 5280 feet**



**1 cm = 10 mm**

**1 m = 100 cm = 1000 mm**

**1 km = 1000 m**

Finally, to print a comma every three digits we can use the formatting string '{:,}' when printing an integer

# Example: distance calculator

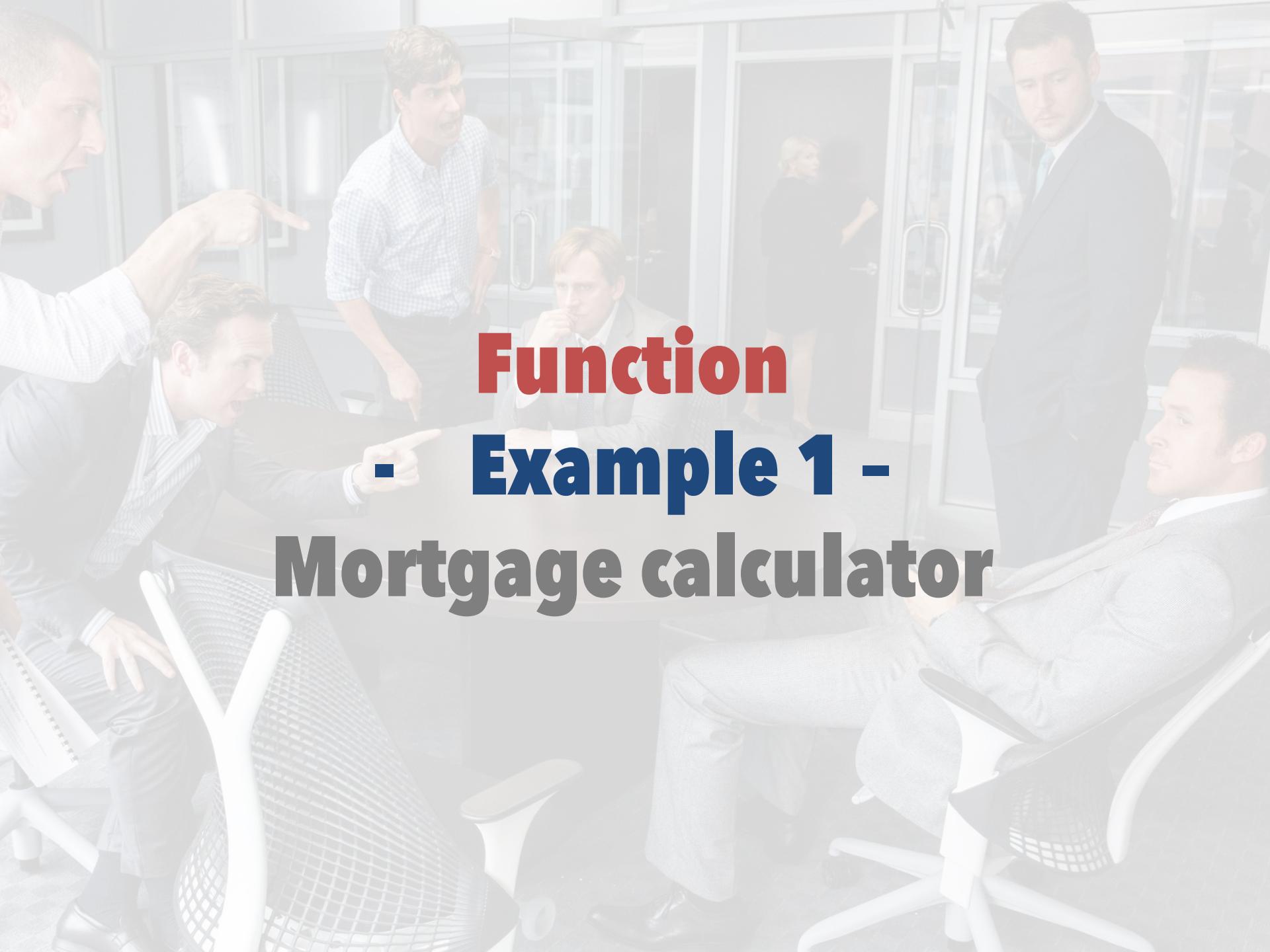
```
def distance(m, y, f):
    return (m * 5280 * 12) + (y * 3 * 12) + (f * 12)

def main():
    miles = int(input('Enter the number of miles: '))
    yards = int(input('Enter the number of yards: '))
    feet = int(input('Enter the number of feet: '))

    inches = distance(miles, yards, feet)

    print('Distance in inches: ' + '{:,}'.format(inches))

main()
```



# **Function**

## **- Example 1 -**

# **Mortgage calculator**

# Example: mortgage calculator

## Problem:

The monthly payment on a fixed-rate mortgage can be calculated using this formula:

$$\text{payment} = (r * P) / (1 - (1 + r)^{-n})$$

Where:

- $P$  is the principal (the amount we borrowed)
  - $r$  is the monthly interest rate as a decimal (i.e., the annual interest rate as a decimal divided by 12)
  - $n$  is the number of months the loan will last
- 
- To include a comma every three digits and have 2 decimal precision, write the format string as: '`{ : , .2f}`'
  - Also, a format string can be saved in a variable if it's needed to format several numbers in the same way: `fmt = '{ : , .2f}'`

Now we will create a function to  
compute the payment

# Example: mortgage calculator

```
def main():
    principal = float(input('Enter principal: '))
    annual_rate = float(input('Enter annual interest rate as a percentage: '))
    years = int(input('Enter term of mortgage in years: '))

    payment = monthly_payment(principal, annual_rate / 12 / 100, years * 12)
    totalPaid = payment * years * 12
    totalInterest = totalPaid - principal

    fmt = '{:.2f}' # formatter string
    print('Principal: $' + fmt.format(principal))
    print('Annual interest rate: ' + fmt.format(annual_rate) + '%')
    print('Term of loan in years: ' + str(years))
    print('Monthly payment: $' + fmt.format(payment))
    print('Total money paid back: $' + fmt.format(totalPaid))
    print('Total interest paid: $' + fmt.format(totalInterest))

main()
```

# Multi-output functions

A multi-output function in Python is a function that returns more than one value

```
def split_string(s):
    """
    Splits a string into two parts: the first word and the rest of the string.
    Returns a tuple containing the first word and the rest of the string.
    """
    words = s.split()
    if words:
        return words[0], ' '.join(words[1:])
    else:
        return '', ''

s = 'Hello, world! This is a test.'
first_word, rest_of_string = split_string(s)
print(f"First word: {first_word}")
print(f"Rest of string: {rest_of_string}")
```



# **- Conditional statement -**

# Conditional statements

We often have to make decisions in life

**If** I miss the bus, **then** I need to wait for the next, **or else** I walk to my destination

Similarly, in programming, an algorithm needs to make decisions. These decisions are taken using **conditional statements**

## ! Terminology alert !

**Conditional statement:** A conditional statement in programming is a statement that allows a program to make decisions and execute different actions based on certain conditions.



# Conditional statements

Often an algorithm needs to make a decision

The steps which are executed next depend on the outcome of the decision

## Example

If the income is above a certain minimum, use one tax rate; otherwise, use a lower rate

In Python, an **if statement** allows testing conditions and executing different steps depending on the outcome

# Conditional statements: Example 1

Suppose part-time students (< 12 credits) at a fictional college pay \$600 per credit and full-time students pay \$5,000 per semester.

Let's use an if statement to write a short program that implements this logic

```
numCredits = int(input('Enter number of credits: '))

if numCredits < 12:
    cost = numCredits*600
    print('A student taking ' + str(numCredits) + \
          ' credits is part-time and will pay $' + \
          str(cost) + ' in tuition.')

else:
    print('A student taking ' + str(numCredits) + \
          ' credits is full-time and will pay \n \
          $5,000 in tuition.') 
```

# Conditional statements

If statements can also appear in functions

```
def tax_rate(income):
    if income < 10000:
        return 0.0
    else:
        return 5.0
```

The value returned by the function depends on the value passed as an argument to the parameter

Things to note about the **if** statement:

- The words **if** and **else** are keywords
- There is a **colon (:)** at the end of the if and else **clauses**
- The statements to be executed are **indented**

# Multi-way if-statements

When an algorithm needs to choose among more than two alternatives, it can use **elif** clauses

**Note:** **elif** is short for “else if”

## Example

This function distinguishes between three tax brackets:

```
def tax_rate(income):
    if income < 10000:
        return 0.0
    elif income < 20000:
        return 5.0
    else:
        return 7.0
```

**Note:** you can use as many **elif** as you need!

# Match

When you have a large number of multiple cases you might prefer to use the function `match` instead of multiple `elif`

## Using `elif`

```
def switch(lang):
    if lang == "JavaScript":
        return "You can become a web developer."
    elif lang == "PHP":
        return "You can become a backend developer."
    elif lang == "Python":
        return "You can become a Data Scientist"
    elif lang == "Solidity":
        return "You can become a Blockchain developer."
    elif lang == "Java":
        return "You can become a mobile app developer"

print(switch("JavaScript"))
print(switch("PHP"))
print(switch("Java"))
```

## Using `match`

```
lang = input("What's the programming language" + \
            "\n you want to learn? ")

match lang:
    case "JavaScript":
        print("You can become a web developer.")
    case "Python":
        print("You can become a Data Scientist")
    case "PHP":
        print("You can become a backend developer")
    case "Solidity":
        print("You can become a Blockchain developer")
    case "Java":
        print("You can become a mobile app developer")
```

**Note:** `elif` is more versatile but `match` is easier to maintain and might be slightly faster

**- Boolean expression-**

# Boolean expressions

The expressions inside `if` and `elif` statements are often evaluated as Boolean expressions, which means that their result is either `True` or `False`. An expression that evaluates to `True` or `False` is called a **Boolean expression**.

Boolean expressions often involve **relational operators**:

- equal to / not equal to
- greater than / greater than or equal to
- less than / less than or equal to

# Boolean expressions

The notation  $\geq$  means “greater than or equal to” and is one of six **relational operators** supported by Python:

Mathematical Operator	Python Equivalent	Meaning
=	==	is equal to
$\neq$	!=	is not equal to
>	>	is greater than
$\geq$	$\geq$	is greater than or equal to
<	<	is less than
$\leq$	$\leq$	is less than or equal to

## Example: Overtime Calculator

Someone who works more than 40 hours a week is entitled to “time-and-a-half” overtime pay

- How can the following be determined?
  1. Whether an employee is entitled to overtime pay
  2. If so, how much are they paid?

For #1 use an if statement // For #2 a different calculation is required depending on whether the employee will earn overtime pay or not

Regular pay formula:

$$\text{hourly wage} \times \text{hours worked}$$

The overtime formula has two parts:

- The pay for the first 40 hours
- The pay for additional overtime hours ( $1.5 \times \text{hourly wage}$ )

# Example: Overtime Calculator

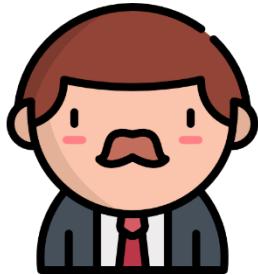
```
def compute_pay(hours, wage):
    if hours <= 40:
        paycheck = hours * wage
    else:
        paycheck = 40 * wage + (hours - 40) * 1.5 * wage
    return paycheck

def main():
    hours_worked = float(input('Enter # of hours worked: '))
    hourly_wage = float(input('Enter hourly wage: '))

    pay = compute_pay(hours_worked, hourly_wage)
    print('Your pay is $' + '{:.2f}'.format(pay))

main()
```

# Example: Hiring decision



A hiring manager is trying to decide which candidates to hire

Each candidate is evaluated on 3 criterions



A **GPA** of at least 3.3 is worth 1 point



An **interview score** of 7 or 8 (out of 10) is worth 1 point; a score of 9 or 10 is worth 2 points



An **aptitude test** score above 85 is worth 1 point

Hiring decisions are then based on point totals:

- 0, 1 or 2 total points: Not hired
- 3 total points: hired as a Junior Salesperson
- 4 points: hired as a Manager-in-Training

# Example: Hiring decision

- Next slide will show a function that takes these three values and returns the hiring decision as a string
- The following Python capabilities will help with this task:
  - The `+=` operator can be used to increment a variable by some amount

```
amount += 5  
amount = amount + 5    # Exact same as the line above
```

- `-=`, `*=` and `/=` also exist and perform similar operations
- A variable can be used to maintain a running total
- An `if` statement can contain `elif` clauses without a final `else` clause

## Example: Hiring decision

```
def decision(gpa, interview, test):  
    points = 0                      # Variable to store the total points  
  
    if gpa >= 3.3:  
        points += 1  
  
    if interview >= 9:  
        points += 2  
    elif interview >= 7:  
        points += 1                  # note: no else clause  
  
    if test > 85:  
        points = points + 1  
  
    if points <= 2:  
        return 'Not hired'  
    elif points == 3:  
        return 'Junior Salesperson'  
    else:  
        return 'Manager-in-Training'
```

# Range & relational operators

The relational operators can be used to express ranges of values

## Examples

- An age in the range 1 through 25, inclusive:

```
1 <= age <= 25
```

- A length in the range 15 (inclusive) up to, but not including, 27:

```
15 <= length < 25
```

- A year in the range 1900 through 1972, exclusive of both:

```
1900 <= year < 1972
```

# **- More on strings -**

# Range & relational operators

Single or double quotes can be used to declare a string

```
'Stony Brook' == "Stony Brook"
```

Strings can be concatenated with a “+”

```
'Stony' + "Brook"
```

The multiplication (asterisk) repeat a string a specified number of times

```
'Hello'*3 → 'HelloHelloHello'
```

# String functions

Strings are very **fundamental to programming**

- Most languages (including Python) support many functions and other operations for strings.

The Python function named **len** (short for “length”) counts the number of characters in a string

- **len** counts every character in a string, including digits, spaces, and punctuation marks
- Example:

```
school = 'Stony Brook University'  
n = len(school)                      # n will equal 22
```

# String Methods

Many other functions on strings are called using  
a different syntax

- Instead of writing **func(s)** they are written **s.func()**
  - The name of the string is written first, followed by a period, and then the function name
- Functions called using this syntax are referred to as **Methods**

Function	vs.	Method
can have many parameters		the object is one of its parameters
exists on its own		belongs to a certain class
function()		object.method()

# String Methods

- As an example of a string method, consider how to figure out how many words are in a sentence
- If there is exactly one space between each word, just count the number of space characters and add one
- The method named `count` does exactly that:

```
sentence = 'It was a dark and stormy night.'  
sentence.count(' ') + 1      # equals 7
```

- Note the argument passed to `count` is a string containing exactly one character: a single space character.

# String Methods

Two other useful methods are **startswith** and **endswith**

- These are both Boolean functions and return **True** or **False** depending on whether a string begins or ends with a specified value

Examples:

- `sentence = 'It was a dark and stormy night.'`
- `sentence.startswith('It')` # True
- `sentence.startswith('it')` # False
- `sentence.startswith("It's")` # False
- `sentence.endswith('?')` # False
- `sentence.endswith('.')` # True

# String Methods

## Another example

```
filename = input('Enter a filename: ')
if filename.endswith('.py'):
    print('The file contains a Python program.')
else:
    print('The file does not contain a Python program.') 
```

A blurred photograph of a library or bookstore interior. The background is filled with tall, dark wooden bookshelves packed with books. In the foreground, there are several study carrels made of dark wood, each containing a desk, a chair, and a lamp. The overall atmosphere is quiet and scholarly.

**- Libraries -**

# Library

Did you know that Python can do much more, thanks to all its **libraries**?

## *Definition*

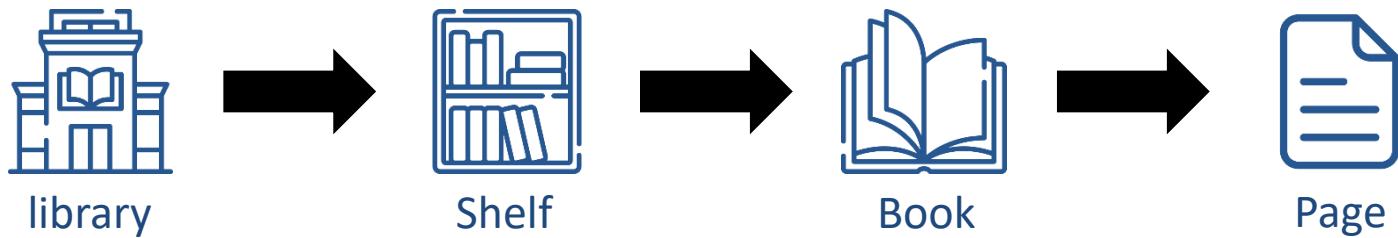
In Python, a **library** is a collection of pre-written code modules that can be imported and used in your own programs to extend their functionality. It can save time and effort by providing commonly used functionality so that you don't have to write everything from scratch.

A library can be created by yourself or other people and can be imported in any of your python project. It is intended to improve **reusability**

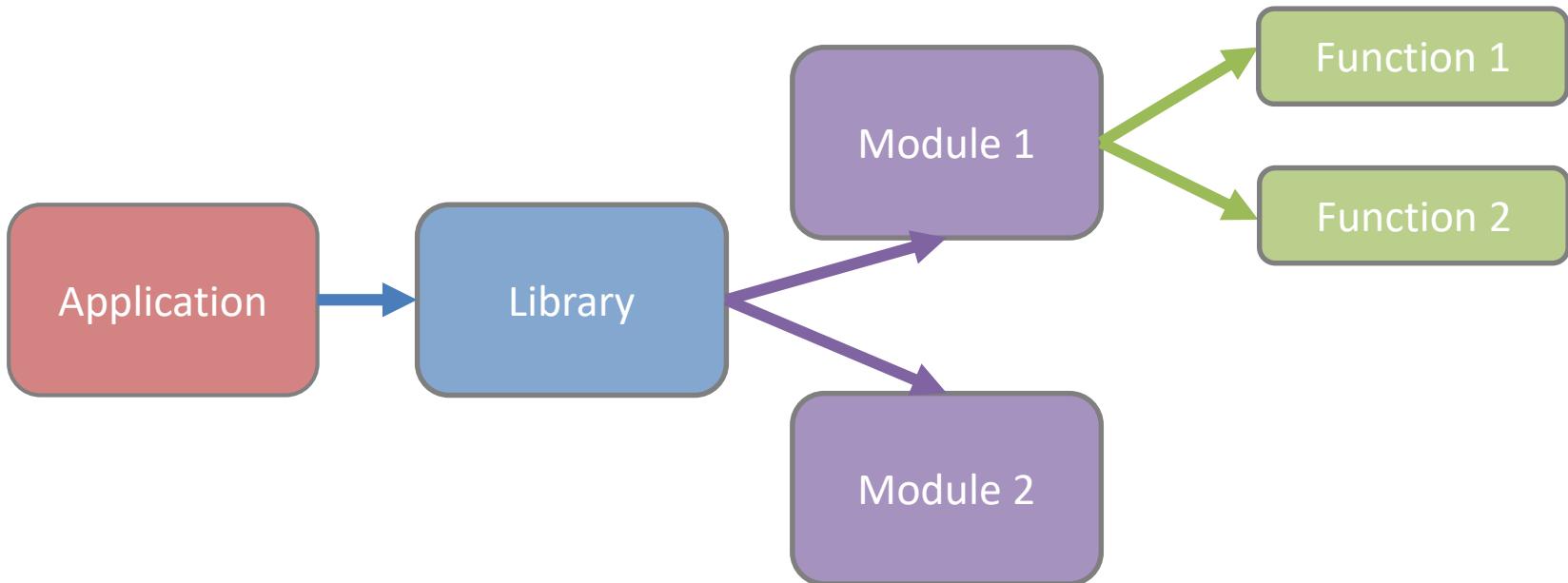


# Library organization

Why do we call it a library?



Similar organization for coding library



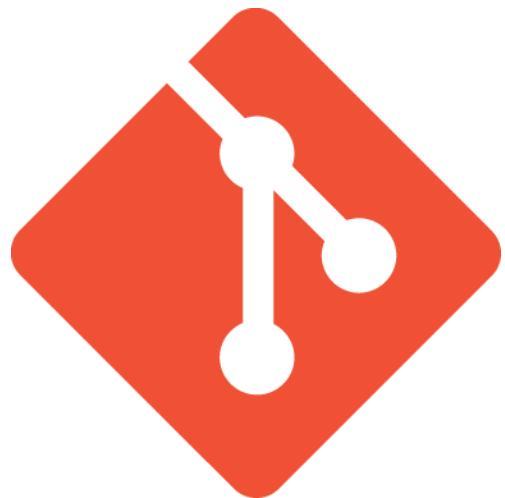
Please note it is a simplification not mentioning classes etc.



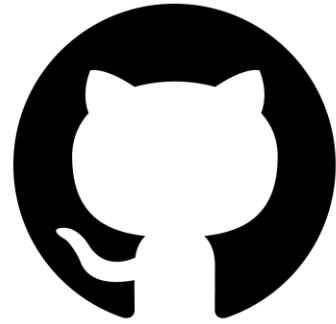
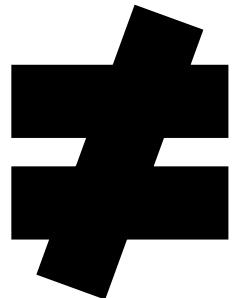
# What is Github?



# What is Git? What is Github?



Git



GitHub

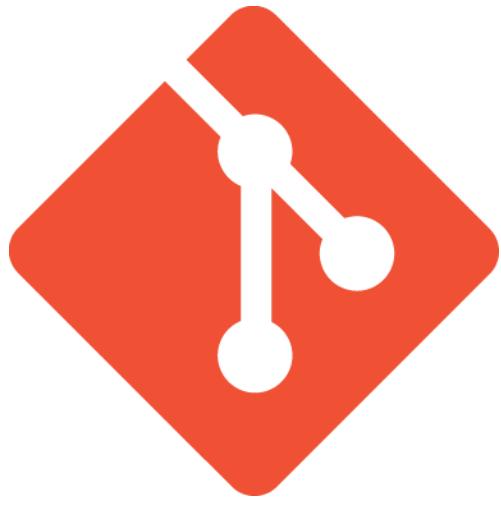
# What is Git?



Git is a distributed version control system developed by [Linus Torvalds](#) in 2005



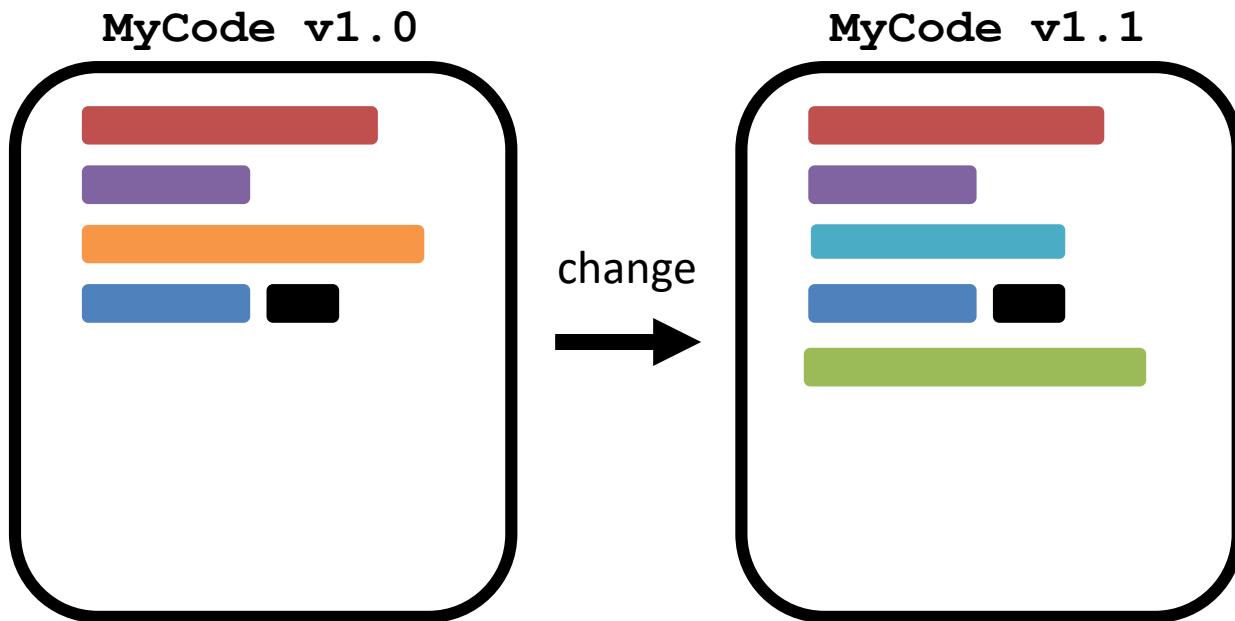
# What is Git?



## Git in a nutshell

- Git is a tool used by developers to keep track of changes made to code. It's like a time machine for code that allows you to go back and see how the code looked at different points in time.
- Allows for collaborative development
- Allows to see who made a change and when

# What is Git?



What if your code v1.1 contains mistakes?

# What is Git?



*Developer 1*



*Developer 2*

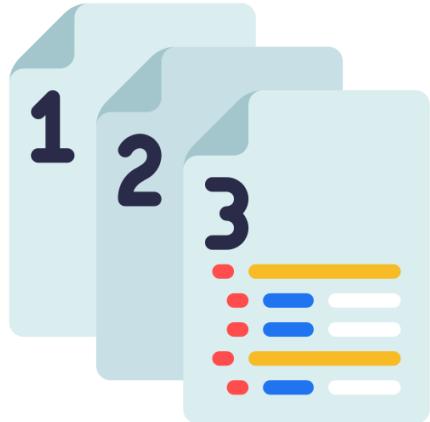


⋮  
⋮  
⋮

Without a version control system would be a  
logistical nightmare

# Why git?

Version control systems resolve two problems



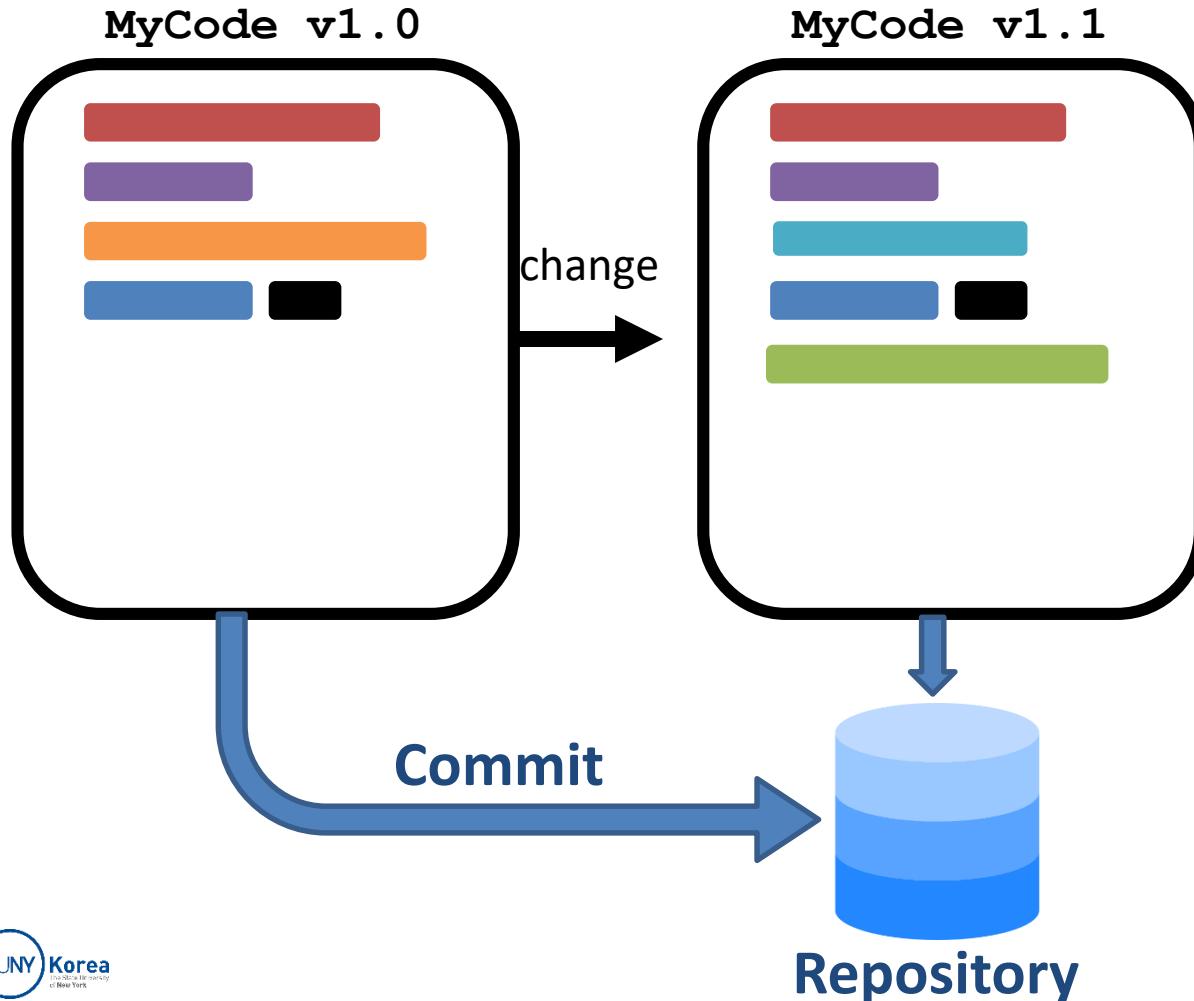
Track code history



Work together

# Git commit

The different versions of the code are saved in a repository

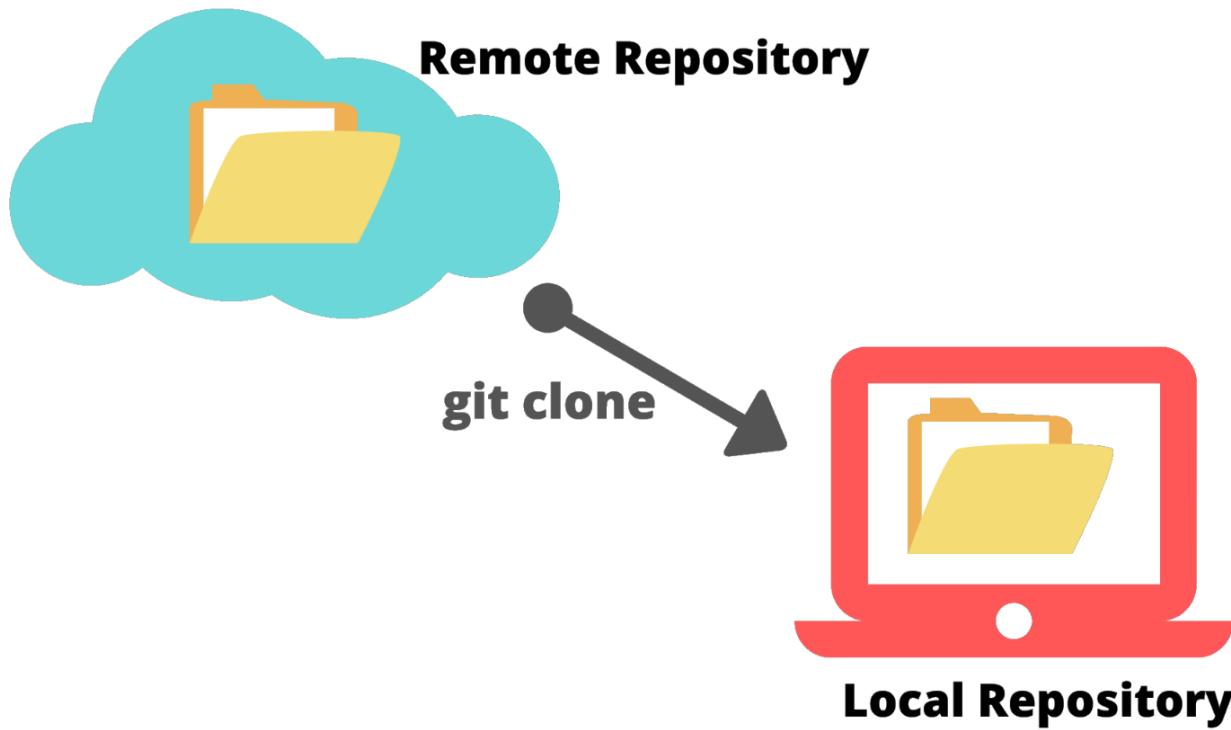


- You can record how your files look at a given point in time: a **snapshot**
- You decide when to take a snapshot and of what files
- You can visit the repository and go back in time as much as you want
- The act of taking a snapshot is called a **commit**
- All your snapshots are saved in a **repository**
- A project is made of lots of commits

# Git cloning

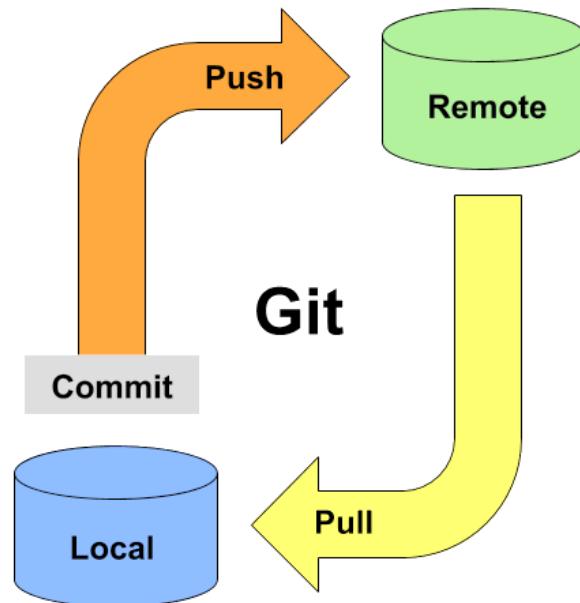
Copying a repository from a remote server is called **cloning**.

This is an essential function to work together



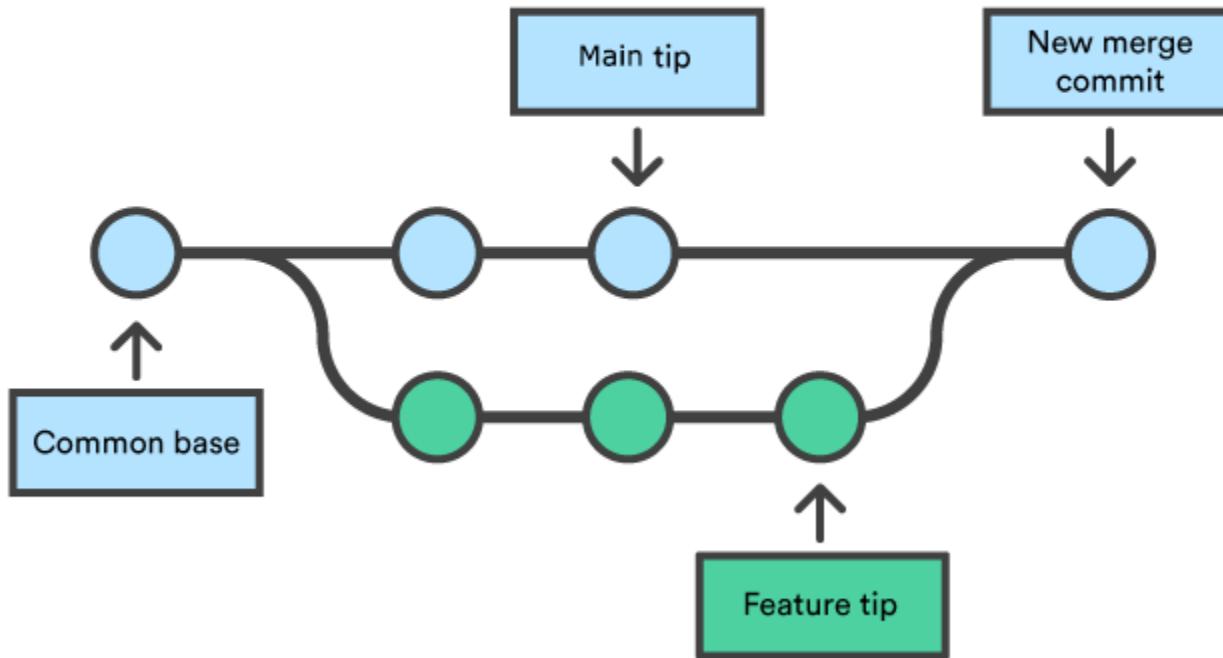
# Git push and pull

- **Git pushing:** Uploading local changes to a remote repository.
- **Git pulling:** Downloading changes from a remote repository to a local copy of the repository.



# Git branches

- All commits on git live on branches
- There can be many many branches
- The main branch is called **master branch**
- A new branch can be **merged** to the master



# Install git

## Windows

1. Download the Git installer for Windows from the Git website (<https://git-scm.com/download/win>).
2. Run the installer and follow the on-screen instructions.
3. When prompted, choose the default settings unless you have a reason to choose otherwise.
4. Complete the installation process.

## Ubuntu

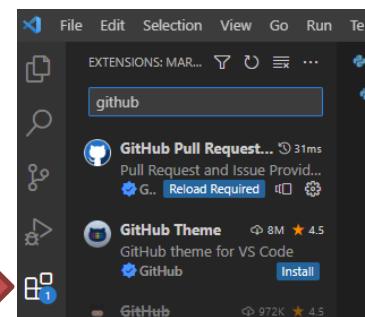
1. Open the Terminal application.
2. Run the following command to update the package list: `sudo apt-get update`.
3. Run the following command to install Git: `sudo apt-get install git`.



## Mac

1. Download the Git installer for Mac from the Git website (<https://git-scm.com/download/mac>).
2. Double-click on the downloaded file to launch the installer.
3. Follow the on-screen instructions.
4. When prompted, choose the default settings unless you have a reason to choose otherwise.
5. Complete the installation process.

## VSCode



### Install:

- Github pull request
- Github classroom

# What is GitHub?

GitHub, on the other hand, is a web-based platform that provides hosting for Git repositories.

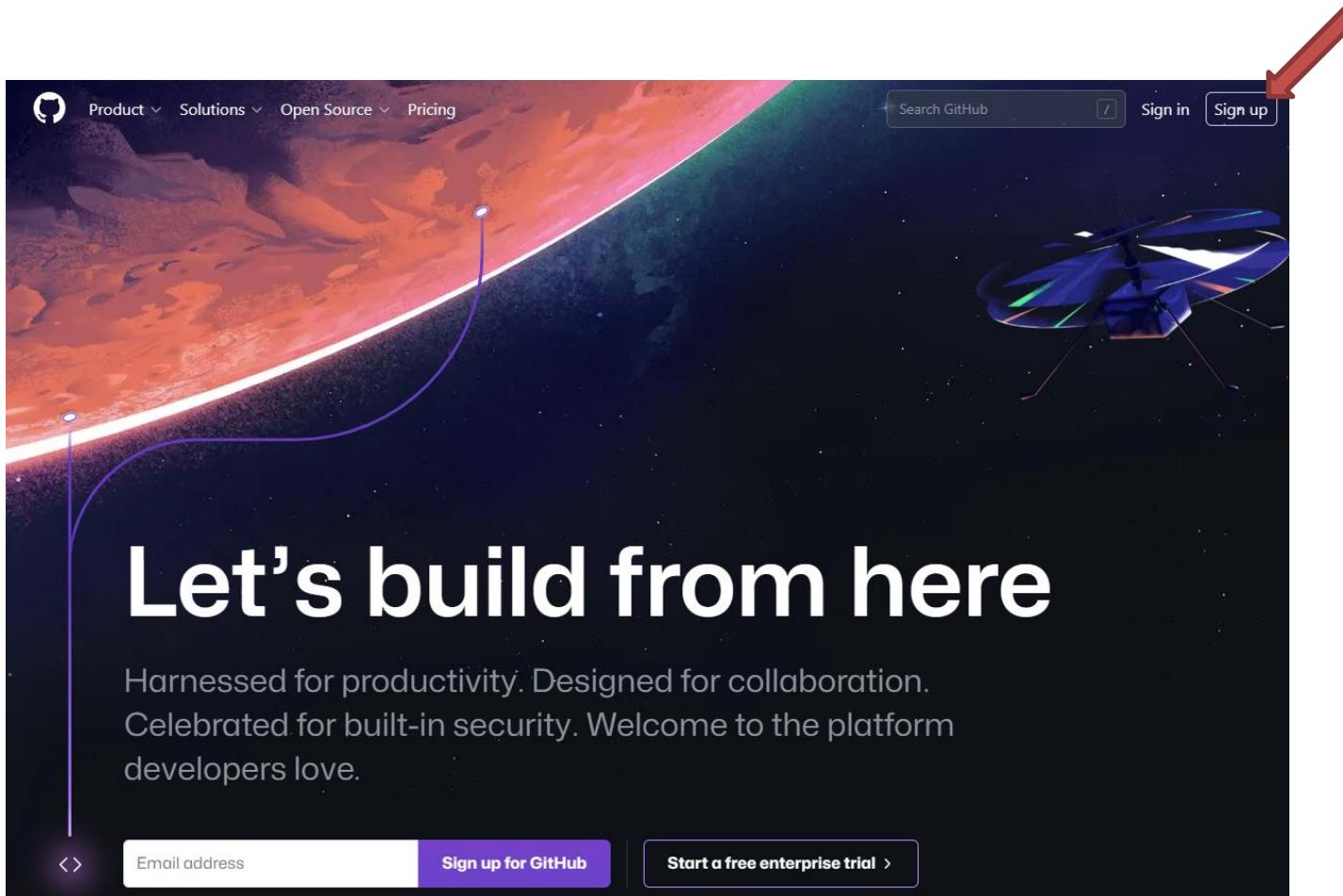
- Largest web-based git repository hosting service
- Allow for code collaboration with anyone online!
- Add extra functionality on the top of git (UI, Documentation, Bug tracking etc.)



# Create a GitHub account

Go to:

<https://github.com/>



# Why GitHub in CSE101?

- To get you familiar with good development practice
- To keep your code available forever
- To ease the marking and homework submission
- GitHub can be used for much more than just programming:
  - Create your website in a few clicks
  - Any collaborative projects
  - Design
  - Project management
  - And much more

# GitHub Classroom



## **Lab 1 - Arithmetic in python**

# GitHub Classroom

On visual studio it is simple to start!

The screenshot shows the GitHub Classroom interface. At the top, there's a navigation bar with the GitHub Classroom logo, GitHub Education, and other icons. Below the bar, a large circular icon with a graduation cap and hands is displayed, accompanied by the text "You're ready to go!". A message below states "You accepted the assignment, Lab 1 - Arithmetic in python." and "Your assignment repository has been created:". A blue button with a link to the repository (<https://github.com/SUNYKorea/lab-1---arithmetic-in-python-rameau-fr>) is shown. A note says "We've configured the repository associated with this assignment ([update](#)).". At the bottom left, a red arrow points to a "Open in Visual Studio Code" button. On the right, a sidebar for the GitHub Student Developer Pack is visible, featuring a backpack icon, the text "Join the GitHub Student Developer Pack", a description about free GitHub Pro and tools for verified students, and a blue "Apply" button.

Note: You may receive an email invitation to join [SUNYKorea](#) on your behalf. No further action is necessary.

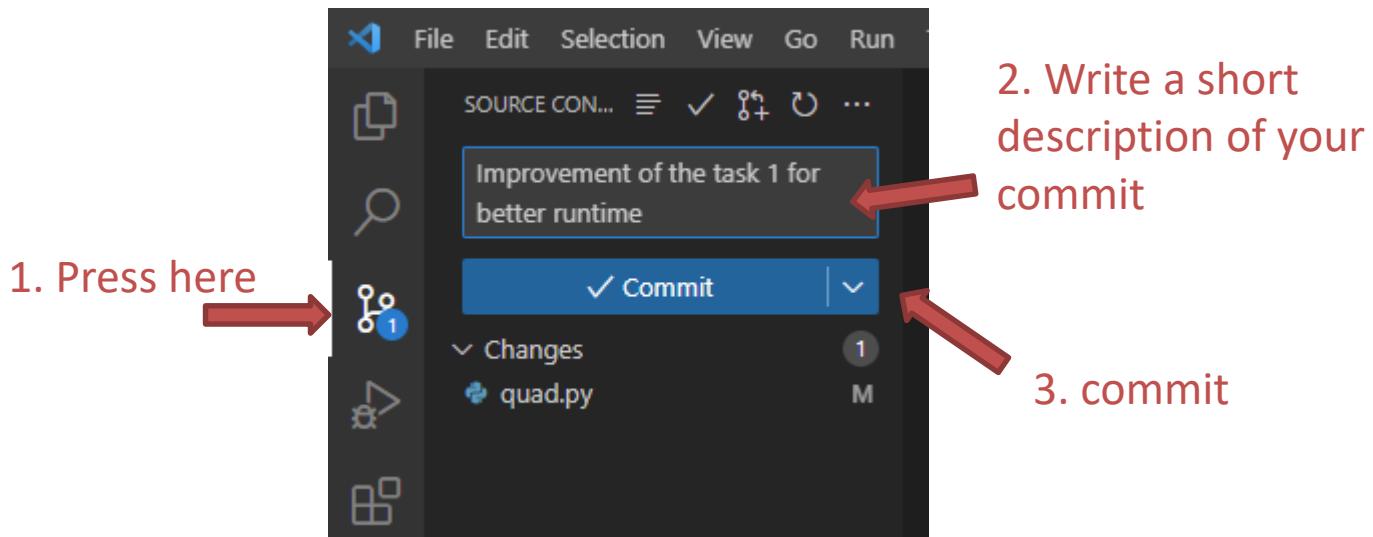
# GitHub Classroom

If it does not work

1. In VSCode press F1
2. Write git clone
3. Press on the first button
4. Type the link to the repo
5. Choice the folder where you want to work
6. Code

# GitHub Classroom

Push your code to send your assignment



# GitHub Classroom

If you do not use VSCode

1. Create a folder where you want to work
2. Run a terminal in this folder
3. Type `git clone link_to_the_repo`
4. Code
5. `git add file_you_want.py`
6. `git commit -m "short description"`
7. `git push -u origin main`

**! alert !**

You might need to set an ssh key on your github account. Please request assistance during the lab!

