

Get started with Python

```
... object to mirror  
mirror_mod.mirror_object =
```

```
operation == "MIRROR_X":  
mirror_mod.use_x = True
```

```
mirror_mod.use_y = False  
mirror_mod.use_z = False
```

```
operation == "MIRROR_Y":  
mirror_mod.use_x = False
```

```
mirror_mod.use_y = True  
mirror_mod.use_z = False
```

```
operation == "MIRROR_Z":  
mirror_mod.use_x = False
```

```
mirror_mod.use_y = False  
mirror_mod.use_z = True
```

```
selection at the end -add  
mirror_ob.select= 1
```

```
mirror_ob.select=1  
context.scene.objects.active =
```

```
("Selected" + str(modifier.name))  
mirror_ob.select = 0
```

```
= bpy.context.selected_objects[0]  
data.objects[one.name].select =
```

```
print("please select exactly one")  
-- OPERATOR CLASSES --
```

```
class MirrorX(bpy.types.Operator):  
    """Mirror X mirror to the selected object"""
```

```
    bl_label = "Mirror X"  
    bl_options = {'DEFAULT', 'REGISTERED'}  
    mirror_x = False
```

```
    @classmethod  
    def poll(cls, context):  
        obj = context.active_object
```

```
        return obj.type == 'MESH'  
    def execute(self, context):  
        obj = context.active_object
```

Why Python?

1) Easy to Learn and Use

2) Mature and Supportive Python Community

3) Support from Renowned Corporate Sponsors

4) Hundreds of Python Libraries and Frameworks

5) Versatility, Efficiency, Reliability, and Speed

6) Big data, Machine Learning and Cloud Computing

7) First-choice Language

8) The Flexibility of Python Language

9) Use of python in academics

10) Automation

First Example: Hello, World!

+ **print**("Hello, World!")



How to code in Python?

- + Python offers a number of different options.
- + The code can be written in a text editor and stored as a file that is then run from the command line window or an integrated development environment (IDE).

How to run a Python app

- + To write and run the "Hello, World!", open your favorite editor (Atom, gedit, Emacs etc.), type the given line and save the file with a suitable filename, for instance, `hello.py`.
- + Then, open a terminal window, navigate to the directory where you saved the file, and type `python hello.py` using a regular terminal.

Ways to use Python

- + Python offers some alternatives to the traditional style of programming using a text editor and a terminal window, and some of these alternatives can be very useful when learning to program.
- + You can use Python interactively by simply typing `python` in a terminal window, without a subsequent file name.
- + This will open an environment for typing and running Python commands, which is not very suitable for writing programs over several lines, but extremely useful for testing Python commands and statements.

Terminal

```
goncalosantosmarques — Python — 80x24
Last login: Wed Aug  4 14:04:54 on ttys000
-bash: /Users/goncalosantosmarques/.profile: No such file or directory

The default interactive shell is now zsh.
To update your account to use zsh, please run `chsh -s /bin/zsh`.
For more details, please visit https://support.apple.com/kb/HT208050.
MacBook-Pro-de-Goncalo-195:~ goncalosantosmarques$ python3
Python 3.8.5 (v3.8.5:580fbb018f, Jul 20 2020, 12:11:27)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

```
goncalosantosmarques — Python — 80x24
Last login: Wed Aug  4 09:14:46 on ttys000
-bash: /Users/goncalosantosmarques/.profile: No such file or directory

The default interactive shell is now zsh.
To update your account to use zsh, please run `chsh -s /bin/zsh`.
For more details, please visit https://support.apple.com/kb/HT208050.
MacBook-Pro-de-Goncalo-195:~ goncalosantosmarques$ python

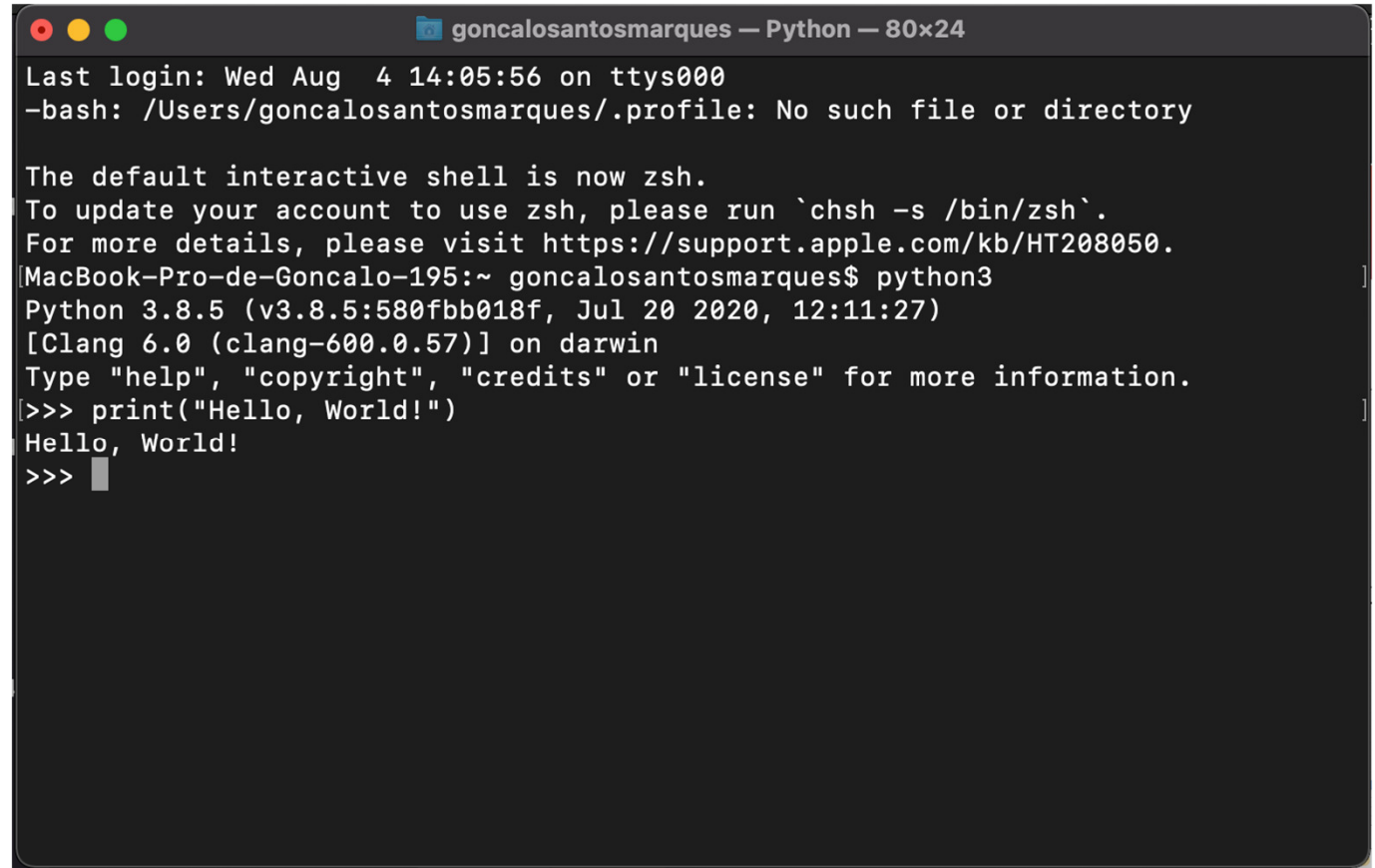
WARNING: Python 2.7 is not recommended.
This version is included in macOS for compatibility with legacy software.
Future versions of macOS will not include Python 2.7.
Instead, it is recommended that you transition to using 'python3' from within Te
rminal.

Python 2.7.16 (default, Jun 18 2021, 03:23:53)
[GCC Apple LLVM 12.0.5 (clang-1205.0.19.59.6) [+internal-os, ptrauth-isa=deploy
on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Attention:

- + Several Python versions are available.
- + You can have different Python versions installed on your computer.
- + We will use Python 3 in this class.
- + If you do not have Python installed find it here:
<https://www.python.org/downloads/>

Hello World!

A terminal window with a dark background and light text. The title bar at the top reads 'goncalosantosmarques — Python — 80x24'. The terminal output shows a login message, a shell change from bash to zsh, and the execution of a Python script that prints 'Hello, World!'.

```
goncalosantosmarques — Python — 80x24
Last login: Wed Aug  4 14:05:56 on ttys000
-bash: /Users/goncalosantosmarques/.profile: No such file or directory

The default interactive shell is now zsh.
To update your account to use zsh, please run `chsh -s /bin/zsh`.
For more details, please visit https://support.apple.com/kb/HT208050.
MacBook-Pro-de-Goncalo-195:~ goncalosantosmarques$ python3
Python 3.8.5 (v3.8.5:580fbb018f, Jul 20 2020, 12:11:27)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> print("Hello, World!")
Hello, World!
>>> 
```

Jupyter notebooks

- + *Jupyter notebooks* are a form of interactive notebooks that combine code and text.
- + The notebooks are viewed through a browser and look quite like a simple web page, but with the important difference that the code segments are "live" Python code that can be run, changed, and re-run while reading the document.
- + These features are particularly useful for teaching purposes, since detailed explanations of new concepts are easily combined with interactive examples.

- Data.ipynb
- Fasta.ipynb
- Julia.ipynb
- Linear Regression.ipynb
- Lorenz.ipynb
- lorenz.py
- R.ipynb
- untitled.dio
- untitled1.dio
- untitled2.dio
- untitled3.dio
- untitled4.dio
- untitled5.dio
- untitled6.dio

In this section we will start with a quick intuitive walk-through of the mathematics behind this well-known problem, before seeing how before moving on to see how linear models can be generalized to account for more complicated patterns in data.

We begin with

```
[1]: %matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
```

Simple

We will start

where α is

Consider the

```
[2]: rng = np.random.randn(1000)
x = 10 * rng
y = 2 * x
plt.scatter(x, y)
```



We can use

```
[3]: from sklearn
```

```
<h1><font
color="#f37626">pyt</font>hon
not<font
color="#f37626">e</font>book</h1>
```

Slide Type

Raw NBConvert Format

Advanced Tools

Cell Metadata

```
{}
```

Notebook Metadata

```
{
  "kernelspec": {
    "display_name": "Python 3",
    "language": "python",
    "name": "python3"
  },
  "language_info": {
    "codemirror_mode": {
      "name": "ipython",
      "version": 3
    },
    "file_extension": ".py",
    "mimetype": "text/x-python",
    "name": "python",
    "nbconvert_exporter": "python",
    "pygments_lexer": "ipython3",
    "version": "3.6.7"
  },
  "toc-autonumbering": false,
  "toc-showcode": true,
  "toc-showmarkdowntxt": true
}
```

Launcher

Notebook

- Python 3
- C++11
- C++14
- C++17
- Julia 1.1.0
- phylogenetics (Python 3.7)
- R

Console

- Python 3
- C++11
- C++14
- C++17

Julia

```
[10]: using RDatasets, Gadfly
plot(dataset("datasets", "iris"), x="Sepal.Length", y="Petal.Length", color=:Species)
```

```
[8]: eigen(x)
[8]: Eigen{Complex{Float64},Complex{Float64},Array{Complex{Float64},2},Array{Complex{Float64},1}}
eigenvalues:
10-element Array{Complex{Float64},1}:
 4.793881566545466 + 0.01im
-0.9445989635995898 + 0.01im
```

python notebook

```
***
[1]: %matplotlib inline
from ipywidgets import interactive, fixed
```

We explore the Lorenz system of differential equations:

$$\begin{aligned} \dot{x} &= \sigma(y - x) \\ \dot{y} &= \rho x - y - xz \\ \dot{z} &= -\beta z + xy \end{aligned}$$

Let's change (σ, β, ρ) with ipywidgets and examine the trajectories.

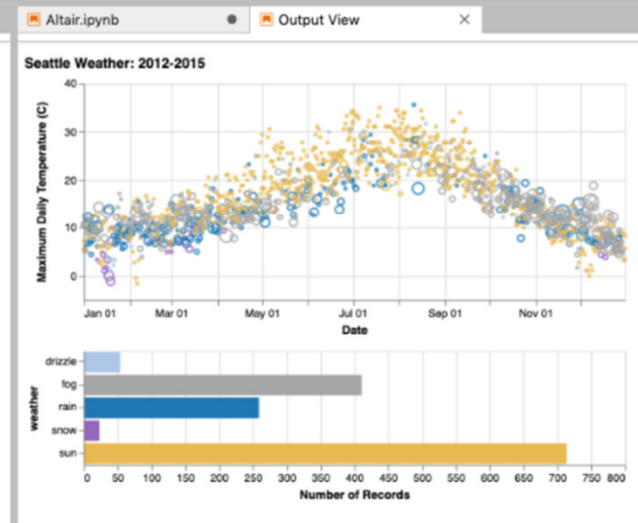
```
[2]: from lorenz import solve_lorenz
w = interactive(solve_lorenz, sigma=(0.0, 50.0), rho=(10.0, 20.0), beta=(2.666666666666667, 2.666666666666667))
```

R

```
[3]: ggplot(data=iris, aes(x=Sepal.Length, y=Petal.Length, color=Species)) +
  geom_point()
```

```
[1]: head(iris)
```

Sepal.Length	Sepal.Width	Petal.Length
5.1	3.5	1.4
4.9	3.0	1.4





Basic mathematical functions

- + `import math`
- + This module provides access to the mathematical functions defined by the C standard.

Square root

- + `import math`
- + `math.sqrt(4) # 2.0`

Note that: attempting to use this function with negative arguments will lead to errors.

Trigonometric functions

- + `theta = math.pi / 4`
- + `math.cos(theta) # 0.7071067811865476`
- + `math.sin(theta) # 0.7071067811865475`
- + `math.tan(theta) # 0.9999999999999999`

Inverse trigonometric functions

- + `math.asin(-1)` # -1.5707963267948966
- + `math.acos(-1)` # 3.141592653589793
- + `math.atan(1)` # 0.7853981633974483

Logarithm functions

- + `math.log(10)` # 2.302585092994046
- + `math.log(10, 10)` # 1.0

Others

Gamma functions: `math.gamma(5)` # 24.0

Gaussian error : `math.erf(2)` # 0.9953222650189527

Factorial: `math.factorial(5)` # 120

Numpy arrays

- + NumPy provides an N-dimensional array type, the ndarray, which describes a collection of “items” of the same type.
- + The items can be indexed using for example N integers.

Create and access to array data

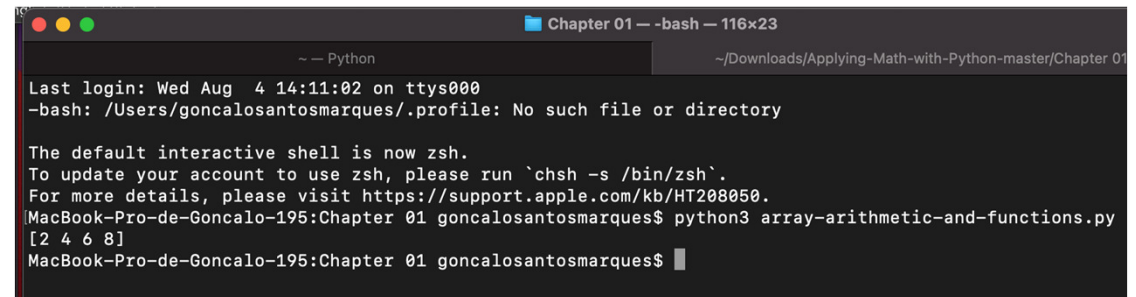
- + `import numpy as np`
- + `ary = np.array([1, 2, 3, 4])` # `array([1, 2, 3, 4])`
- + `ary[0]` # 1
- + `ary[2]` # 3
- + `ary[::2]` # `array([1, 3])`

Array Type

```
+ np.array([1, 2, 3, 4], dtype=np.float32)
+ # array([1., 2., 3., 4.], dtype=float32)
+ arr = np.array([1, 2, 3, 4])
+ print(arr.dtype) # dtype('int64')
+ arr.dtype = np.float32
+ print(arr)
+ # [1.e-45 0.e+00 3.e-45 0.e+00 4.e-45 0.e+00 6.e-45 0.e+00]
+ arr = arr.astype(np.float32)
+ print(arr)
+ # [1. 2. 3. 4.]
```

Array operations

```
import numpy as np
arr_a = np.array([1, 2, 3, 4])
arr_b = np.array([1, 0, -3, 1])
arr_a + arr_b # array([2, 2, 0, 5])
arr_a - arr_b # array([0, 2, 6, 3])
arr_a * arr_b # array([ 1, 0, -9, 4])
arr_b / arr_a # array([ 1., 0., -1., 0.25])
arr_b**arr_a # array([1, 0, -27, 1])
arr = np.array([1, 2, 3, 4])
new = 2*arr
print(new) # [2, 4, 6, 8]
```



A terminal window titled "Chapter 01 — -bash — 116x23" with a sub-header "Python" and a path "~/Downloads/Applying-Math-with-Python-master/Chapter 01". The terminal shows the following output:

```
Last login: Wed Aug  4 14:11:02 on ttys000
-bash: /Users/goncalosantosmarques/.profile: No such file or directory

The default interactive shell is now zsh.
To update your account to use zsh, please run `chsh -s /bin/zsh`.
For more details, please visit https://support.apple.com/kb/HT208050.
MacBook-Pro-de-Goncalo-195:Chapter 01 goncalosantosmarques$ python3 array-arithmetic-and-functions.py
[2 4 6 8]
MacBook-Pro-de-Goncalo-195:Chapter 01 goncalosantosmarques$
```

Create arrays

- + `import numpy as np`
- + `np.linspace(0, 1, 5) # array([0., 0.25, 0.5, 0.75, 1.0])`
- + `np.arange(0, 1, 0.3) # array([0.0, 0.3, 0.6, 0.9])`

Higher dimensional Arrays

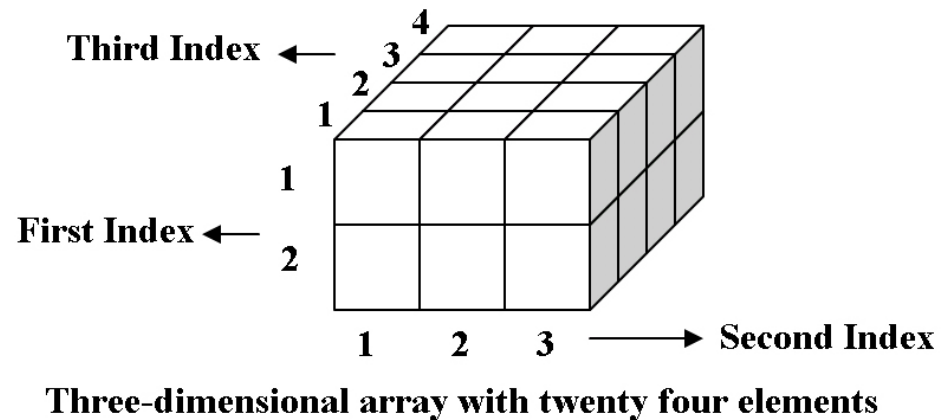
- + `import numpy as np`
- + `mat = np.array([[1, 2], [3, 4]])`
- + `vec = np.array([1, 2])`
- + `mat.shape` `# (2, 2)`
- + `vec.shape` `# (2,)`

Reshape

- + NumPy array is stored in one-dimensional array.
- + An array can be reshaped.
- + `mat.reshape(4,)`
- + `# array([1, 2, 3, 4])`

Higher dimensions

- + `mat1 = [[1, 2], [3, 4]]`
- + `mat2 = [[5, 6], [7, 8]]`
- + `mat3 = [[9, 10], [11, 12]]`
- + `arr_3d = np.array([mat1, mat2, mat3])`
- + `arr_3d.shape` # (3, 2, 2)
- + `mat[0, 0]` # 1 - top left element
- + `mat[1, 1]` # 4 - bottom right element
- + `mat[:, 0]` # `array([1, 3])`



Matrices

This matrix is a 3x4 (pronounced "three by four") matrix because it has 3 rows and 4 columns.

$$\begin{array}{cccc} & \text{4 columns} & & \\ & \downarrow \downarrow \downarrow \downarrow & & \\ \left[\begin{array}{cccc} 2 & -5 & -11 & 0 \\ -9 & 4 & 6 & 13 \\ 4 & 7 & 12 & -2 \end{array} \right] & \begin{array}{l} \leftarrow \\ \leftarrow \\ \leftarrow \end{array} & \text{3 rows} \end{array}$$

Python Matrix

+ A = [[1, 4, 5],
[-5, 8, 9]]

$$\begin{bmatrix} 1 & 4 & 5 \\ -5 & 8 & 9 \end{bmatrix}$$

Transpose

The transpose \mathbf{A}^T of a matrix \mathbf{A} can be obtained by reflecting the elements along its main diagonal.

Repeating the process on the transposed matrix returns the elements to their original position.

$$\mathbf{A} = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$$

Transpose in Python

- + `mat = np.array([[1, 2], [3, 4]])`

- + `mat.transpose()`

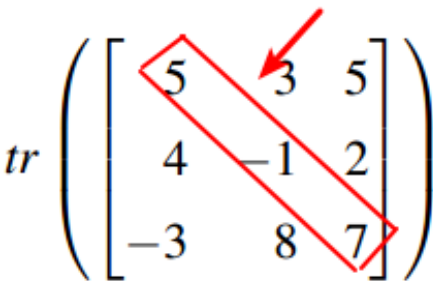
 - # `array([[1, 3],`
 - # `[2, 4]])`

- + `mat.T`

 - # `array([[1, 3],`
 - # `[2, 4]])`

Trace (Linear algebra)

- + In linear algebra, the **trace** of a square matrix \mathbf{A} , denoted $\text{tr}(\mathbf{A})$, is defined to be the sum of elements on the main diagonal (from the upper left to the lower right) of \mathbf{A} .


$$\text{tr} \left(\begin{bmatrix} 5 & 3 & 5 \\ 4 & -1 & 2 \\ -3 & 8 & 7 \end{bmatrix} \right) = 5 - 1 + 7 = 11.$$

Trace in Python

- + `A = np.array([[1, 2], [3, 4]])`
- + `A.trace()` # 5

Matrix multiplication

$$\begin{matrix} & A & & B & \\ \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} & \times & \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix} & = & \begin{bmatrix} 19 & 22 \\ 43 & 50 \end{bmatrix} \end{matrix}$$

$$\begin{aligned} 1 \times 6 + 2 \times 8 &= 22 \\ 1 \times 5 + 2 \times 7 &= 19 \\ 3 \times 5 + 4 \times 7 &= 43 \\ 3 \times 6 + 4 \times 8 &= 50 \end{aligned}$$

8 multiplications

Matrix multiplication in Python

```
A = np.array([[1, 2], [3, 4]])
```

```
B = np.array([[5, 6], [7, 8]])
```

```
print(A@B)
```

```
#[[19 22]
```

```
 [43 50]]
```

Other operations in Python

- + `A = np.array([[1, 2], [3, 4]])`
- + `from numpy import linalg`
- + `linalg.det(A) # -2.00000000000000000004`
- + `linalg.inv(A)`
- + `# array([[-2. , 1.],`
- + `# [1.5, -0.5]])`

Computing with Formulas



Programming Simple Mathematics

$$A = P (1 + (r/100))^n$$

P is the initial deposit (the *principal*),

r is the yearly interest rate given in percent,

n is the number of years

A is the final amount.

Let's code

- + The task is now to write a program that computes A for given values of P , r and n .
- + To evaluate the formula above, we first need to assign values to P , r and n , and then make the calculation.
- + **$P = 100, r = 5.0$, and $n = 7$**

Option 1:

We can use our terminal and type the following:

```
print(100*(1 + 5.0/100)**7)
```

Test this:

```
write(100*(1+5,0/100)^7)
```

- + This make sense for us but not for Python.
- + Output:
- +

```
>>> write(100*(1+5,0/100)^7)
```
- + Traceback (most recent call last):
- + File "<stdin>", line 1, in <module>
- + NameError: name 'write' is not defined

Variables and Variable Types

```
primary = 100  
r = 5.0  
n=7  
amount = primary * (1+r/100)**n  
print(amount)
```


Comment your code

```
# program for computing the growth of  
# money deposited in a bank  
primary = 100 #initial amount  
r = 5.0          #interest rate  
n = 7           #the number of years  
amount = primary * (1+r/100)**n  
print(amount)
```

All variables have types

- + **print(type(hello))** #<class 'str'>
- + **print(type(r))** # <class 'float'>
- + **print(type(primary))** # <class 'float'>
- + **print(type(n))** # <class 'int'>

Formatting Text Output

- + **print**(primary,final_amount)
- + 100 140.71004226562505
- + Target: After 7 years, 100 EUR has grown to xxx EUR.

Python text formatting

```
print(f"After {n} years, 100 EUR has grown to {amount} EUR.")
```

After 7 years, 100 EUR has grown to 140.71004226562505 EUR.

Note: Do we need that decimal cases?

String format

```
t = 1.234567
```

```
print(f"Default output gives t = {t}.")
```

Default output gives t = 1.234567

```
print(f"We can set the precision: t = {t:.2}.")
```

We can set the precision: t = 1.2.

```
print(f"Or control the number of decimals: t = {t:.2f}.")
```

Or control the number of decimals: t = 1.23.

String format

print(f"We may set the space used for the output: t = {t:8.2f}.")

We may **set** the space used **for** the output: t = 1.23

How to interact with user?

- + So far, all the values we have assigned to variables have been written directly into our programs.
- + If we want a different value of a variable, we need to edit the code and rerun the program. Of course, this is not how we are used to interacting with computer programs.
- + Usually, a program will receive some input from users, most often through a graphical user interface (GUI).

Input

+ We need to request three variables to the user.

P is the initial deposit (the *principal*),

r is the interest rate,

n is the number of years

Input

```
P = input("Please enter initial deposit :\n")
```

```
r = input("Please enter yearly interest rate :\n")
```

```
n = input("Please enter number of years :\n")
```

Input Type

```
P = float(input("Please enter initial deposit :\n"))  
r = float(input("Please enter yearly interest rate :\n"))  
n = float(input("Please enter number of years :\n"))
```

When Your Input Might Raise an Exception

```
>>> P = float(input("Please enter initial deposit :\n"))
Please enter initial deposit :
1
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: could not convert string to float: '1'
>>> █
```

```
while True:
```

```
    try:
```

```
        P = float(input("Please enter initial deposit :\n"))
```

```
        r = float(input("Please enter yearly interest rate :\n"))
```

```
        n = float(input("Please enter number of years :\n"))
```

```
    except ValueError:
```

```
        print("Sorry, I didn't understand that.")
```

```
        break
```

```
    else:
```

```
        amount = P * (1+r/100)**n
```

```
        break
```

```
print(f"After {n} years, 100 EUR has grown to {amount:.2f} EUR.")
```