CSE 101

Computer Science Principle

Lecture 03: Variables & Functions

March. 2023
Prof. Francois Rameau
francois.rameau@sunykorea.ac.kr

SUNY Korea
The State University of New York

Attendance time!

Scan the QR code and select your name in the list

What are we studying today?

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

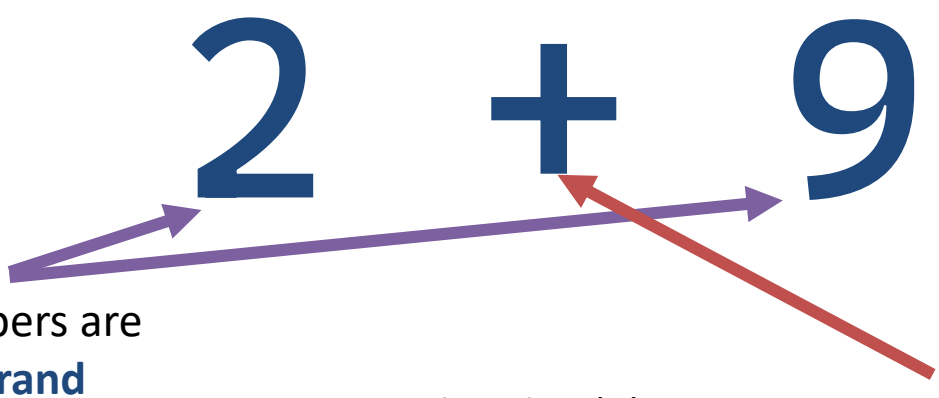
Numbers

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



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	Remainder
/	//	%
	(divides and rounds down to the nearest whole number)	(gives the remainder of an integer division, also called "**modulo**")

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

Exponential
**

Arithmetic in python

Some examples

Add	Subtract	Multiply
11 + 5 → 16	11 - 5 → 6	11 * 5 → 55

Divide	Integer divide	Remainder
11 / 5 → 2.2	11 // 5 → 2	11 % 5 → 1

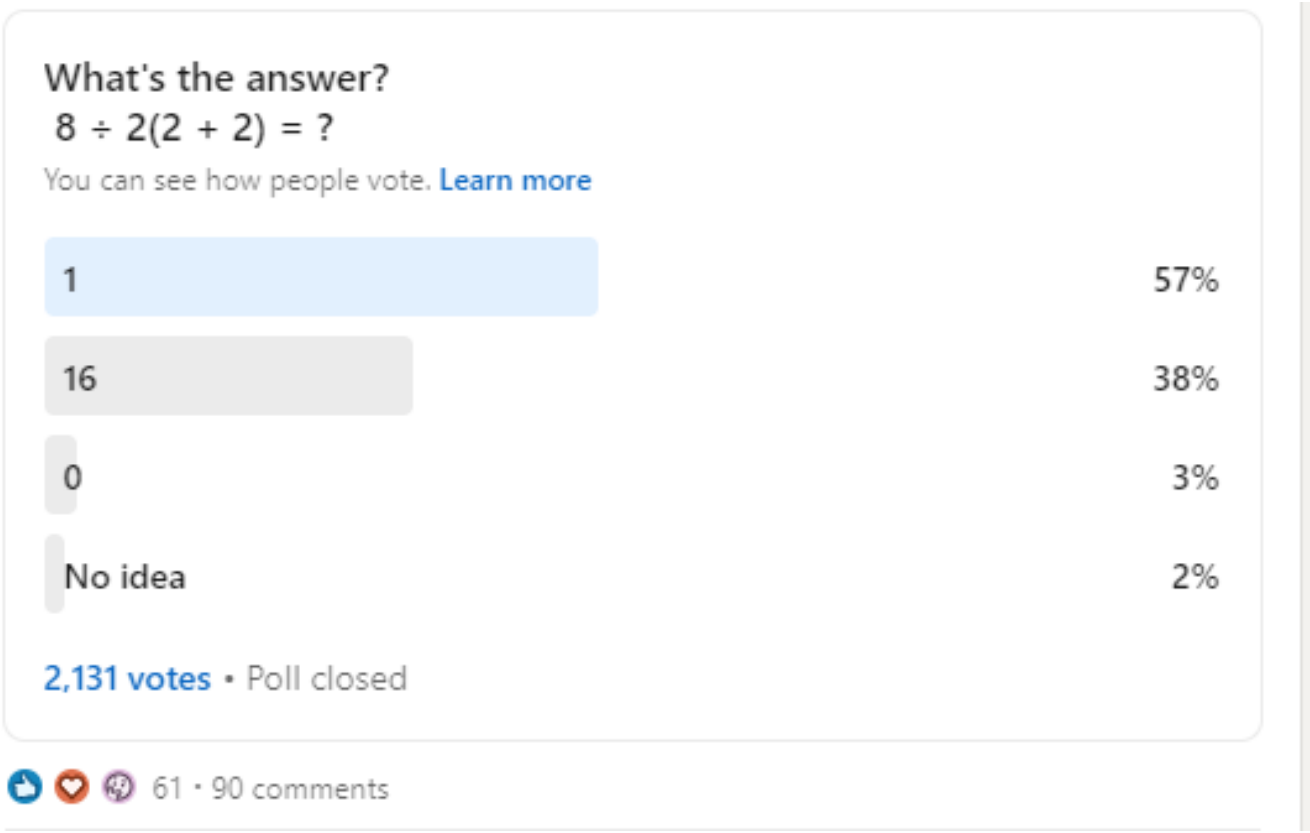
(This example shows integer division. Any remainder is discarded)

Exponential
**

- 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

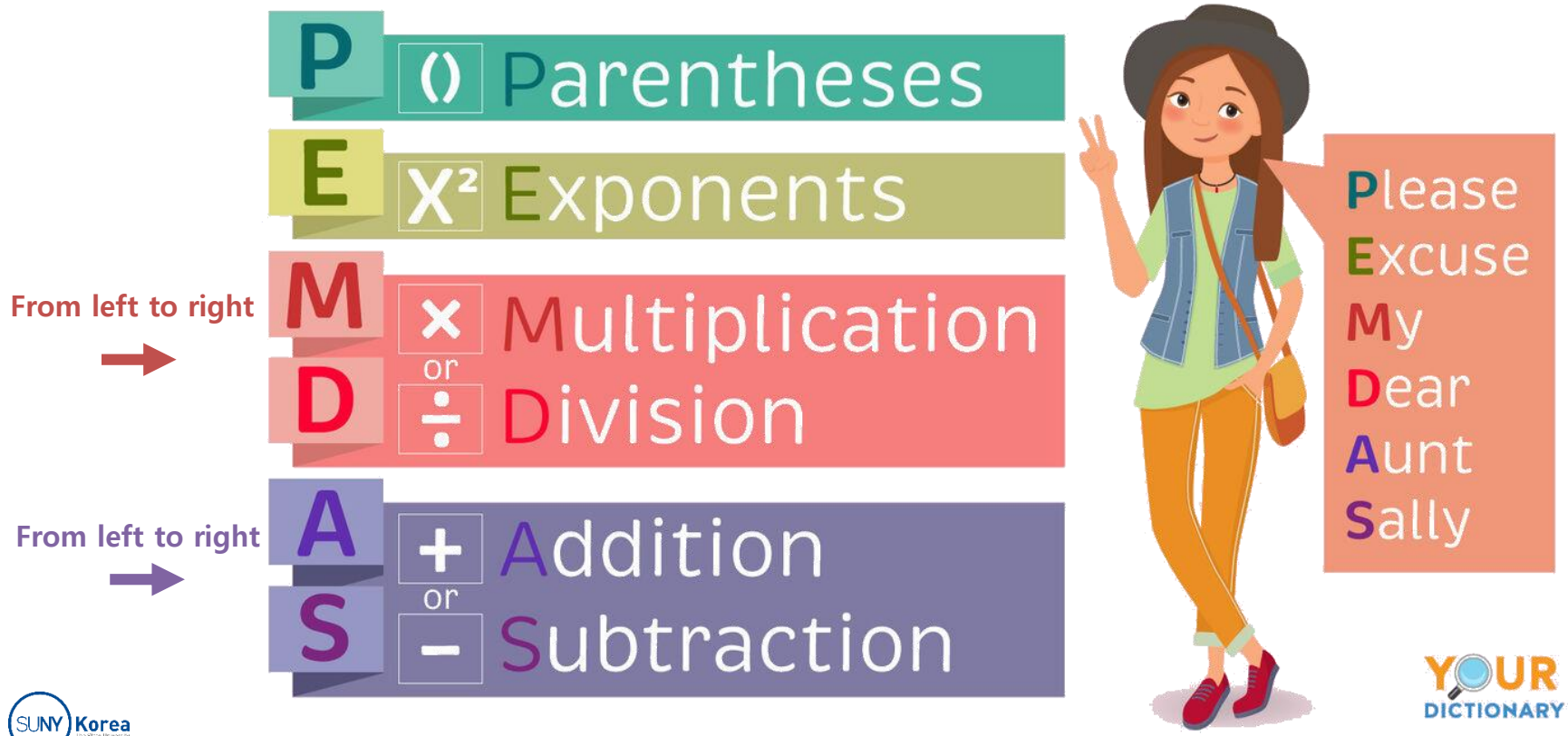
What happen when you have complex operations?



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 ½ is the same as taking a square root

So `16 ** 0.5` would be the same as √16 which is 4

Arithmetic in python

Operators on INT and FLOAT

```
a + b → sum
a - b → Difference
a * b → Product
```

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

```
a / b → Division
```

- 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?

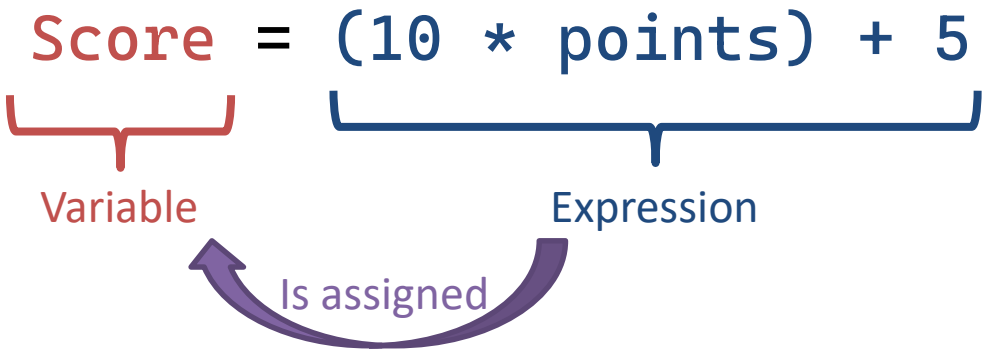
```
billTotal = 10.50
```

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	integer	string
`length = 3.5`	`total = 2 + 5`	`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);
}
```

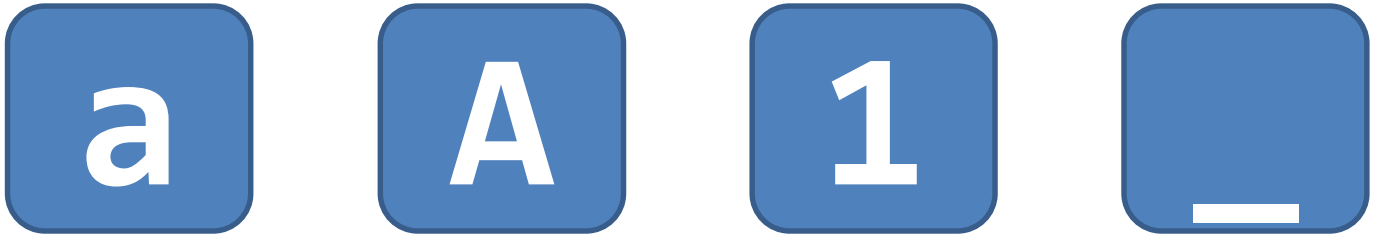
What about now?

```
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);
}
```

Source: CodeAesthetic – Naming things

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!

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

A few examples:

```
if
```

```
for
```

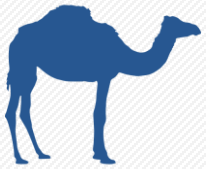
```
and
```

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
Function	Use a lowercase word or words. Separate words by underscores to improve readability.	`function`, `my_function`
Variable	Use a lowercase single letter, word, or words. Separate words with underscores to improve readability.	`x`, `var`, `my_variable`
Class	Start each word with a capital letter. Do not separate words with underscores. This style is called camel case or pascal case.	`Model`, `MyClass`
Method	Use a lowercase word or words. Separate words with underscores to improve readability.	`class_method`, `method`

Type	Naming Convention	Examples
Constant	Use an uppercase single letter, word, or words. Separate words with underscores to improve readability.	`CONSTANT`, `MY_CONSTANT`, `MY_LONG_CONSTANT`
Module	Use a short, lowercase word or words. Separate words with underscores to improve readability.	`module.py`, `my_module.py`
Package	Use a short, lowercase word or words. Do not separate words with underscores.	`package`, `mypackage`

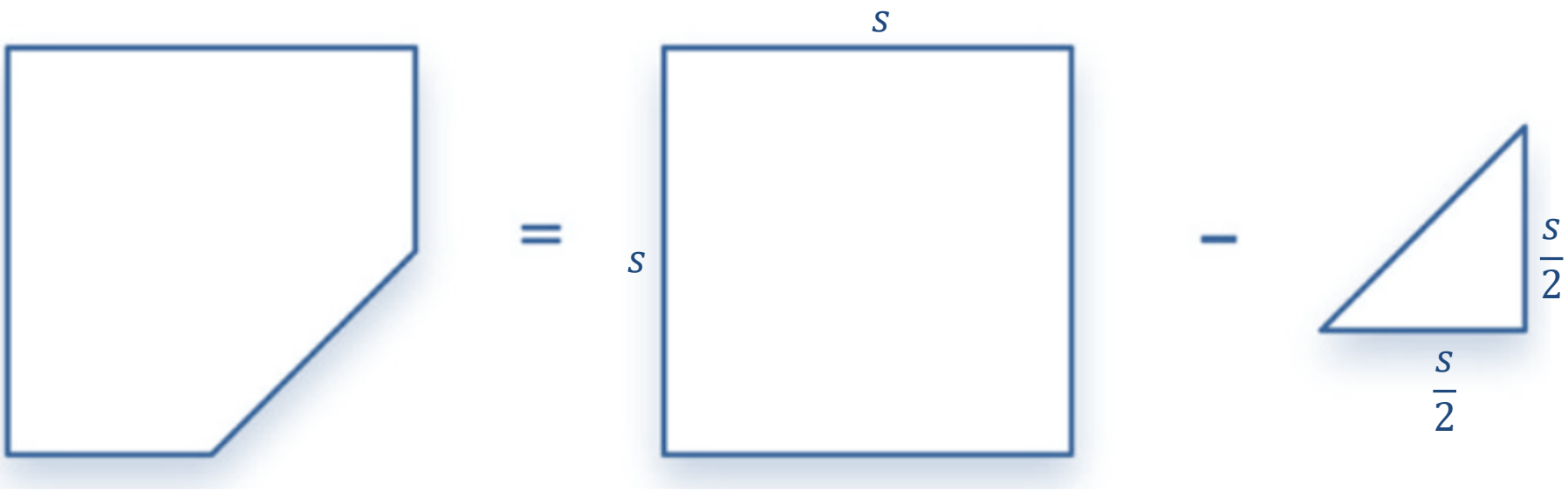
Variables & Arithmetic

- 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

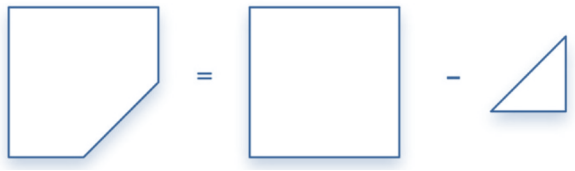
Example: Area calculation

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 = (- b+(b**2-4*a*c)**(1/2)) / (2*a)
x_2 = (- b-(b**2-4*a*c)**(1/2)) / (2*a)

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 o f data can be printed

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

Example: Area calculation (print)

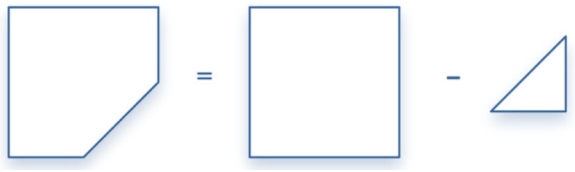
- To print a number, it must first be converted into a string, like so:

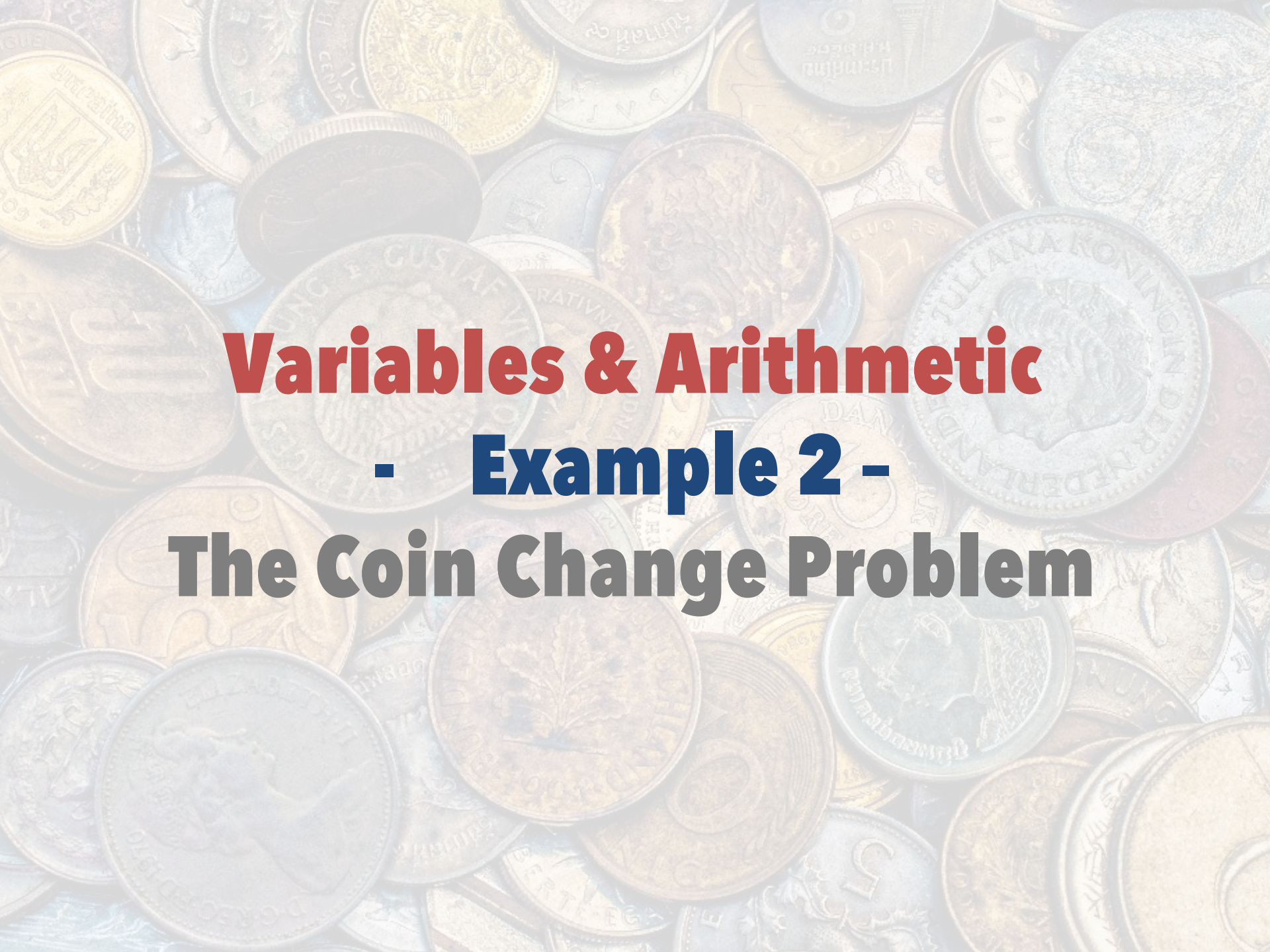
```
print("The area is " + str(area))
```

- 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))
```





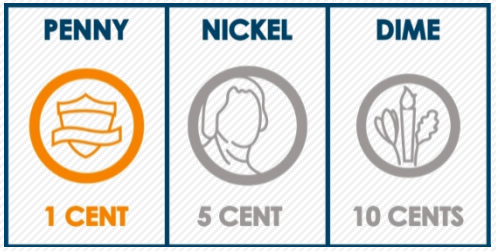
Variables & Arithmetic
- Example 2 –
The Coin Change Problem

The coin change problem

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

Problem

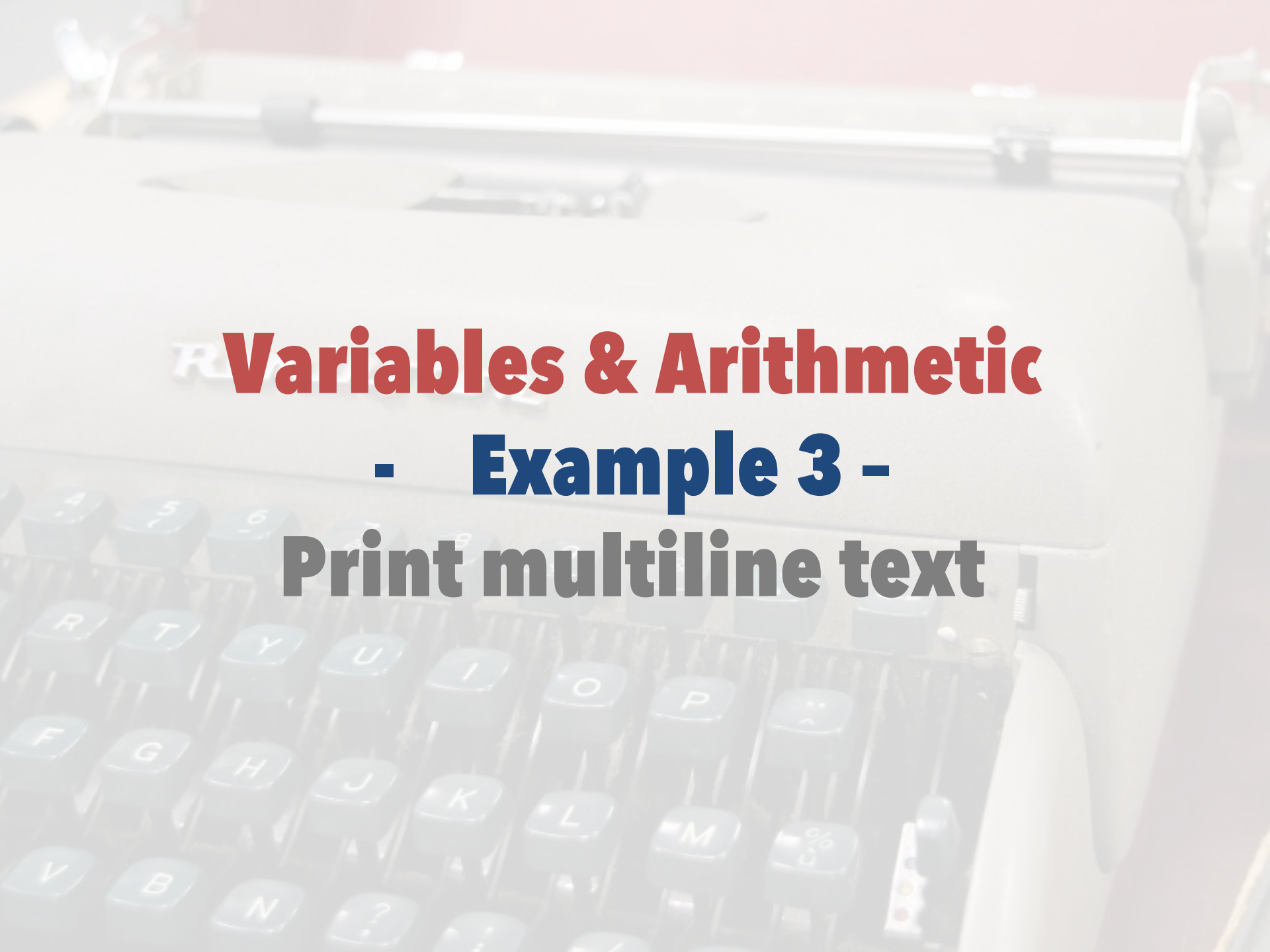
Given a total number of cents, the computer should print how many dimes, nickels, and pennies are needed to make that change while minimizing the number of coins



- 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 π

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

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

Functions

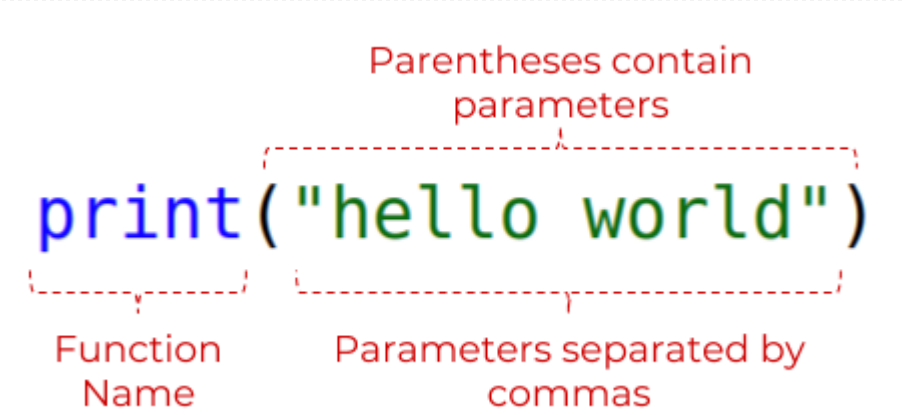
Python's math module contains code related to mathematical functions

Constants	Functions
$\pi, e,$ etc.	$\cos x, log_2$ etc.

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

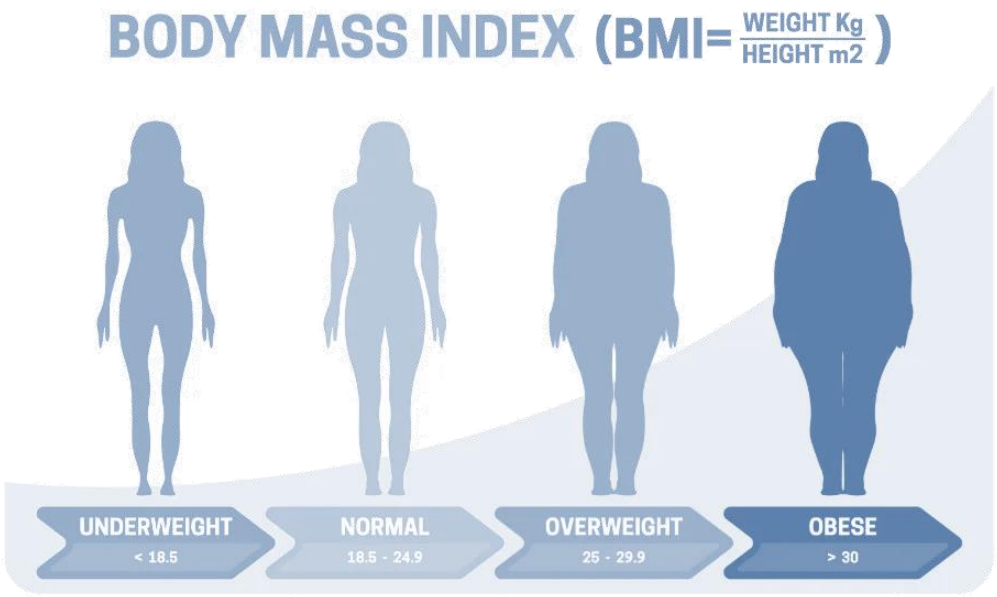
Example

the **input** function pe [redacted] urns the value entered by the user



print() is also a functi [redacted] iple arguments!

Functions: Example BMI



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)
...
```

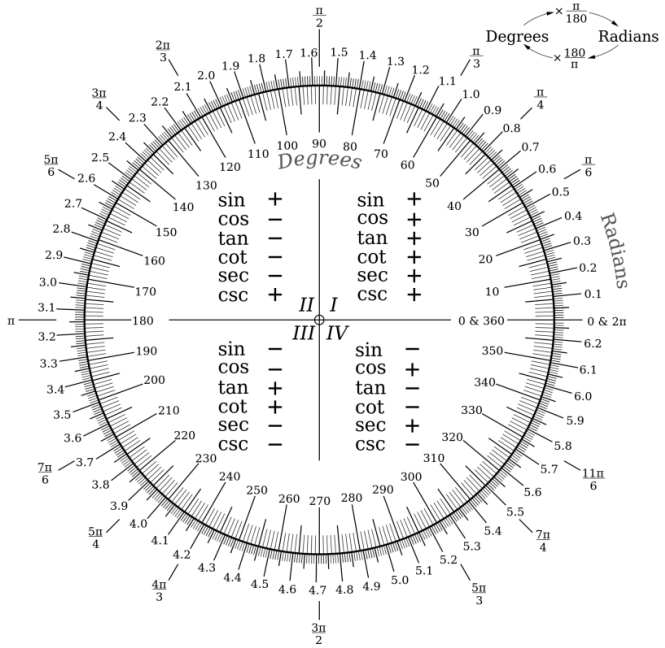
```
import random
random.random()
random.randint(0,100)
```

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.

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

3. Code organization:
Functions group related pieces of functionality together, making it easier to find and modify code.

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

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 math: 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

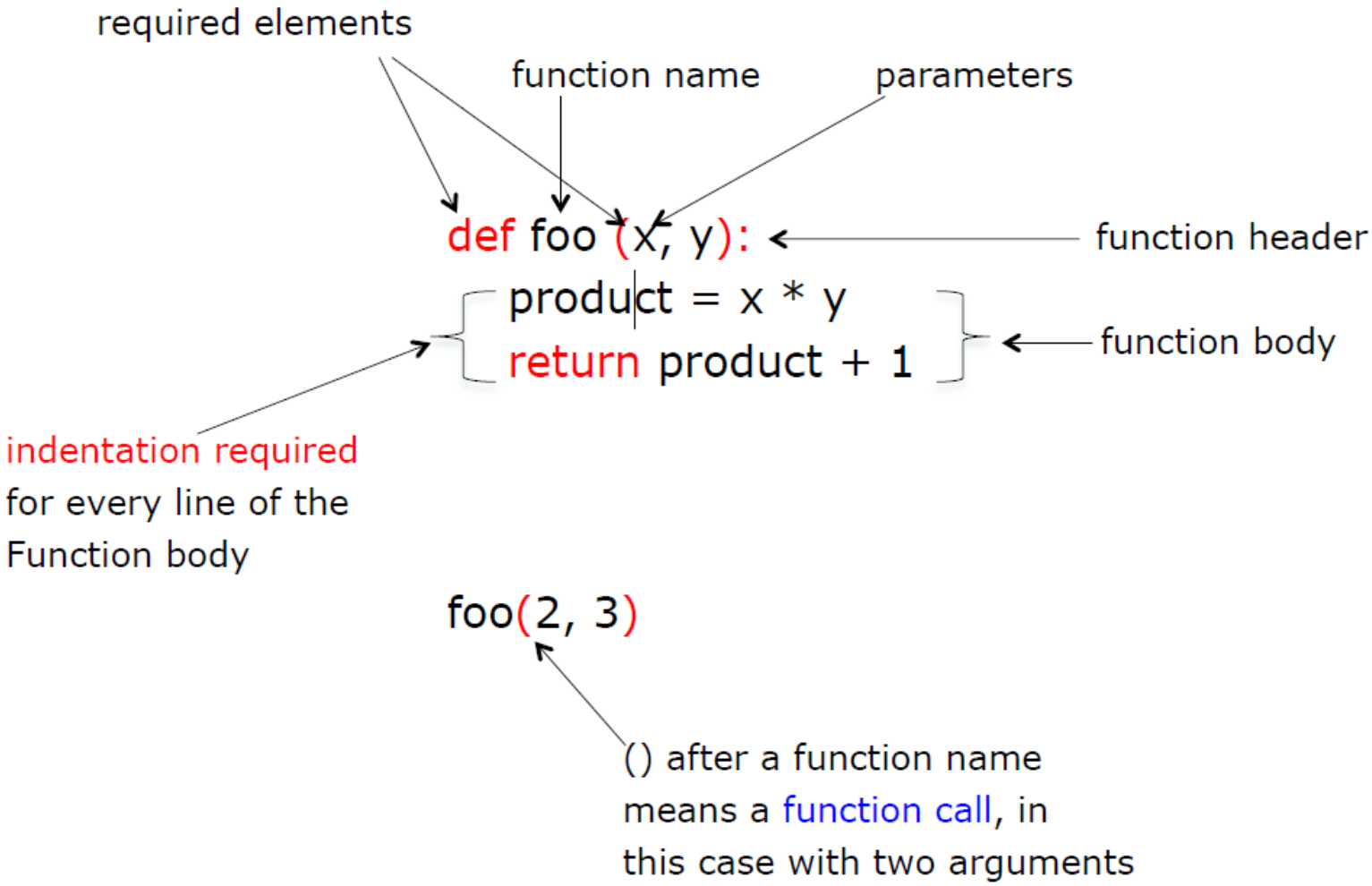
```
def f(x, y):
    return x * y + 1
```

indentation

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

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

Defining/calling a function



A word about indentation

Indentation matters a lot in Python

They are used to denote blocks of code!

```
def print_hello():
    print("Hello,")
    print("how are you?")
    print("I hope you're doing well.")

def print_world():
    print("World!")
    print("What's up?")
    print("Everything good?")

print_hello()
print_world()
```

How do we do indentation??

Officially, 4 spaces are recommended to do the indentation. But using Tab tends to be less confusing

You need to be consistent!! If you use both spaces and Tab … Your code will not work anymore

Parameters and arguments

A function can have zero or more parameters

Functions may be defined with *formal parameters*

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

```
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))
```

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

Function

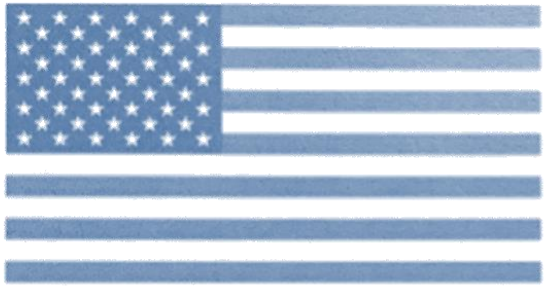
- Example 1 –

Distance calculator

Example: distance calculator

Problem:

A distance traveled is provided in miles, yards, and feet (e.g. 3 miles, 68 yards, 1 6 feet). We need this to be converted to total inches traveled and print the resul. 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

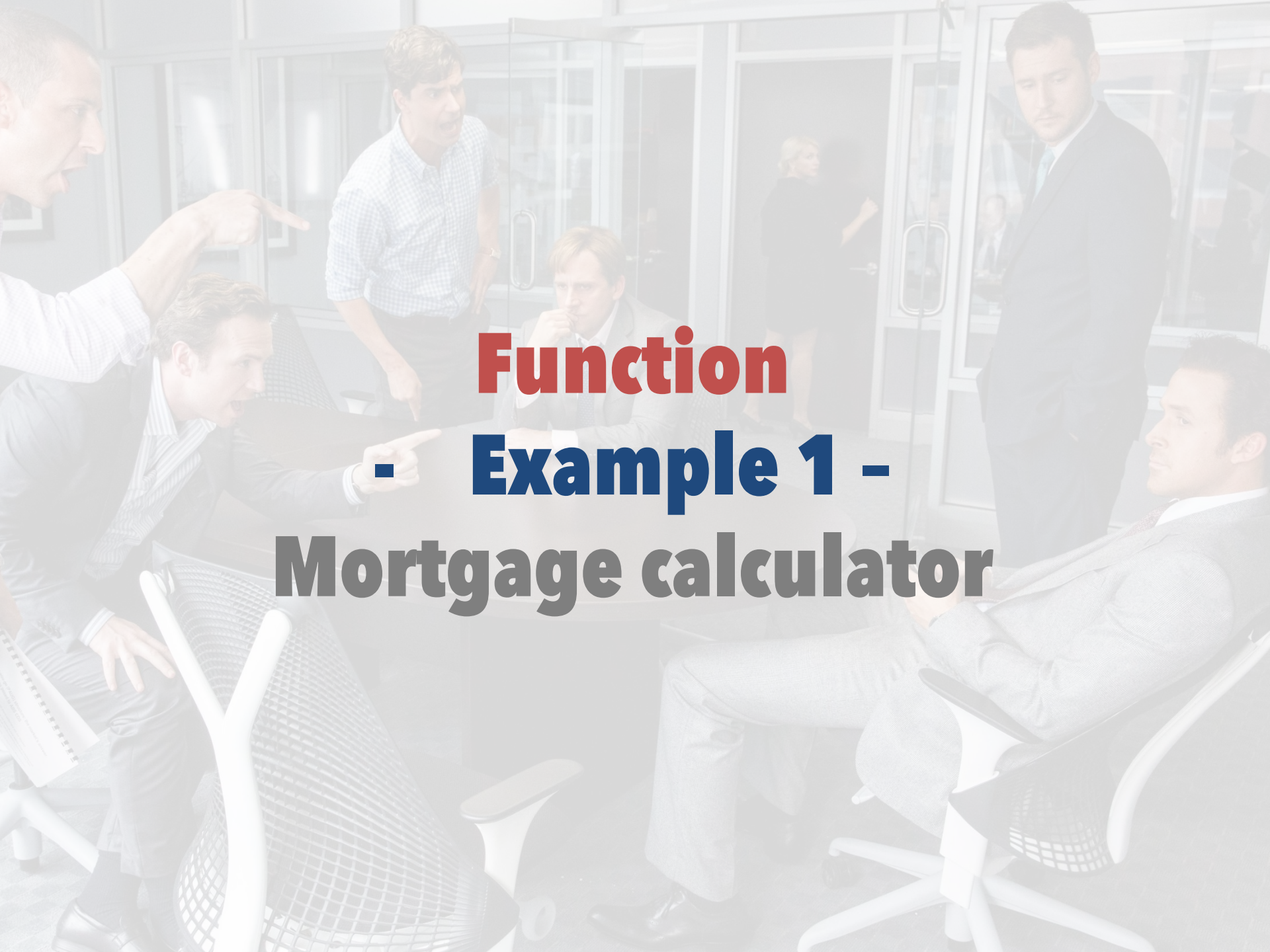
1 m = 100 cm = 1000 mm

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:

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

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 \
          $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" + \
             "\ 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 >= means "greater than or equal to" and is one of six **relational operators** supported by Python:

Mathematical Operator	Python Equivalent	Meaning
=	==	is equal to
≠	!=	is not equal to
>	>	is greater than
≥	>=	is greater than or equal to
<	<	is less than
≤	<=	is less than or equal to

Be careful: `= and == are very different!! A single = is used for assignment while == is used for comparison!`

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:

$$hourly\ wage \times hours\ worked$$

The overtime formula has two parts:

- The pay for the first 40 hours
- The pay for additional overtime hours (1.5 x 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 candidates 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
```

Logical connective

- What if you want to test multiple conditions at the same time?
  - This is when you need logical connectives: **and**, **or, not**

COND1 and COND2 # true when both COND1 and COND2 are true

COND1 or COND2 # true when at least one of COND1 and COND2 is true

not COND1 # true when COND1 is false

```
def decision(gpa, interview, test):
        if gpa > 3.3 and interview > 2:
                print("Very good")
        elif gpa > 3.3 and interview <= 2:
                print("Nice")
        else:
                print("Meh")
```

```
def decision2(gpa, interview, test):
        if not (gpa > 3 or interview > 1):
                print("Not good")
        if test < 2 or gpa < 2:
                print("Not good at all")
```

Logic operators on Boolean

Assume a and b are variable names with Boolean values

Not

`not a` → `True` if `a` is `False`, `False` if `a` is `True`

And

`a and b` → `True` if both are `True`

Or

`a or b` → `True` if either or both are `True`

a	b	a and b	a or b
True	True	True	True
True	False	False	True
False	True	False	True
False	False	False	False

Logic operators on Boolean

AND

```
print("Welcome to the voting eligibility checker!")
age = int(input("Enter your age: "))
citizen = input("Are you a US citizen (yes or no)? ").lower()
if age >= 18 and citizen == "yes":
    print("Congratulations! You are eligible to vote in the US election.")
else:
    print("Sorry, you are not eligible to vote in the US election.")
```

OR

```
# Adventure game: choose a path to follow (left or right)
print("Welcome to the adventure game!")
print("You are in a dark forest and you see two paths.")
path = input("Do you want to take the left or right path? ").lower()
if path == "left" or path == "right":
    print("You chose the", path, "path.")
    print("You follow the path and encounter a giant spider!")
else:
    print("Invalid path! Please choose either the left or right path.")
```

- 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

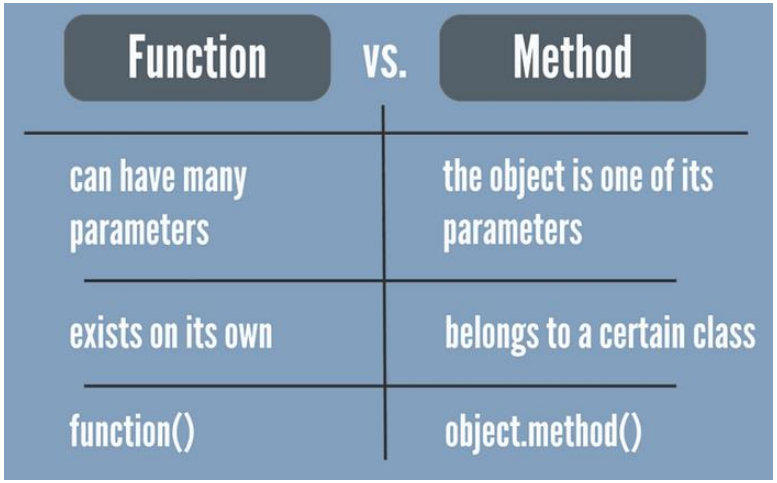
- **`len`** counts every character in a string, including digits, spaces, and punctuation marks
- Example:

```
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**



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.')
```

String Methods

index()	Returns the position (=index) of the first occurrence
isalnum()	Is the string alphanumeric?
isalpha()	Is the string alphabetic?
isdecimal()	Does the string consist of decimals (0 – 9)?
isdigit()	Does the string consist of digits? (including "₩u00B2")
isnumeric()	Does the string consist of numeric characters? (including '²' and '¾')
islower()/isupper()	Are the characters all lower/upper case?
find()	Almost identical to index(), except find() returns -1 for failure
split()	Splits the string by the given delimiter
join()	Joins a list of strings by the given delimiter

Access individual letter

A string is a list of characters; each individual character can be accessed in the following manner:

```
word = "Cat"
print(word[0])
print(word[1])
print(word[2])
```

- Libraries -

Library

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

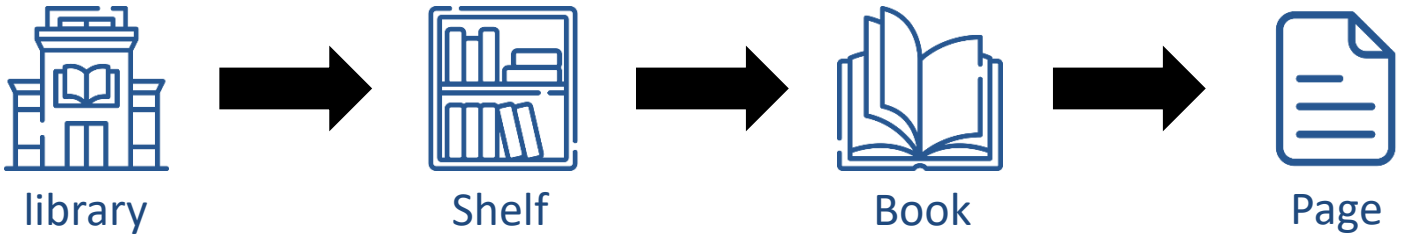
Definition

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.

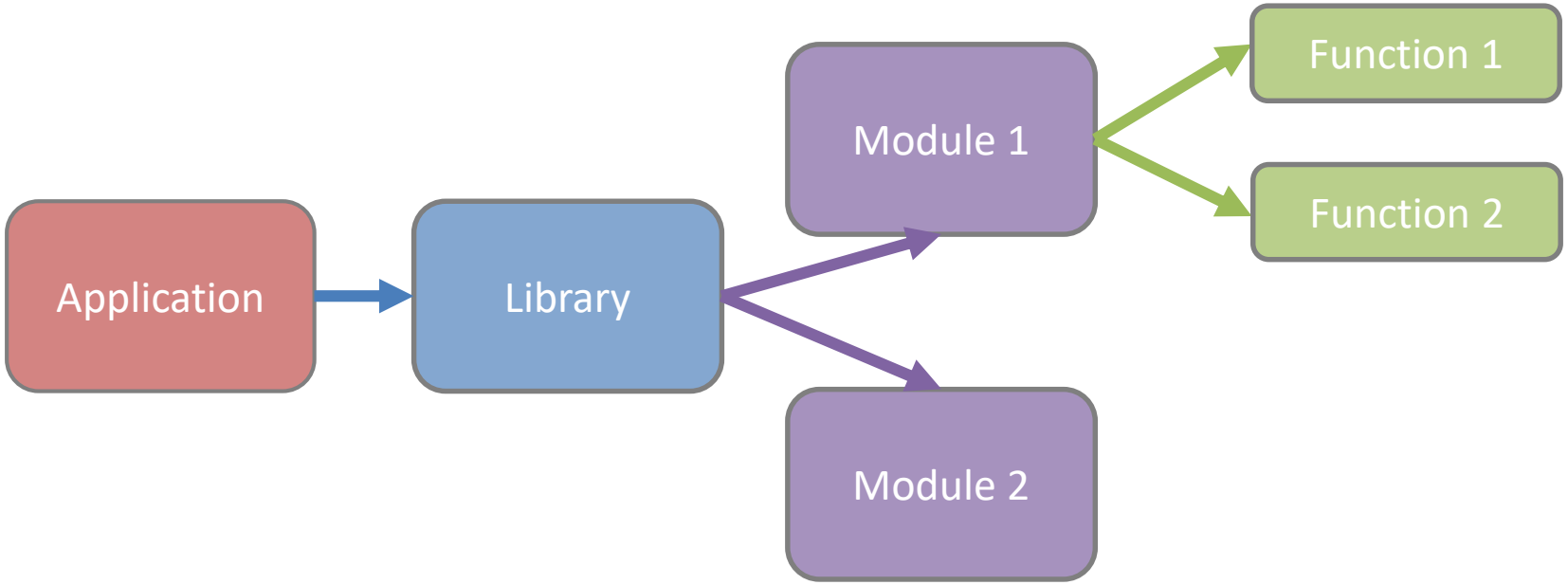
A library can be created by yourself or other people and can be imported in any of your python project. It is intended to improve **reusability**

Library organization

Why do we call it a library?



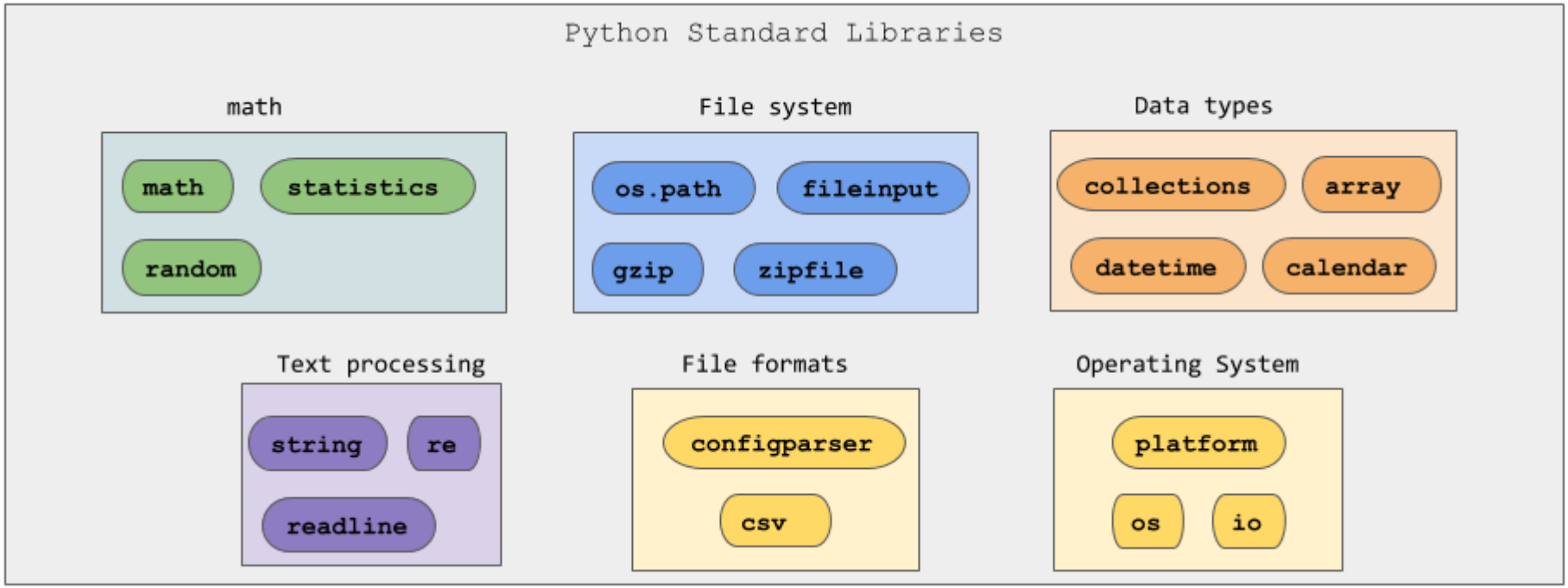
Similar organization for coding library



Please note it is a simplification not mentioning classes etc.

Python standard libraries

Python has a **standard library** composed of multiple modules directly accessible to anyone who installed Python



For instance, we have seen before the use of the math module which contains many useful functions!

Python standard libraries

Unlike `print()`, you do not have access to standard Python modules by default.

You need to explicitly **import** them into your program!

If you need to generate some random numbers, it is likely you will need the module "random". You found a function called `random.randrange(2, 20)` which generates a random int in a given range.

How do we use it?

Let's try to use it right away!

```
my_random_number = random.randrange(2, 20)
```

```
my_random_number = random.randrange(2, 20)
NameError: name 'random' is not defined
```

import

You first need to use the import command!

```
import random
my_random_number = random.randrange(2, 20)
```

- The import command imports the ENTIRE module, and this is convenient but it requires you to use the syntax `random.module_name`

- You can be more specific in the functions you need by using the command `from` (it is loading the function into the local namespace, be careful with conflits)

```
from random import randrange
my_random_number = randrange(2, 20)
```

Packages

One of the main strengths of Python is its large choice of third parties library, also called packages

Packages

Where to get these packages? PyPI

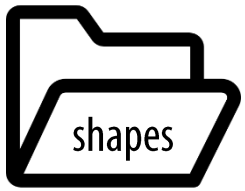
All these verified packages can be very easily installed via the Python packet manager pip

To install the package numpy, you simply need to write in your terminal:

```
pip install numpy
```

Packages

You can also very easily make your own libraries!
Let's say you want to create a library for calculating the area of different geometric shape



__init__.py

```
from .circle import area as circle_area
from .rectangle import area as rectangle_area
```

circle.py

```
import math

def area(radius):
    return math.pi * radius ** 2
```

rectangle.py

```
def area(length, width):
    return length * width
```

Packages

In the root file you can create your main python code

```
import shapes

radius = 5
circle_area = shapes.circle_area(radius)
print(f"The area of a circle with radius {radius} is {circle_area}")

length = 10
width = 5
rectangle_area = shapes.rectangle_area(length, width)
print(f"The area of a rectangle with length {length} and width {width} is {rectangle_area}")
```

Git Tutorial for Beginners: Learn Git in 1 Hour

Git 101: Git and GitHub for Beginners

What is Github?

What is Git? What is Github?

Git

≠

What is Git?

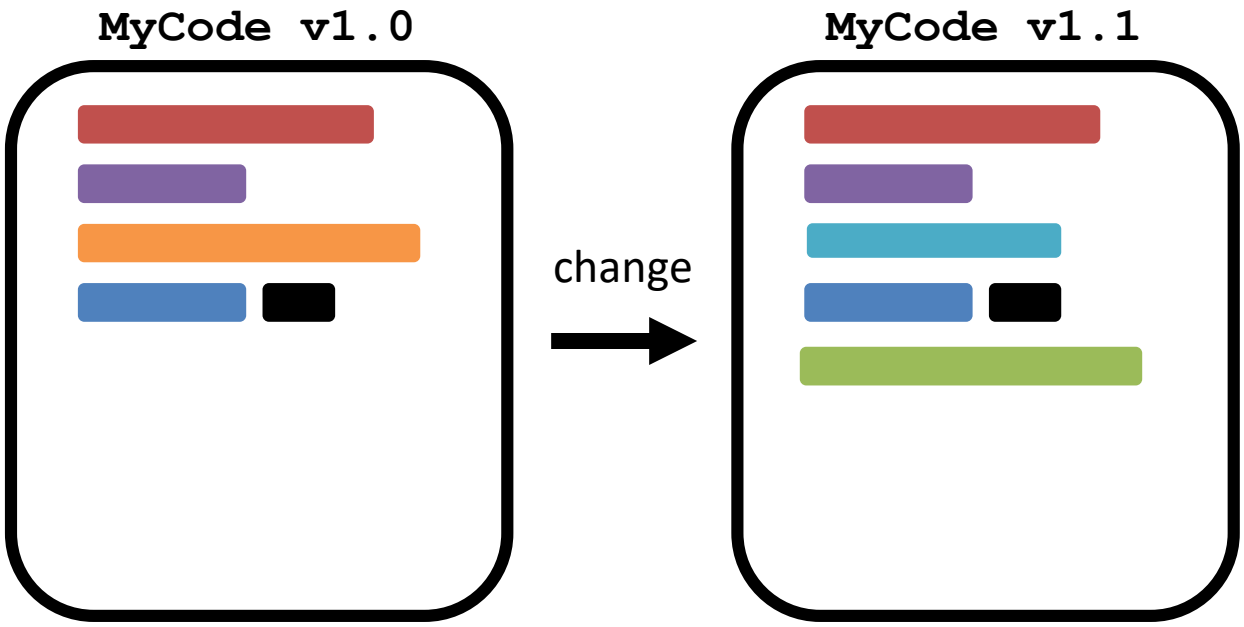
Git is a distributed version control system developed by **Linus Torvalds** in 2005

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?

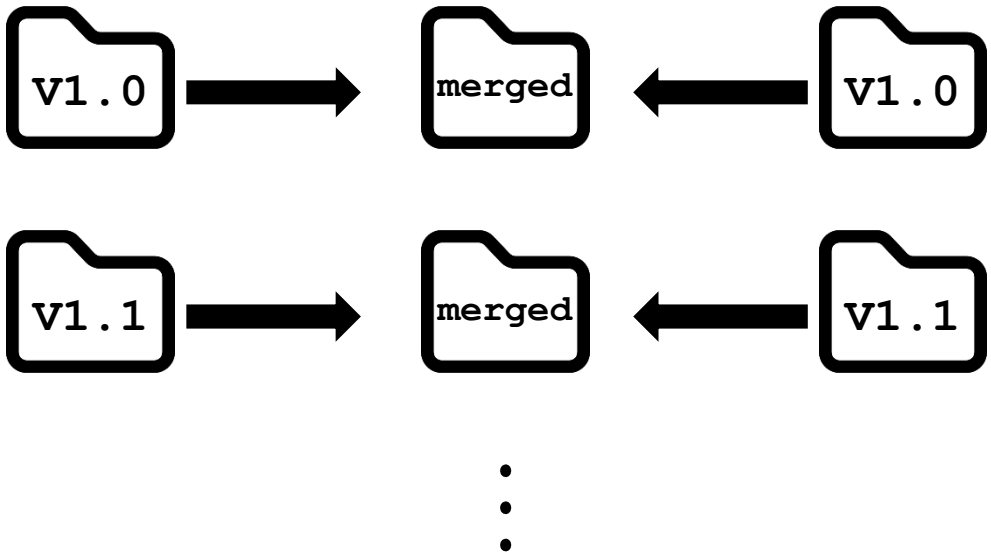


What if your code v1.1 contains mistakes?

What is Git?

Developer 1

Developer 2



Without a version control system would be a logistical nightmare

Why git?

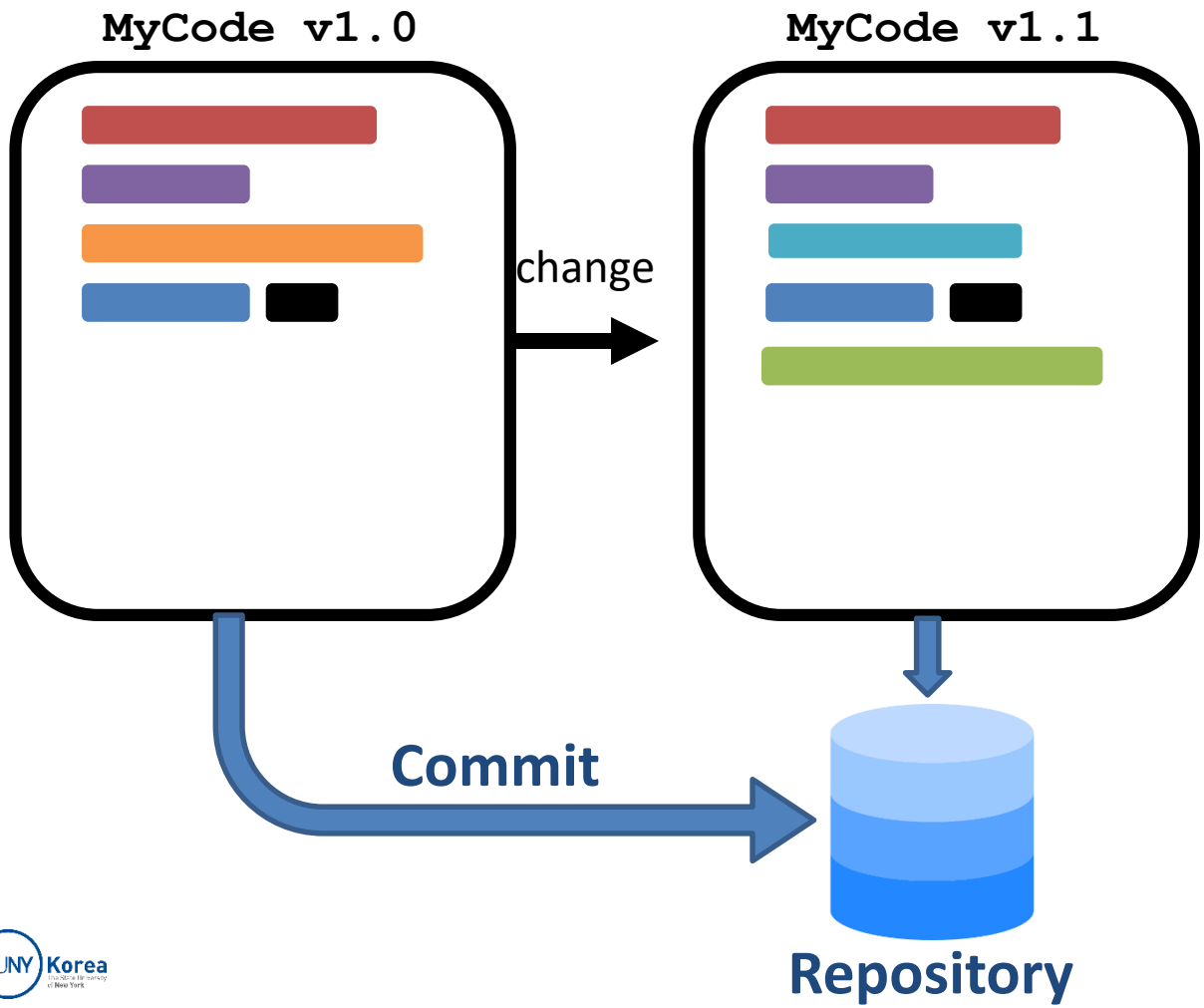
Version control systems resolve two problems

Track code history

Work together

Git commit

The different versions are the code are saved in a **repository**

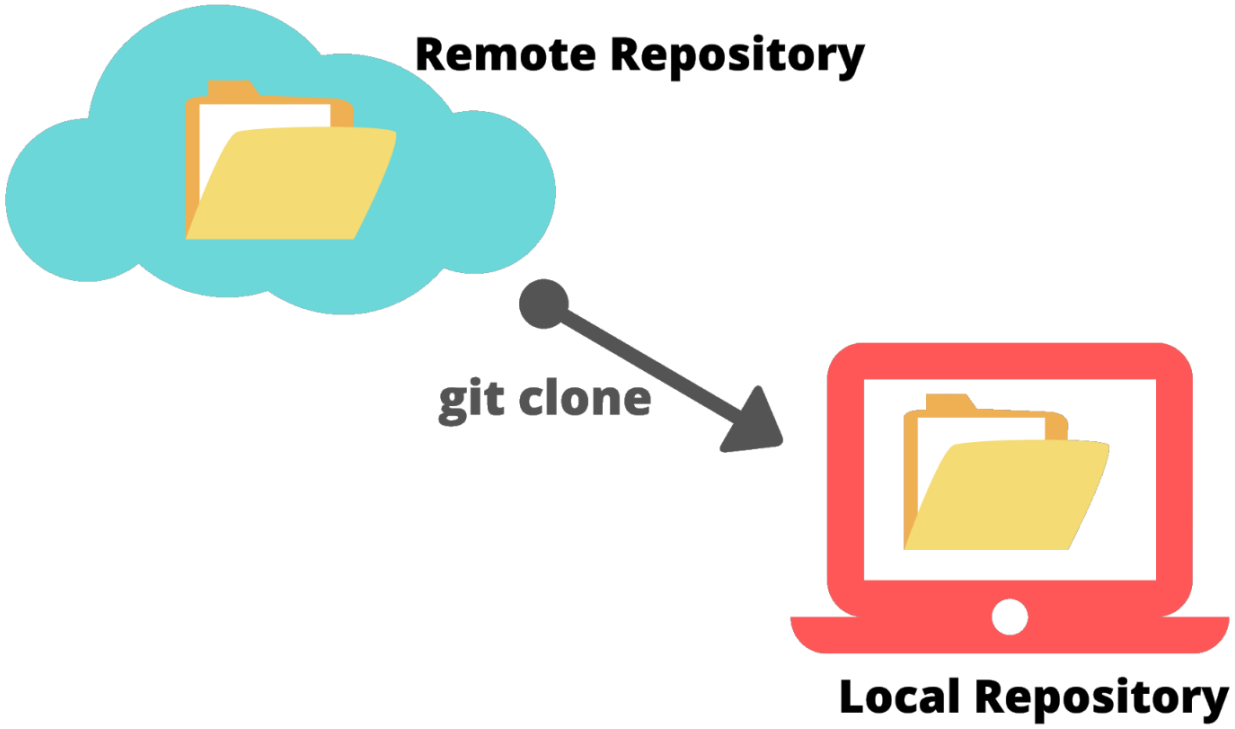


- You can record how your files look at a given point in time: a **snapshot**
- You decide when to take a snapshot and of what files
- You can visit the repository and go back in time as much as you want
- The act of taking a snapshot is called a **commit**
- All your snapshots are saved in a **repository**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

Git cloning

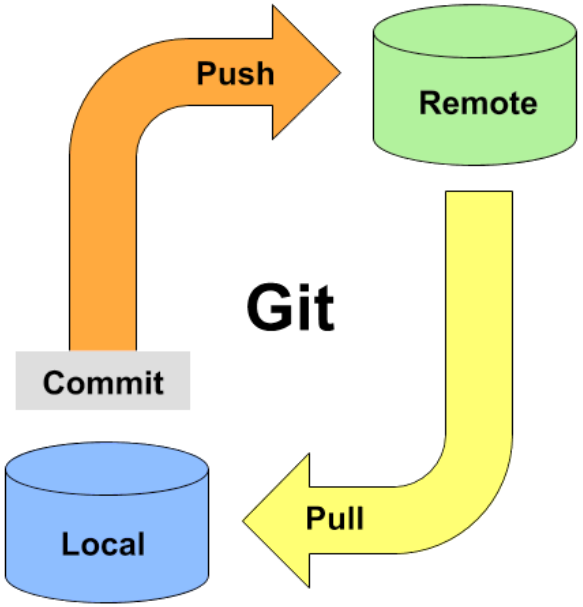
Copying a repository from a remote server is called **cloning**.

This is an essential function to work together



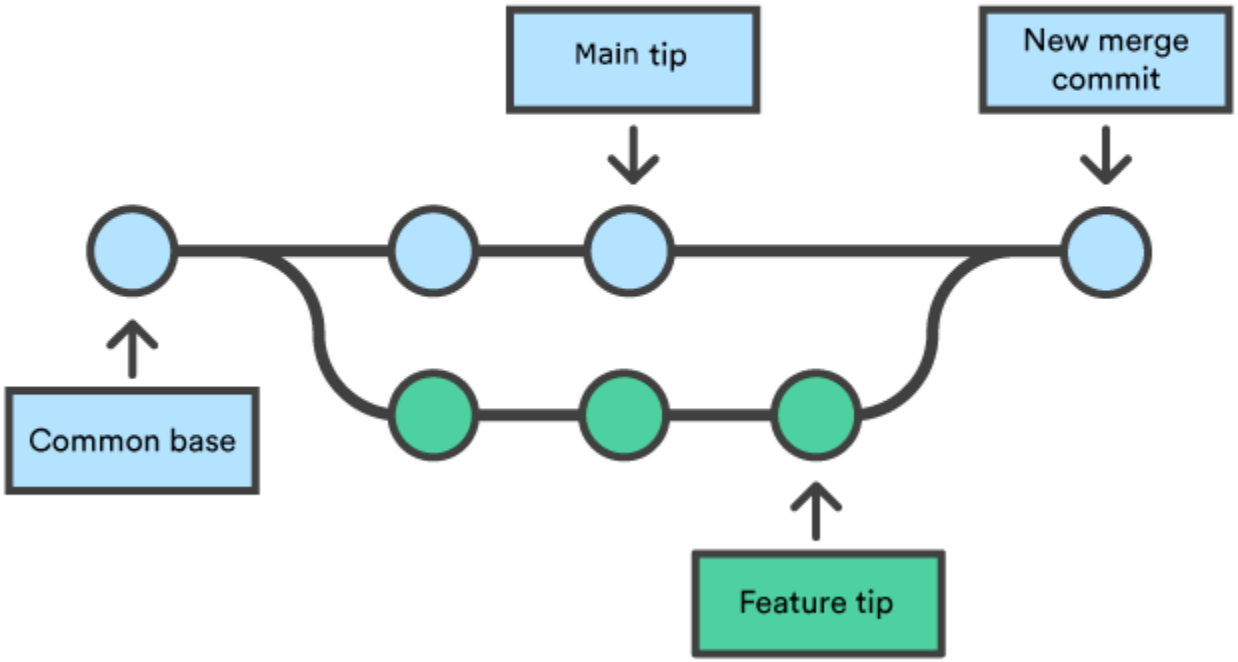
Git push and pull

- **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.

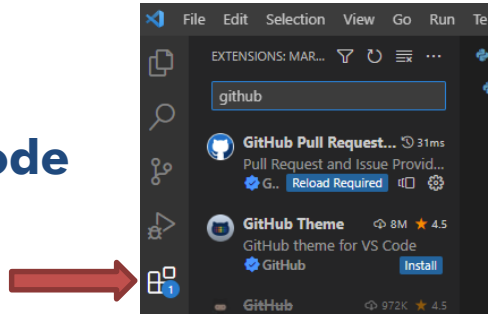
Ubuntu

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?

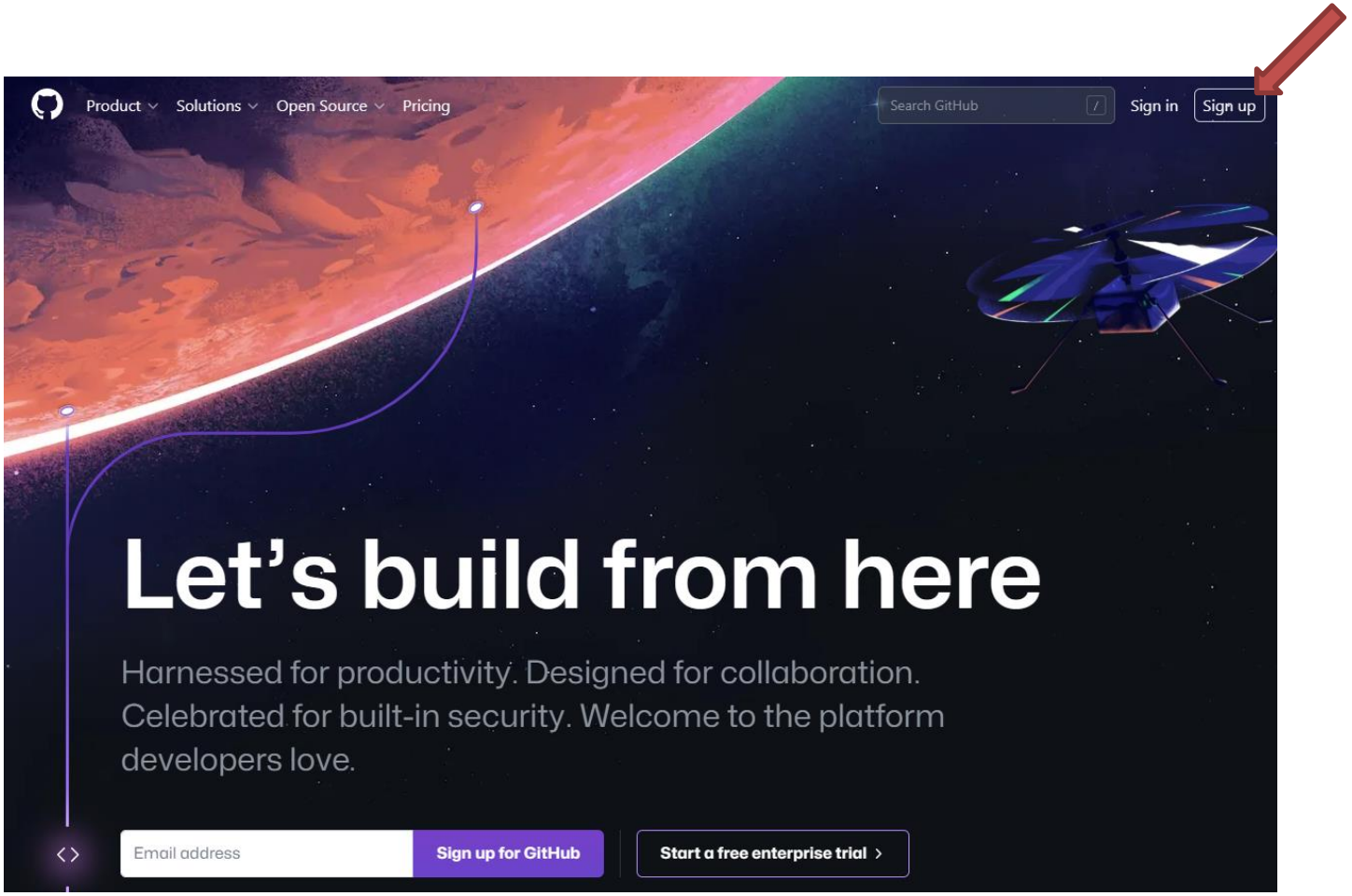
GitHub, on the other hand, is a web-based platform that provides hosting for Git repositories.

- 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/



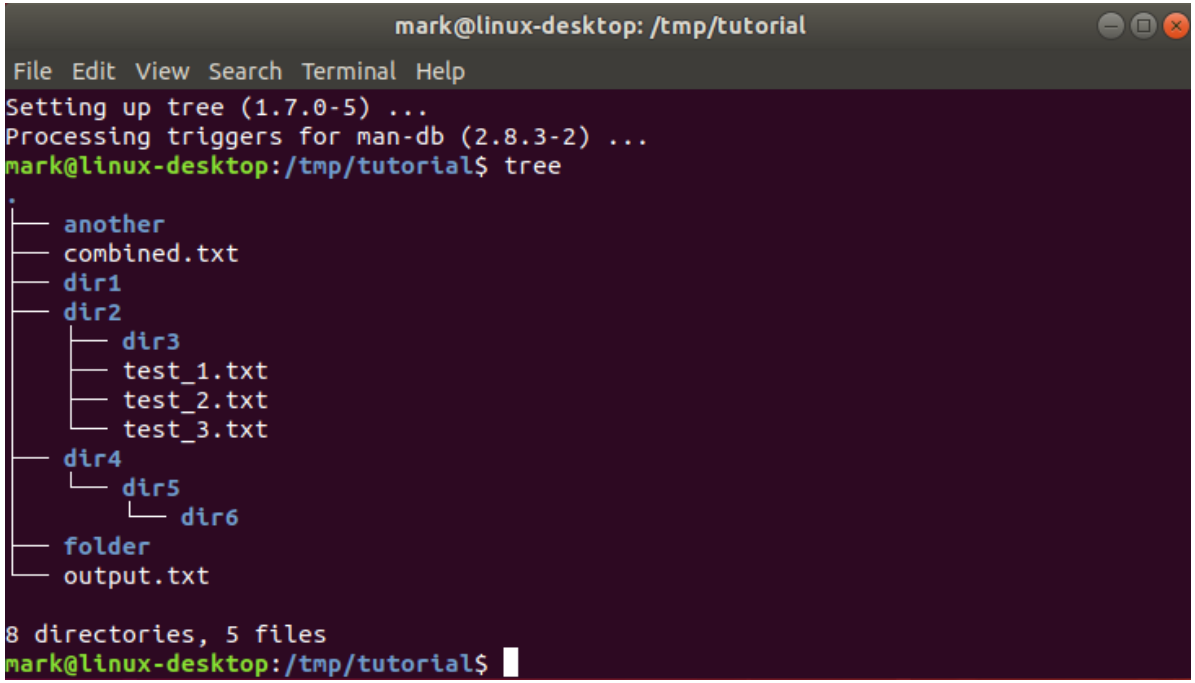
Why GitHub in CSE101?

- To get you familiar with good development practice
- To keep you 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

- Use your terminal -

What is a terminal?

a terminal on a computer is a text-based interface used to interact with the operating system and execute commands



The terminal is often preferred by experienced users over graphical user interfaces (GUIs) because it can provide more precise control and flexibility in system operations.

A pop icon

In many movies we can see the terminal used by hackers and talented programmer

Scene from Mr. Robot

Why using a terminal?

In modern OS, we have very intuitive Graphical User Interface (GUI). Why do we need a terminal?

Speed

Using a terminal with command-line tools can be faster and more efficient than using a GUI

Flexibility

The terminal allows users to customize and configure their environment to their liking.

Tools

Many powerful tools and utilities are designed to be used in a terminal environment.

Remote Access

The terminal allows users to remotely connect to servers and other systems over a network

Portability

The terminal works across all platforms.

How do we run a terminal

Starting a Terminal can be slightly different depending on your OS

On Windows

Press ⊞ and search cmd to run the prompt/terminal.

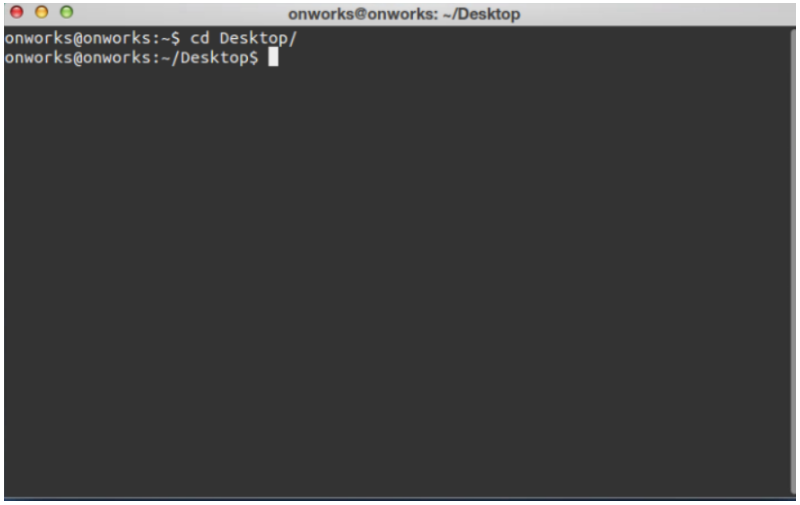
On Ubuntu

Press “Ctrl + Alt + T” to open the terminal and write

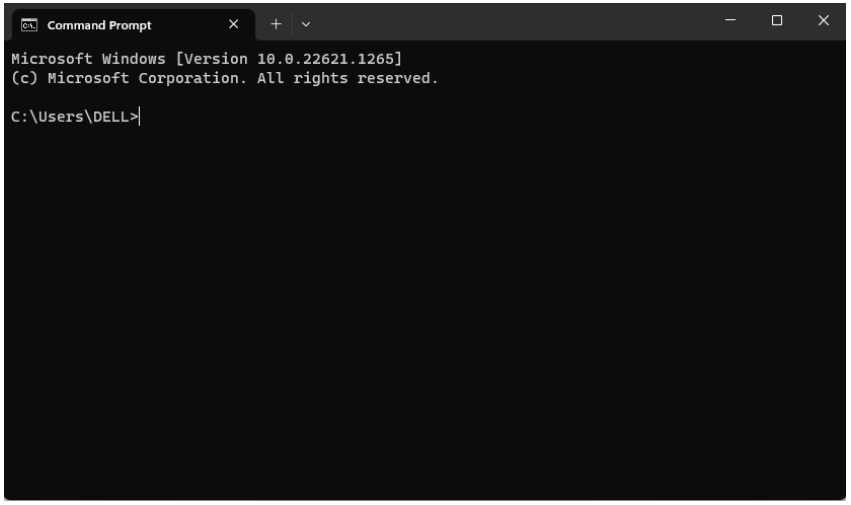
On Mac

Go to launchpad and search for terminal. Or run the spotlight and search for terminal.

How does it looks like?



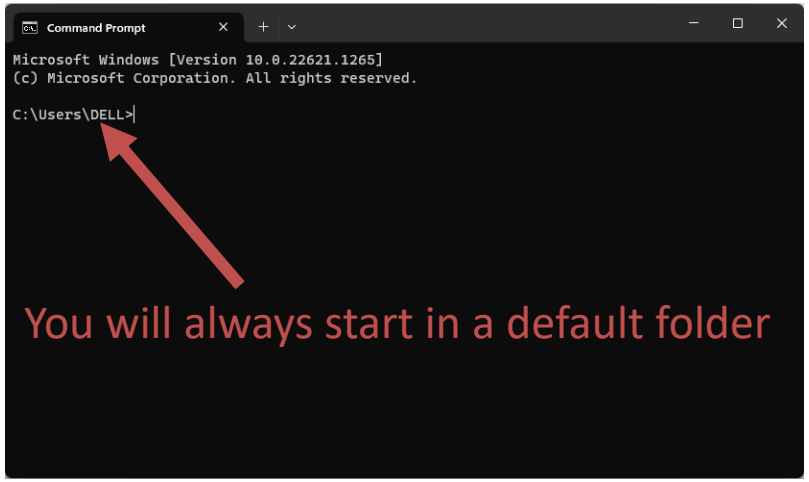
Mac / Linux



Windows

When I am?

When you open a terminal you get such a window



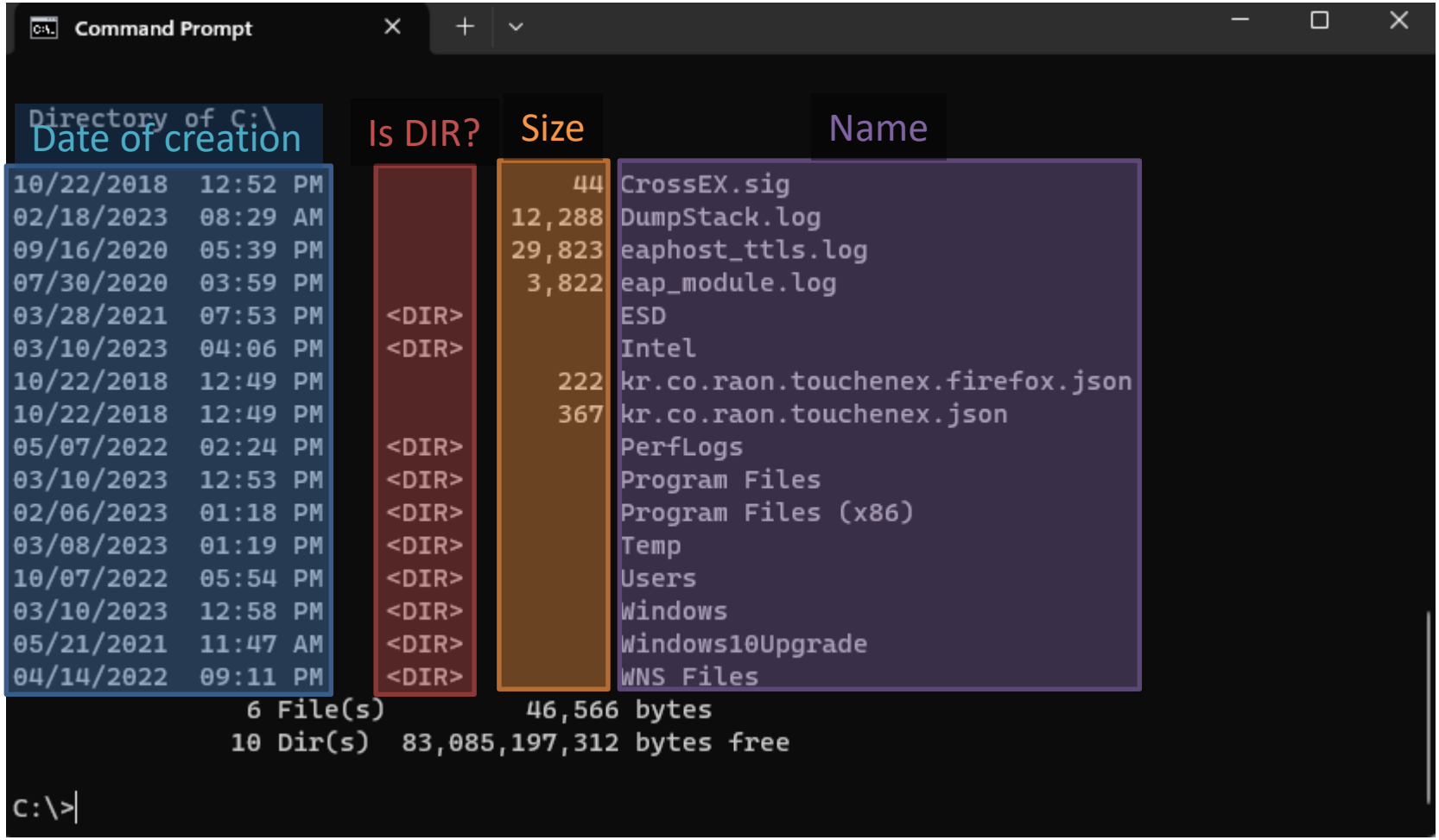
If you want to see the different files and folders at your current location type

```
dir
```

```
ls
```

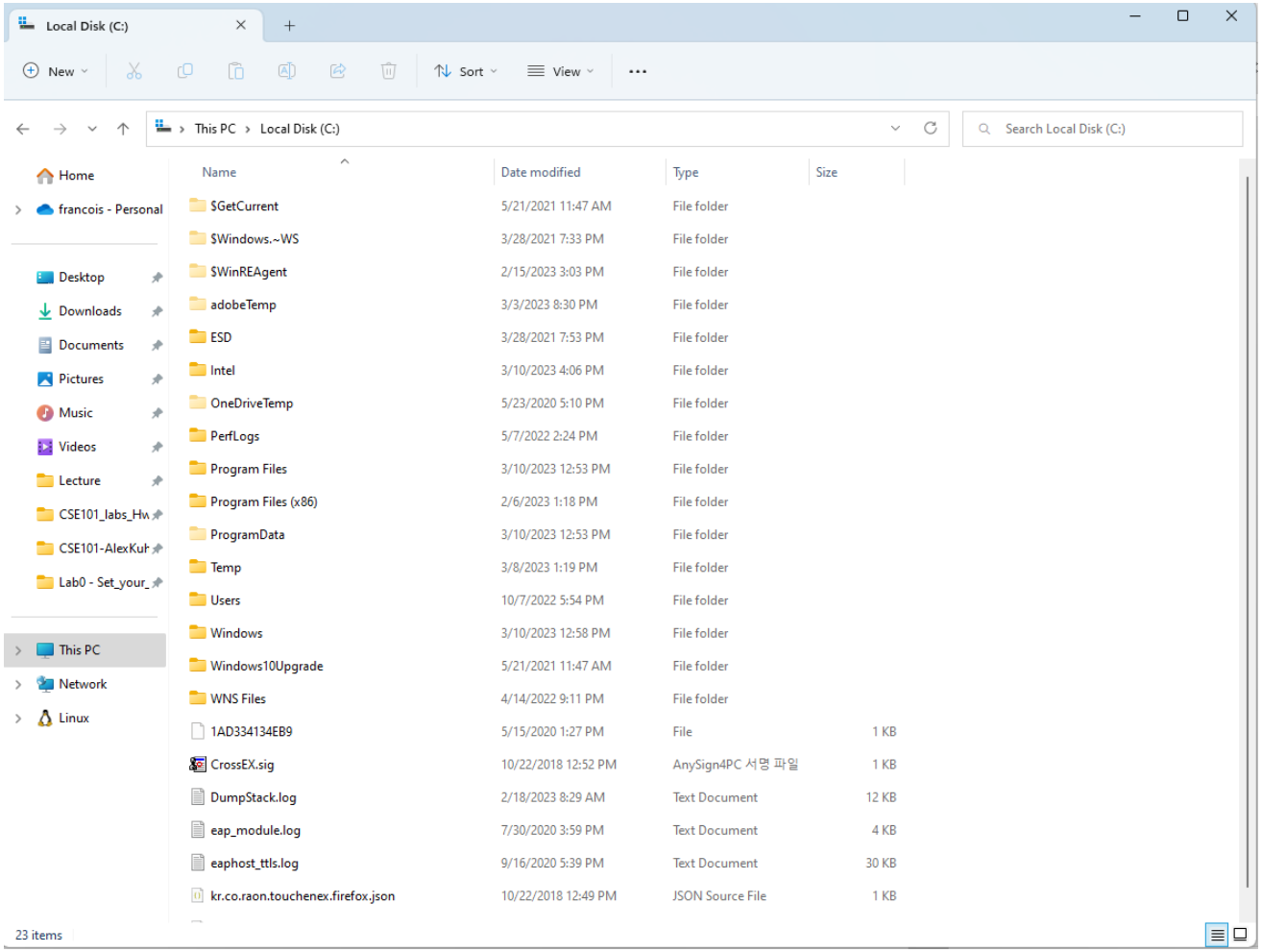
When I am?

The previous comment will show you everything that exist in your current folder



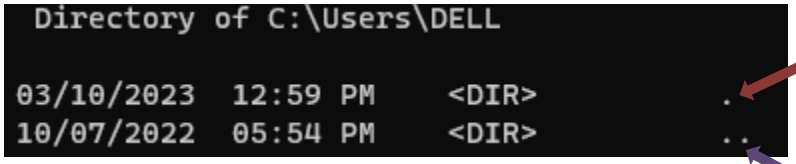
When I am?

It actually display exactly the same as the user interface!



How do I navigate?

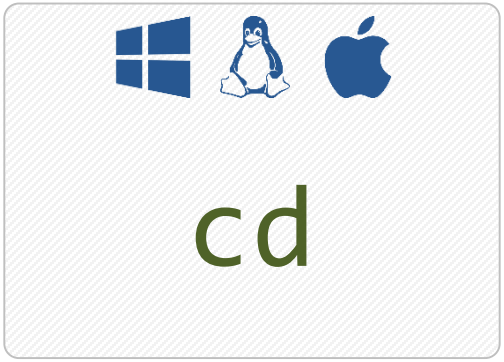
In general, the two fist folders to appear are the following



Current directory

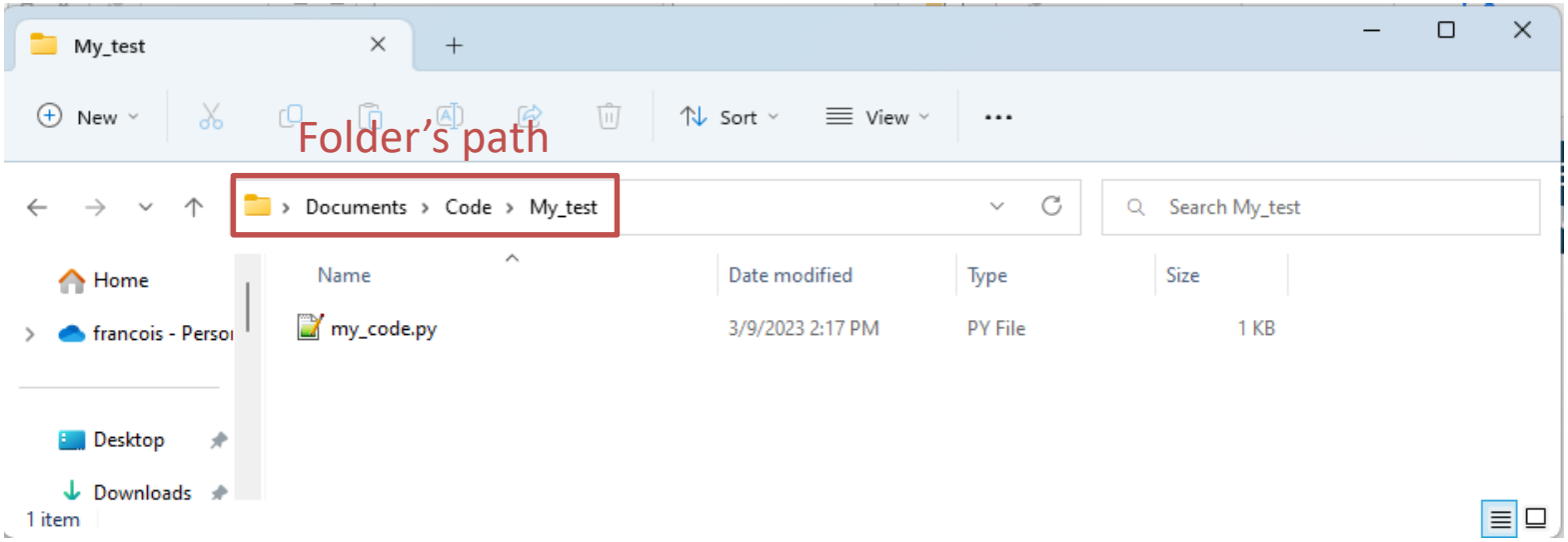
Parent/Previous directory

Now you can navigate wherever you want in your computer using the command:



How do I navigate?

Imagine you have a python file somewhere on the computer that you want to run via the Terminal



You can select the **absolute** path by pressing "Ctrl + L" you can then copy it using "Ctrl + C". To access this folder via the terminal, type

```
cd path_to_folder
```

How do I navigate?

Example

```
C:\Users\DELL>cd C:\Users\DELL\Documents\Code\My_test

C:\Users\DELL\Documents\Code\My_test>
```

You can also access folder via the **relative path**

For instance, if you want to access the **parent folder** type

```
cd ..
```

Or, if you want to access the **child folder** type

```
cd ./My_Child_foler
```

Or

```
cd My_Child_foler
```

Create a folder

To create a folder use the function

syntax

```
mkdir folder_name
```

For example

```
C:\Users\DELL\Documents\Code\My_test>mkdir my_new_folder

C:\Users\DELL\Documents\Code\My_test>dir
 Volume in drive C has no label.
 Volume Serial Number is 7019-4ACA

 Directory of C:\Users\DELL\Documents\Code\My_test

03/13/2023  03:06 PM    <DIR>          .
03/10/2023  01:41 PM    <DIR>          ..
03/09/2023  02:17 PM               188 my_code.py
03/13/2023  03:06 PM    <DIR>          my_new_folder
               1 File(s)            188 bytes
               3 Dir(s)  83,104,989,184 bytes free
```

This command create a new folder called *my_new_folder*

Note: Regardless your current folder, you can create a new folder anywhere on your computer using an absolute path

Create a file

To create a file use the function

`touch`

`type`

Syntax mac/Linux

```
touch filename.ext
```

Syntax Windows

```
type nul > filename.ext
```

Create a file

Delete a file or a folder

`rm`

`del`

Syntax mac/Linux

```
rm filename.ext
```

Syntax Windows

```
del filename.ext
```

! Dangerous !

You can also delete all files and folders in a directory using the recursive function. BUT be careful, it can be dangerous.

"With great power comes great responsibility"

To sum up

With a terminal you can do everything you do via the GUI (and much more). For now, try to just master these commands:

```
dir
cd
mkdir
```

If you do not remember how to use them type

```
help “command_name”
```

commands cheat sheet

Essential Windows CMD Commands You Should Know

MUO

ASSOC	Displays or modifies file extension associations
ATTRIB	Displays or changes file attributes
BREAK	Sets or clears extended CTRL+C checking
BCDBOOT	Used to copy critical files to the system partition and to crate a new system BCD store
BCDEDIT	Sets properties in boot database to control boot loading
CACLS	Shows or changes access control lists (ACLs) of files
CALL	Calls a batch program from another
CD	Shows the name of or changes to a current directory
CHCP	Displays or sets the active code page number
CHDIR	Displays the name of or changes to the current directory
CHKDSK	Checks a disk and displays a status report
CHKNTFS	Displays or modifies the checking of disk at boot time
CHOICE	Batch file command that allows users to select from a set of options
CIPHER	Displays or alters the encryption of directories (files) on NTFS partitions
CLIP	Redirects output off another command to the Windows clipboard
CLS	Clears the screen
CMD	Starts a new instance of the Windows command interpreter
CMDKEY	Creates, lists, and deletes stored user names and passwords or credentials
COLOR	Sets the default console colors
COMP	Compares the contents of two files or sets of files byte-by-byte
COMPACT	Displays or alters the compression of files on NTSF partitions
CONVERT	Converts FAT volumes to NTFS. You cannot convert the current drive
COPY	Copies one of more files to another location
DATE	Displays or sets the date
DEFRAG	Disk defragment accessory
DEL	Deletes one or more files
DIR	Displays a list of files and sub-directories in a directory
DISKCOMP	Compares the contents of two floppy disks
DISKCOPY	Copies the contents of one floppy disk to another
DISKPART	Displays or configures Disk Partition properties
DOSKEY	Edits command lines, recalls Windows commands, and creates macros
DRIVERQUERY	Displays current device driver status and properties
ECHO	Displays messages, or turns commands echoing on or off
ENDLOCAL	Ends localization of environment changes in a batch file
ERASE	Deletes one of more files
EXIT	Quits and closes the command shell
EXPAND	Expands compressed files
FC	Compares two files or sets of files, and displays the differences between them
FIND	Searches for a text string in a file or files
FINDSTR	Searches for strings in files
FOR	Runs a specified command for each item in a set
FORFILES	Selects files in a folder for batch processing
FORMAT	Formats a disc for use with Windows
FSUTIL	Displays or configures the file system properties
FTYPE	Displays or modifies file types used in file extensions associations
GOTO	Directs the Windows command interpreter to a labeled line in a batch program
GPRESULT	Displays Group Policy Information for machine or user.
GRAFTABL	Enables Windows to display an extended character set in graphics mode
HELP	Provides help information for Windows commands
ICACLS	Display, modify, backup, or restore ACLs for files and directories
IF	Performs conditional processing in batch programs.
IPCONFIG	Displays all current TCP/IP network configuration values
LABEL	Creates, changes, or deletes the volume label of a disk
MD	Creates a directory
MKDIR	Creates a directory
MKLINK	Creates Symbolic Links and Hard Links
MODE	Configures a system device
MORE	Displays output one screen at a time
MOVE	Moves one or more files from one directory to another directory
NETSTAT	Returns a list of currently open ports and related IP addresses
OPENFILES	Queries, displays, or disconnects open files or files opened by network users
PATH	Displays or sets a search path for executable files
PATHPING	Advanced version of ping, used if there are multiple routers between your PC and the device you're testing
PAUSE	Suspends processing of a batch file
PING	Sends a series of test packets to the specified address
POPD	Restores the previous value of the current directory saved by PUSHD
POWERCFG	Manages and tracks energy utilization and power consumption
PRINT	Prints a text file
PROMPT	Changes the Windows command prompt
PUSHD	Saves the current directory then changes it
RD	Removes a directory
RECIMG	Configures the custom Windows recovery image
RECOVER	Recovers readable information from a bad or defective disk
REM	Designates comments (remarks) in batch files
REN	Renames a file or files
RENAME	Renames a file or files
REPLACE	Replaces files
RMDIR	Removes a directory
ROBOCOPY	Advanced utility to copy files and directory trees
SET	Displays, sets, or removes environment variables for current session
SETLOCAL	Begins localization of environment changes in a batch file
SETX	Sets environment variables
SFC	Finds corrupt/missing files, replaces them with cached copies
SC	Displays or configures services (background processes)
SCHTASKS	Schedules commands and programs to run on a computer
SHIFT	Shifts the position of replaceable parameters in batch files
SHUTDOWN	Allows proper local or remote shutdown of machine
SORT	Sorts input
START	Starts a separate window to run a specified programs or command
SUBST	Associates a path with a drive letter
SYSTEMINFO	Displays machine specific properties and configuration
TAKEOWN	Allows an administrator to take ownership of a file
TASKLIST	Displays all currently running tasks including services
TASKKILL	Kill running process or applications
TIME	Displays or sets the system time
TIMEOUT	Pauses the command processor for the specified number of seconds
TITLE	Sets the window title for a CMD.EXE session
TRACERT	Returns information about each step in the route between your PC and the target
TREE	Graphically displays the directory structure of a drove or path
TYPE	Displays the contents of a text file
VER	Displays the Windows version
VERIFY	Tells Windows whether or verify that your files are written correctly to a disk
VOL	Displays a disk volume label and serial number
VSSADMIN	Volume Shadow Copy Service administration tool
WHERE	Displays the locations of files that match a search pattern
WMIC	Displays WMI information inside interactive command shell
XCOPY	Copies files and directory trees

Unix/Linux Command Reference

FOSSwire.com

File Commands

ls – directory listing
ls -al – formatted listing with hidden files
cd *dir* - change directory to *dir*
cd – change to home
pwd – show current directory
mkdir *dir* – create a directory *dir*
rm *file* – delete *file*
rm -r *dir* – delete directory *dir*
rm -f *file* – force remove *file*
rm -rf *dir* – force remove directory *dir* *
cp *file1 file2* – copy *file1* to *file2*
cp -r *dir1 dir2* – copy *dir1* to *dir2*; create *dir2* if it doesn't exist
mv *file1 file2* – rename or move *file1* to *file2*
if *file2* is an existing directory, moves *file1* into directory *file2*
ln -s *file link* – create symbolic link *link* to *file*
touch *file* – create or update *file*
cat > *file* – places standard input into *file*
more *file* – output the contents of *file*
head *file* – output the first 10 lines of *file*
tail *file* – output the last 10 lines of *file*
tail -f *file* – output the contents of *file* as it grows, starting with the last 10 lines

Process Management

ps – display your currently active processes
top – display all running processes
kill *pid* – kill process id *pid*
killall *proc* – kill all processes named *proc* *
bg – lists stopped or background jobs; resume a stopped job in the background
fg – brings the most recent job to foreground
fg *n* – brings job *n* to the foreground

File Permissions

chmod *octal file* – change the permissions of *file* to *octal*, which can be found separately for user, group, and world by adding:

- 4 – read (r)
- 2 – write (w)
- 1 – execute (x)

Examples:
chmod 777 – read, write, execute for all
chmod 755 – rwx for owner, rx for group and world
For more options, see **man chmod**.

SSH

ssh *user@host* – connect to *host* as *user*
ssh -p *port user@host* – connect to *host* on port *port* as *user*
ssh-copy-id *user@host* – add your key to *host* for *user* to enable a keyed or passwordless login

Searching

grep *pattern files* – search for *pattern* in *files*
grep -r *pattern dir* – search recursively for *pattern* in *dir*
command* | grep *pattern – search for *pattern* in the output of *command*
locate *file* – find all instances of *file*

System Info

date – show the current date and time
cal – show this month's calendar
uptime – show current uptime
w – display who is online
whoami – who you are logged in as
finger *user* – display information about *user*
uname -a – show kernel information
cat /proc/cpuinfo – cpu information
cat /proc/meminfo – memory information
man *command* – show the manual for *command*
df – show disk usage
du – show directory space usage
free – show memory and swap usage
whereis *app* – show possible locations of *app*
which *app* – show which *app* will be run by default

Compression

tar cf *file.tar files* – create a tar named *file.tar* containing *files*
tar xf *file.tar* – extract the files from *file.tar*
tar czf *file.tar.gz files* – create a tar with Gzip compression
tar xzf *file.tar.gz* – extract a tar using Gzip
tar cjf *file.tar.bz2* – create a tar with Bzip2 compression
tar xjf *file.tar.bz2* – extract a tar using Bzip2
gzip *file* – compresses *file* and renames it to *file.gz*
gzip -d *file.gz* – decompresses *file.gz* back to *file*

Network

ping *host* – ping *host* and output results
whois *domain* – get whois information for *domain*
dig *domain* – get DNS information for *domain*
dig -x *host* – reverse lookup *host*
wget *file* – download *file*
wget -c *file* – continue a stopped download

Installation

Install from source:
./configure
make
make install
dpkg -i *pkg.deb* – install a package (Debian)
rpm -Uvh *pkg.rpm* – install a package (RPM)

Shortcuts

Ctrl+C – halts the current command
Ctrl+Z – stops the current command, resume with **fg** in the foreground or **bg** in the background
Ctrl+D – log out of current session, similar to **exit**
Ctrl+W – erases one word in the current line
Ctrl+U – erases the whole line
Ctrl+R – type to bring up a recent command
!! - repeats the last command
exit – log out of current session

* use with extreme caution.