# Getting started with coding: The Basics of Python and Beyond

Vasilis Ntranos
UCSF

BMS 225A — Lecture 02 — October 8, 2025

# Today’s session

Week 2: Lecture 2

- Python Basics
  - Getting started with coding

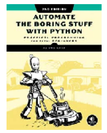
- Intro to pandas
  - Overview, prep for tomorrow’s workshop
  - HW: Extended practice notebook

- Working with AI code companions
  - How to use it as a learning tool

# Python Basics

Overview of the **core functionality** and programming concepts

- Operations, expressions and data types
- Storing values in variables
- Flow control (if/else, for loops)
- Functions
- Lists, Sets and Dictionaries
- Useful **Packages** and Data Structures




**Automate the Boring Stuff with Python**

By Al Sweigart https://automatetheboringstuff.com/

**Important to practice at home:**

Basic Python notebook

https://colab.research.google.com/gist/vasilisNt/438d6e8c1b187800e56f219e211e0950/python-basics-bms225a_2025.ipynb

# **Python Basics** | Operations, expressions and data types

Just like in math, Python can handle basic arithmetic operations

```
2+3
```

```
5
```

```
3*4-1
```

```
11
```

```
3*(4-1)
```

```
9
```

```
2.5 * ( (0.5+1) * 2 / (2-0.5) )
```

```
5.0
```

```
'Hello' + ', BMS! ' + '👋'
```

```
'Hello, BMS! 👋'
```

common data types:

| Data type | Examples |
|---|---|
| Integers | `-2, -1, 0, 1, 2, 3, 4, 5` |
| Floating-point numbers | `-1.25, -1.0, -0.5, 0.0, 0.5, 1.0, 1.25` |
| Strings | `'a', 'aa', 'aaa', 'Hello!', '11 cats'` |

# **Python Basics** | Operations, expressions and data types

Just like in math, Python can handle basic arithmetic operations

```
'Hello, '*10 + 'BMS!'
```

```
'Hello, Hello, Hello, Hello, Hello, Hello, Hello, Hello, Hello, Hello, BMS!'
```

```
'Hello' - 'o'
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-26-9793da777727> in <cell line: 1>()
----> 1 'Hell' - 'o'

TypeError: unsupported operand type(s) for -: 'str' and 'str'
```

```
'Hello' + 2
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-43-b3173ca6e2b4> in <cell line: 1>()
----> 1 'Hello' + 2

TypeError: can only concatenate str (not "int") to str
```

```
'Hello, '*2.5
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-44-4eeba1506da1> in <cell line: 1>()
----> 1 'Hello, '*2.5

TypeError: can't multiply sequence by non-int of type 'float'
```

# **Python Basics** | Storing values in variables

We can create variables to store values, results etc. and use them later in our script

```
gene_name = 'KRAS'
```

variable name

assignment operator

value

Computer Memory
(the box analogy)

# **Python Basics** | Storing values in variables

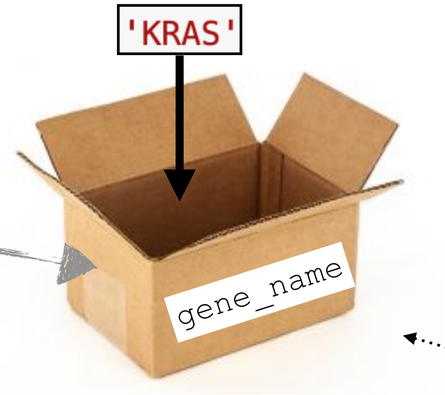
We can create variables to store values, results etc. and use them later in our script

```
gene_name = 'KRAS'
```

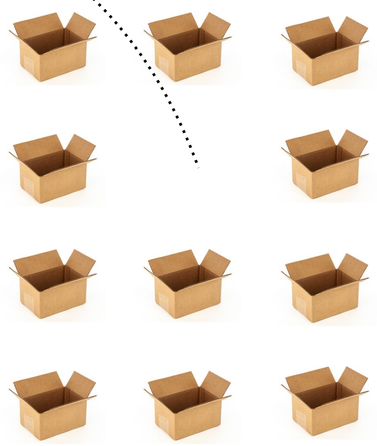
variable name

assignment operator

value




Computer Memory
(the box analogy)



```
print(gene_name)
```

```
KRAS
```

Note: variable names are accessed without quotes:

```
print('gene_name')
```

```
gene_name
```

# **Python Basics** | Storing values in variables

- Remember to use **informative names** for your variables!
- Variable naming restrictions:
  - It can be only one word with no spaces.
  - It can use only letters, numbers, and the underscore (_) character.
  - It can't begin with a number.


### Example: Valid and Invalid Variable Names

| Valid variable names | Invalid variable names |
|---|---|
| `current_balance` | `current-balance` (hyphens are not allowed) |
| `currentBalance` | `current balance` (spaces are not allowed) |
| `account4` | `4account` (can't begin with a number) |
| `_42` | `42` (can't begin with a number) |
| `TOTAL_SUM` | `TOTAL_$UM` (special characters like `$` are not allowed) |
| `hello` | `'hello'` (special characters like `'` are not allowed) |

# **Python Basics** | Storing values in variables

- Note that you can **overwrite** variables

The notebook environment remembers what happened in previous cells

This is a cell →

```
[1] gene_name = 'KRAS'
```

0s

This is another cell →
(execution order in brackets)

```
[2] x = 5
    x = x + 2
    # x = x - 2
    gene_name = gene_name+'>'
    x = gene_name*x
    print(x)
```

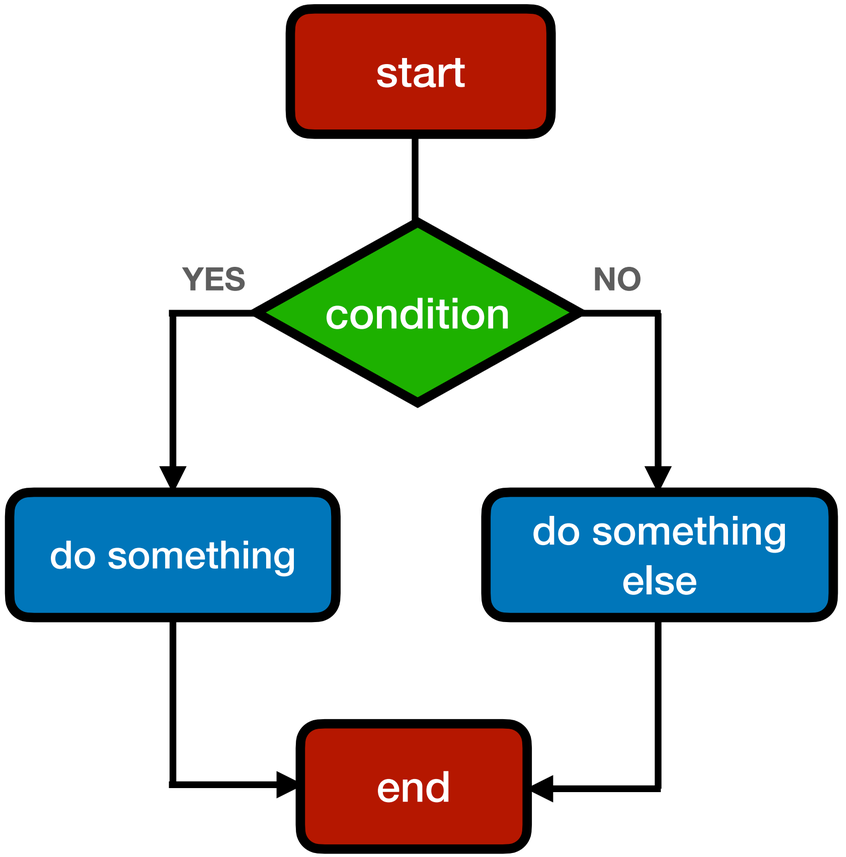
0s

← all lines of code in the cell are executed sequentially (one after the other)

```
KRAS>KRAS>KRAS>KRAS>KRAS>KRAS>KRAS>
```

# Python Basics | Flow control

- Flow statements control which **commands** are executed under which **conditions** so that we can make our scripts more adaptive/general
- A flow statement is a logical expression, asking a question with a yes or no answer (aka Boolean, binary, True/False)
- E.g., x>0, user_name==‘Vasilis’,  etc.
- Based on how expressions evaluate, our script can decide to skip commands, repeat them, or choose one of several commands to run



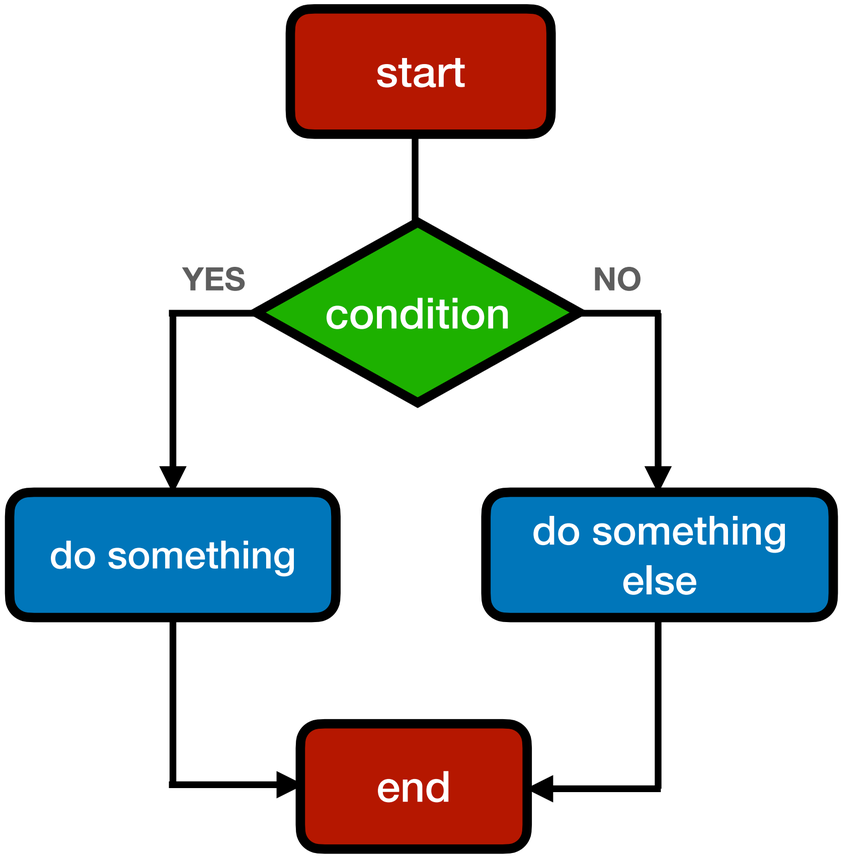

**Comparison Operators**

| Operator | Meaning |
|---|---|
| == | Equal to |
| != | Not equal to |
| < | Less than |
| > | Greater than |
| <= | Less than or equal to |
| >= | Greater than or equal to |

# Python Basics | Flow control

- If / else

```
if gene_expression<=0:
  print('the gene is not expressed')
```



# **Python Basics** | Flow control

- If / else

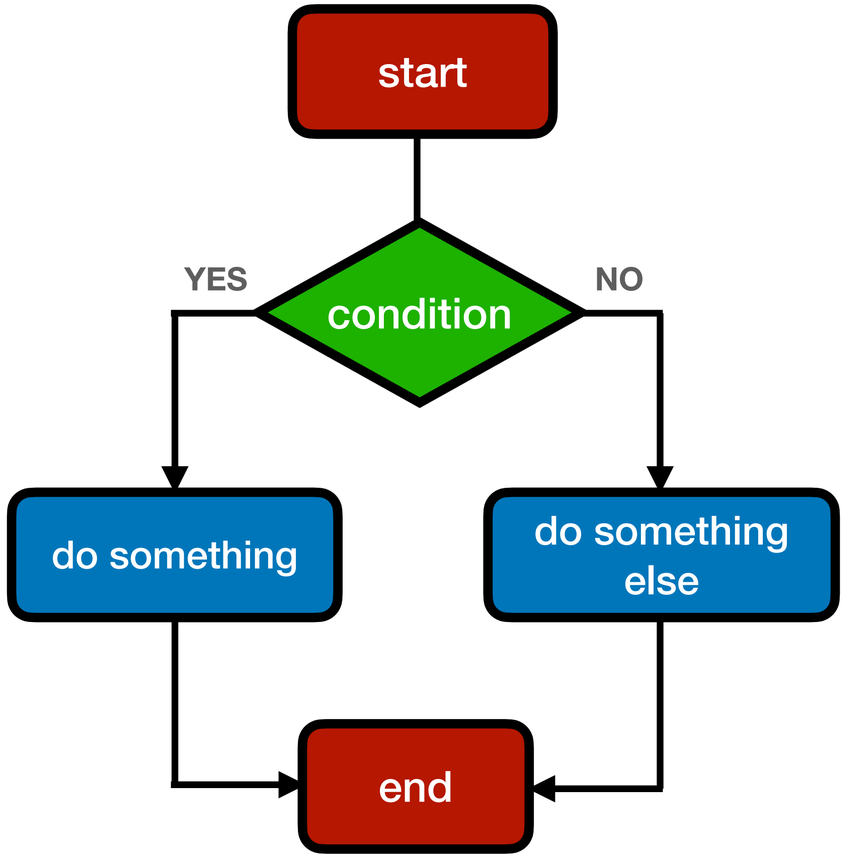
```
gene_expression = 54
```

0s

```
if gene_expression<=0:
  print('the gene is not expressed')
else:
  if gene_expression>0:
    print('the gene is expressed')
  if gene_expression>50:
    print('the gene is highly expressed')
```

0s

```
the gene is expressed
the gene is highly expressed
```



# Python Basics | Flow control

- **while** loop

repetitive code:

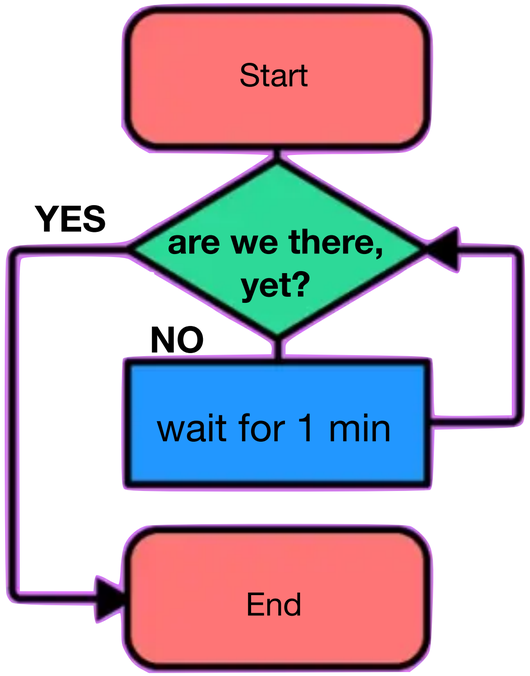
```
number = 0

# iteration 1
print(number)
number = number +1

# iteration 2
print(number)
number = number +1

# iteration 3
print(number)
number = number +1
```

```
0
1
2
```



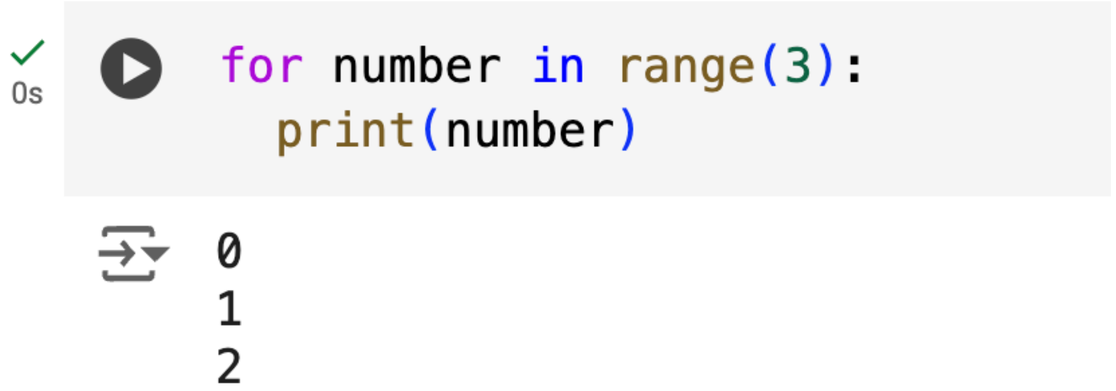

Using the **while** loop:

```
number = 0  # Start at 0
while number < 3:  # Run until number reaches 3
    print(number)
    number = number +1  # Increment number by 1
```

```
0
1
2
```

# **Python Basics** | Flow control

- **for** loop




**for** a variable called **number** taking values **in** the **range** from 0 to **3** (non inclusive), do the following**:**

-> some commands that will be repeated 3 times

# **Python Basics** | Functions

- **Functions** are used to logically break our code into simpler parts which become easy to maintain and understand
- Most frequently used to run a set of tasks multiple times at different locations in the code
- If you have to do it twice, make it a function!
- Sometimes the only thing we need to know is the function's input and output

the **name** of the function

**input** variables and parameters

**definition** keyword

```
[ ]  ### Define the function:
     def add_and_scale(x, y, scale_param = 1):
         result = ( x + y ) * scale_param
         return result
```

**code** executed by the function

output / **return** value

the **print( )** function:

```
builtin_print_impl(PyObject *module, PyObject *args, PyObject *sep,
                   PyObject *end, PyObject *file, int flush)
/*[clinic end generated code: output=3cfc0940f5bc237b input=c143c575d24fe665]*/
{
    int i, err;

    if (file == Py_None) {
        PyThreadState *tstate = _PyThreadState_GET();
        file = _PySys_GetAttr(tstate, &_Py_ID(stdout));
        if (file == NULL) {
            PyErr_SetString(PyExc_RuntimeError, "lost sys.stdout");
            return NULL;
        }

        /* sys.stdout may be None when FILE* stdout isn't connected */
        if (file == Py_None) {
            Py_RETURN_NONE;
        }
    }

    if (sep == Py_None) {
        sep = NULL;
    }
    else if (sep && !PyUnicode_Check(sep)) {
        PyErr_Format(PyExc_TypeError,
                     "sep must be None or a string, not %.200s",
                     Py_TYPE(sep)->tp_name);
        return NULL;
    }
    if (end == Py_None) {
        end = NULL;
    }
    else if (end && !PyUnicode_Check(end)) {
        PyErr_Format(PyExc_TypeError,
                     "end must be None or a string, not %.200s",
                     Py_TYPE(end)->tp_name);
        return NULL;
    }

    for (i = 0; i < PyTuple_GET_SIZE(args); i++) {
        if (i > 0) {
            if (sep == NULL) {
                err = PyFile_WriteString(" ", file);
            }
            else {
                err = PyFile_WriteObject(sep, file, Py_PRINT_RAW);
            }
            if (err) {
                return NULL;
            }
        }
        err = PyFile_WriteObject(PyTuple_GET_ITEM(args, i), file, Py_PRINT_RAW);
        if (err) {
            return NULL;
        }
    }

    if (end == NULL) {
        err = PyFile_WriteString("\n", file);
    }
    else {
        err = PyFile_WriteObject(end, file, Py_PRINT_RAW);
    }
    if (err) {
        return NULL;
    }

    if (flush) {
        if (_PyFile_Flush(file) < 0) {
            return NULL;
        }
    }

    Py_RETURN_NONE;
}
```

# Python Basics | Functions

```
[1] ### Function definition with a docstring explaining what the function does
    def add_and_scale(x, y, scale_param=1):
        """
        Adds two numbers and scales the result by a specified parameter.

        Parameters:
        x (int or float): The first number to add.
        y (int or float): The second number to add.
        scale_param (int or float, optional): The scaling factor to multiply the sum by.
                                              Default is 1.

        Returns:
        int or float: The result of adding x and y, then multiplying by scale_param.

        Example:
        >>> add_and_scale(3, 5)
        8
        >>> add_and_scale(3, 5, 2)
        16
        """
        # Add the two input numbers x and y
        result = (x + y)

        # Scale the sum by the scale_param
        result = result * scale_param

        # Return the final result
        return result
```

```
[2] add_and_scale?
```

Help ×

```
Signature: add_and_scale(x, y,
scale_param=1)
Docstring:
Adds two numbers and scales the result by a
specified parameter.

Parameters:
x (int or float): The first number to add.
y (int or float): The second number to add.
scale_param (int or float, optional): The
scaling factor to multiply the sum by.

Default is 1.

Returns:
int or float: The result of adding x and y,
then multiplying by scale_param.

Example:
>>> add_and_scale(3, 5)
8
>>> add_and_scale(3, 5, 2)
16
File:      /content/<ipython-input-1-
7d807062dcf4>
Type:      function
```

# **Python Basics** | Lists, Sets and Dictionaries

- Python provides basic data structures for us to work with

- **List:** fruit_list = [ 🥝, 🍋, 🍊, 🍓, 🍋, 🍋 ]

```
fruit_list[2]
'🍊'
```

- **Set** fruit_set = { 🍋 🥝 🍓 🍊 }

```
fruit_set.intersection({'🥝', '🍎'})
{'🥝'}
```

- **Dictionary:**

fruit_to_color = { 🥝 : green, 🍓 : red, 🍋 : yellow, 🍊 : orange }

```
fruit_to_color['🍋']
'yellow'
```

# **Python Basics** | Almost everything we discussed in an example

- Example: *Finding the reverse complement of a dna sequence*

```python
def complement(dna_sequence):
    # Dictionary to map each nucleotide to its complement
    complement = {'A': 'T', 'T': 'A', 'C': 'G', 'G': 'C'}

    # Generate the complement sequence
    complement_sequence = ''
    for nucleotide in dna_sequence:
        complement_nucleotide = complement[nucleotide]
        complement_sequence += complement_nucleotide

    return complement_sequence
```

```python
### Define the function:
def reverse(dna_sequence):
    # Generate the reverse sequence
    reverse_sequence = ''
    for nucleotide in dna_sequence:
        reverse_sequence = nucleotide + reverse_sequence

    return reverse_sequence
```

```python
### Call the complement function:
seq = "ATGTCAC"
result = complement(seq)
print(result)
```

```
TACAGTG
```

```python
### Call the reverese function:
seq = "ATGTCAC"
result = reverse(seq)
print(result)
```

```
CACTGTA
```

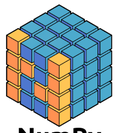
```python
# Get the reverse complement of the sequence
seq = "ATGTCAC"
result = complement( reverse(seq) )
print(result)
```

```
GTGACAT
```

visualize execution in Basic Python Notebook: https://colab.research.google.com/gist/vasilisNt/438d6e8c1b187800e56f219e211e0950/python-basics-bms225a_2025.ipynb

# **Python Basics** | Useful Packages and Data Structures

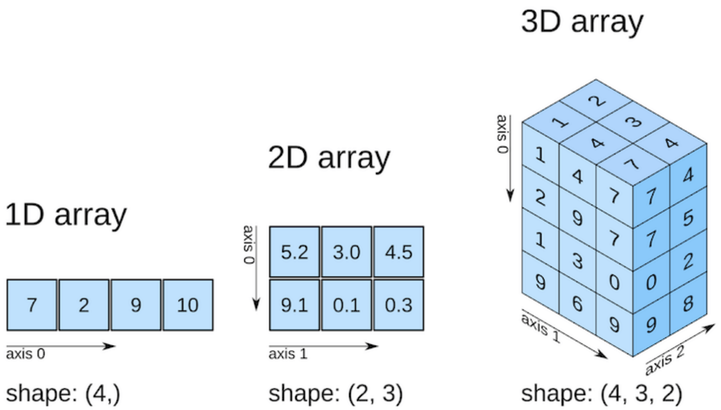
(we will focus on **Pandas** in the second half of the lecture)



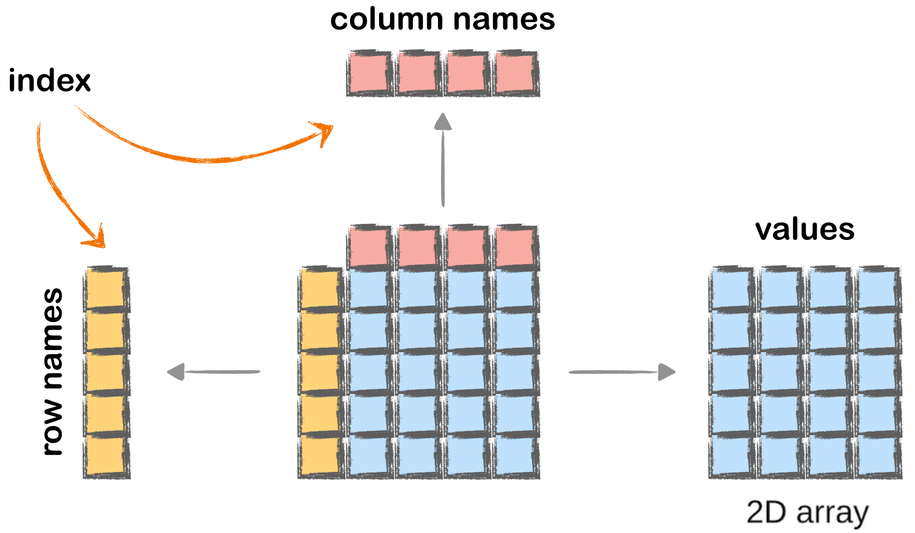

- NumPy:
  A powerful library for numerical computing in Python. It provides support for **Arrays** and matrices, along with a collection of mathematical functions to operate on these data structures.

pandas

- Pandas:
  A popular data analysis library. It provides data structures like **DataFrames** that allow users to work efficiently with labeled, structured datasets, making it easy to explore and analyze data.




**NumPy Arrays**




**Pandas DataFrame**

# **Python Basics** | Useful Packages and Data Structures

- Pandas and NumPy provide the basic data structures needed to work with other (higher-level) packages, like:


- Seaborn:
  Data visualization library built on top of Matplotlib. It provides an easy-to-use interface for creating aesthetically pleasing and informative statistical graphics, making it especially useful for exploratory data analysis.

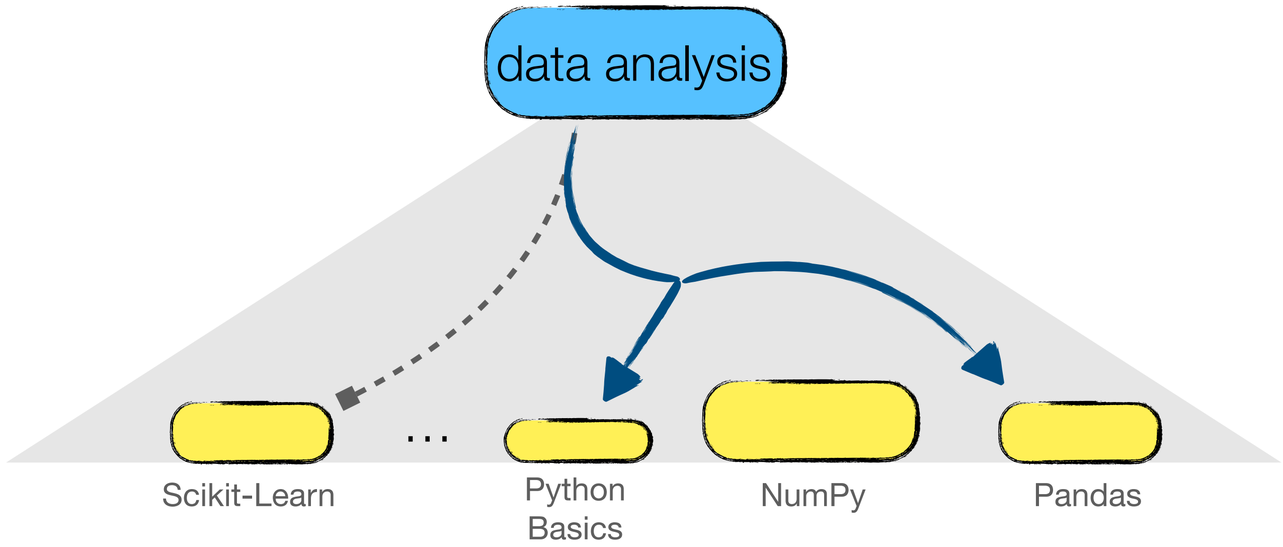

- Scikit-Learn (sklearn):
  Machine learning library that provides a wide variety of tools for data mining and analysis. It includes algorithms for classification, regression, clustering, and dimensionality reduction, all accessible via a clean, consistent interface.


- Scanpy:
  Python library specifically designed for analyzing single-cell gene expression data. Under the hood, **scanpy uses all of the above packages** to provide high-level functionality for single cell data analysis and visualization.

# **Python Basics** | Practice, practice, practice!

- I know this is A LOT of information to digest in a week — let alone in a single lecture.
  Remember that this is to lay some foundations and build resources that we can come back to.

  - And it is important to practice these skills on your own as much as possible!
    —> Highly recommended to go over the **practice sets** at home



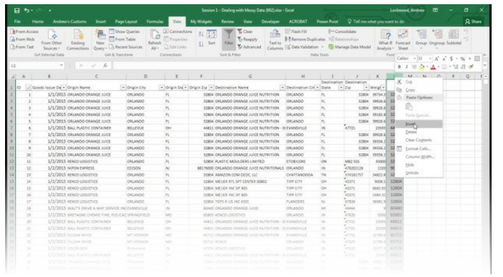

Colab Notebook with extended materials and code covered in the Python Basics available at:

https://colab.research.google.com/gist/vasilisNt/438d6e8c1b187800e56f219e211e0950/python-basics-bms225a_2025.ipynb

# Break

# Intro to pandas

- Goal for today: establish a mental model about pandas before the workshop
  (We will use pandas in the workshop to prepare the data for generating Fig2e in Mennillo et al.)
- Python library that makes working with **table shaped data** much easier
  - Think spreadsheets, but reproducible & scalable
    - Manual clicks and edits are **slow** and **error-prone** when data size grows and steps must be repeated



- Reproducible code:

```
In [8]: flights = pd.read_csv("flights.csv")
        flights = flights[['carrier', 'dep_delay', 'arr_delay']]
        flights.groupby("carrier").mean()
```

Out[8]:

| carrier | dep_delay | arr_delay |
|---|---|---|
| AA | 8.586016 | 0.364291 |
| AS | 5.804775 | -9.930889 |
| DL | 9.264505 | 1.644341 |
| UA | 12.106073 | 3.558011 |
| US | 3.782418 | 2.129595 |

```
load the data
select relevant columns
calculate the average
save the results
```

# Intro to pandas | Our tiny dataset (used throughout)

| order id | item | price | payment |
|---|---|---|---|
| 101 | Latte | 4.5 | Card |
| 102 | Muffin | 3.25 | Cash |
| 103 | Latte | 4.5 | Card |

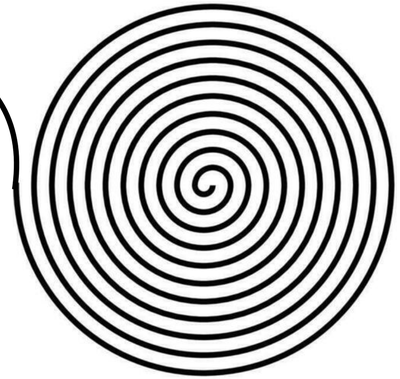
# Intro to pandas

- In python's data analysis ecosystem **pandas sits in the middle**: it reads raw data, helps you clean and reshape, and hands off to visualization or analysis tools.

RAW DATA

pandas

Data analysis
Visualization
Machine Learning



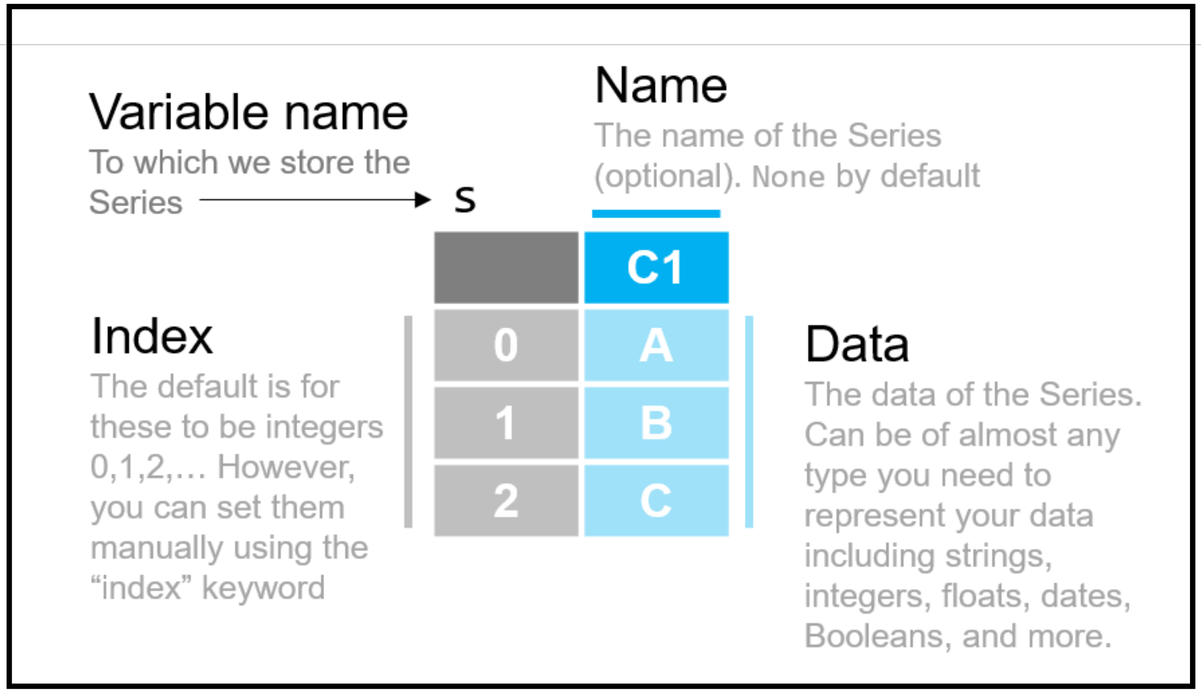
- A common misconception is that **data science** is mostly modeling. In practice, a large share is **thoughtful data preparation**.

# Intro to pandas

- Two Core Data Structures: **Series & DataFrame**

## Series

A single labeled column with an index
(you can think of index as row labels)




## DataFrame

A table made of **multiple Series** that share the **same index**

| INDEX | SERIES 1 | SERIES 2 | SERIES 3 |
|---|---|---|---|
| 0 | A | 1 | [1, 2] |
| 1 | B | 2 | A |
| 2 | C | 3 | 1 |
| 3 | D | 4 | (4, 5) |
| 4 | E | 5 | {"a": 1} |
| 5 | F | 6 | 6 |

# Intro to pandas

- Two Core Data Structures: **Series & DataFrame**

| order id | item | price | payment |
|---|---|---|---|
| 101 | Latte | 4.5 | Card |
| 102 | Muffin | 3.25 | Cash |
| 103 | Latte | 4.5 | Card |
| 104 | Tea | 3.0 | Card |
| 105 | Sandwich | 7.25 | Cash |
| 106 | Latte | 4.5 | Card |

# Intro to pandas

- Two Core Data Structures: **Series & DataFrame**

index ↓

| order id | **item** | **price** | **payment** |
|---|---|---|---|
| 101 | Latte | 4.5 | Card |
| 102 | Muffin | 3.25 | Cash |
| 103 | Latte | 4.5 | Card |
| 104 | Tea | 3.0 | Card |
| 105 | Sandwich | 7.25 | Cash |
| 106 | Latte | 4.5 | Card |

# Intro to pandas

- Two Core Data Structures: **Series & DataFrame**

index ↓

| | order id | item | price | payment |
|---|---|---|---|---|
| 0 | 101 | Latte | 4.5 | Card |
| 1 | 102 | Muffin | 3.25 | Cash |
| 2 | 103 | Latte | 4.5 | Card |
| 3 | 104 | Tea | 3.0 | Card |
| 4 | 105 | Sandwich | 7.25 | Cash |
| 5 | 106 | Latte | 4.5 | Card |

← **column names**

↑ **Series** ↑ **Series** ↑ **Series** ↑ **Series**

# Intro to pandas

- Loading DataFrames

```
[1]
✓ 0s
import pandas as pd

[2]
✓ 0s
df = pd.read_csv('cafe_orders.csv')

[3]
✓ 0s
df
```

| | order_id | item | price | payment |
|---|---|---|---|---|
| 0 | 101 | Latte | 4.50 | Card |
| 1 | 102 | Muffin | 3.25 | Cash |
| 2 | 103 | Latte | 4.50 | Card |
| 3 | 104 | Tea | 3.00 | Card |
| 4 | 105 | Sandwich | 7.25 | Cash |
| 5 | 106 | Latte | 4.50 | Card |

- Selecting columns (e.g., `['item','price']` )

```
df[['item','price']]
```

| | item | price |
|---|---|---|
| 0 | Latte | 4.50 |
| 1 | Muffin | 3.25 |
| 2 | Latte | 4.50 |
| 3 | Tea | 3.00 |
| 4 | Sandwich | 7.25 |
| 5 | Latte | 4.50 |

```
df[['item']]
```

| | item |
|---|---|
| 0 | Latte |
| 1 | Muffin |
| 2 | Latte |
| 3 | Tea |
| 4 | Sandwich |
| 5 | Latte |

Still a DataFrame!

```
df['item']
```

| | item |
|---|---|
| 0 | Latte |
| 1 | Muffin |
| 2 | Latte |
| 3 | Tea |
| 4 | Sandwich |
| 5 | Latte |

**dtype:** object

Series

# Intro to pandas | Inside a pandas DataFrame

DataFrame:

| | patient_id | age | gender | height | weight | smoke | alco | active |
|---|---|---|---|---|---|---|---|---|
| 0 | 43005 | 15375 | Male | 169 | 68.0 | False | False | False |
| 1 | 76878 | 21712 | Female | 182 | 91.0 | True | False | True |
| 2 | 68983 | 22699 | Male | 166 | 72.0 | False | False | False |
| 3 | 61210 | 21163 | Female | 172 | 65.0 | True | True | True |
| 4 | 68948 | 19650 | Male | 168 | 72.0 | False | False | True |

stored in a variable called patient_data

patient_data

**variables (attributes)**

values

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 43005 | 15375 | Male | 169 | 68.0 | False | False | False |
| 76878 | 21712 | Female | 182 | 91.0 | True | False | True |
| 68983 | 22699 | Male | 166 | 72.0 | False | False | False |
| 61210 | 21163 | Female | 172 | 65.0 | True | True | True |
| 68948 | 19650 | Male | 168 | 72.0 | False | False | True |

index

| |
|---|
| 0 |
| 1 |
| 2 |
| 3 |
| 4 |

columns

| patient_id | age | gender | height | weight | smoke | alco | active |
|---|---|---|---|---|---|---|---|

**functions (methods)**

reset_index( ) rename( ) drop_duplicates( )

sum( ) mean( ) pivot_table( ) …

access everything with a .

patient_data.columns

patient_data.values

patient_data.index

patient_data.drop_duplicates( )

patient_data.reset_index( )

# Intro to pandas

- Applying functions

```
df['price'].sum()
```

```
27.0
```

```
df['price'].describe()
```

| | price |
|---|---|
| count | 6.00000 |
| mean | 4.50000 |
| std | 1.50831 |
| min | 3.00000 |
| 25% | 3.56250 |
| 50% | 4.50000 |
| 75% | 4.50000 |
| max | 7.25000 |

```
df.head(3)
```

| | order_id | item | price | payment |
|---|---|---|---|---|
| 0 | 101 | Latte | 4.50 | Card |
| 1 | 102 | Muffin | 3.25 | Cash |
| 2 | 103 | Latte | 4.50 | Card |

```
df.sort_values('price')
```

| | order_id | item | price | payment |
|---|---|---|---|---|
| 3 | 104 | Tea | 3.00 | Card |
| 1 | 102 | Muffin | 3.25 | Cash |
| 0 | 101 | Latte | 4.50 | Card |
| 2 | 103 | Latte | 4.50 | Card |
| 5 | 106 | Latte | 4.50 | Card |
| 4 | 105 | Sandwich | 7.25 | Cash |

```
df.set_index('order_id')
```

| order_id | item | price | payment |
|---|---|---|---|
| 101 | Latte | 4.50 | Card |
| 102 | Muffin | 3.25 | Cash |
| 103 | Latte | 4.50 | Card |
| 104 | Tea | 3.00 | Card |
| 105 | Sandwich | 7.25 | Cash |
| 106 | Latte | 4.50 | Card |

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6 entries, 0 to 5
Data columns (total 4 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   order_id  6 non-null      int64
 1   item      6 non-null      object
 2   price     6 non-null      float64
 3   payment   6 non-null      object
dtypes: float64(1), int64(1), object(2)
memory usage: 324.0+ bytes
```

# Intro to pandas

- Selecting rows (boolean masking)

```
df['item']
```

| | item |
|---|---|
| 0 | Latte |
| 1 | Muffin |
| 2 | Latte |
| 3 | Tea |
| 4 | Sandwich |
| 5 | Latte |

**dtype:** object

```
df['item'] == 'Latte'
```

| | item |
|---|---|
| 0 | True |
| 1 | False |
| 2 | True |
| 3 | False |
| 4 | False |
| 5 | True |

**dtype:** bool

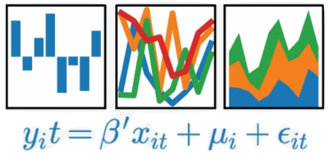
```
df[df['item'] == 'Latte']
```

| | order_id | item | price | payment |
|---|---|---|---|---|
| 0 | 101 | Latte | 4.50 | Card |
| 1 | 102 | Muffin | 3.25 | Cash |
| 2 | 103 | Latte | 4.50 | Card |
| 3 | 104 | Tea | 3.00 | Card |
| 4 | 105 | Sandwich | 7.25 | Cash |
| 5 | 106 | Latte | 4.50 | Card |

| | order_id | item | price | payment |
|---|---|---|---|---|
| 0 | 101 | Latte | 4.5 | Card |
| 2 | 103 | Latte | 4.5 | Card |
| 5 | 106 | Latte | 4.5 | Card |

# Intro to pandas | Resources for Getting Started

(We will use pandas in the workshop to prepare the data for generating Fig2e in Mennillo et al.)




Practice Set: https://gist.github.com/vasilisNt/814f5aabdfcb8d23f46cecf149f9397a

Open in Colab

In [ ]:
```
# importing pandas
import pandas as pd
```

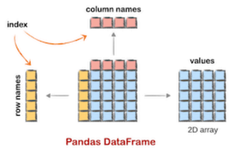
### 1. Loading a csv file with Pandas

We'll start by reading the `patient_data.csv` file. This dataset contains the following patient information (adapted from: `https://www.kaggle.com/datasets/thedevastator/exploring-risk-factors-for-cardiovascular-diseas/data`):

- `patient_id` : unique id of the patient
- `age` : the age of the patient (in days!)
- `gender` : Male/Female
- `height` : measured height in cm
- `weight` : patient weight in kg
- `smoke` : smoking or not
- `alco` : drinks alcohol or not
- `active` : physically active or not

> Reading and Writing Data Reading data means bringing it into Python from a file, such as CSV. Pandas provides a very convenient way to do this using the `read_csv()` function. After making changes to our data, we often want to save it back to a file. We can use the `to_csv()` function to write a DataFrame to a CSV file.

In [ ]:
```
# Reading the patient data from the CSV file into a dataframe
patient_data = pd.read_csv('patient_data.csv')
```

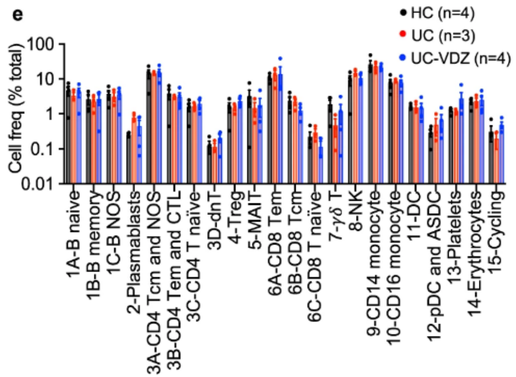



- read and write data
- indexing, subsetting, creating columns
- groupby, pivot, melt
- combine data (join, merge)
- manipulate text data

| | |
|---|---|
| intro tutorials | https://pandas.pydata.org/docs/getting_started/intro_tutorials/index.html |
| user guide | https://pandas.pydata.org/docs/user_guide/index.html |
| comparison with R | https://pandas.pydata.org/docs/getting_started/comparison/comparison_with_r.html |

# Preparing for Thursday's workshop

(We will use pandas in the workshop to prepare the data for generating Fig2e in Mennillo et al.)




- No installations / data needed for the workshop (everything will be downloaded in Colab)

…but please go over the Pandas practice notebook once before the workshop:

https://gist.github.com/vasilisNt/814f5aabdfcb8d23f46cecf149f9397a

1) Download the CSV files from the CLE
2) Upload the files to your Colab Notebook
3) Run each cell one-by-one and review what it does
   (at the depth you are most comfortable with)

# Preparing for next week (single cell RNA seq)

- Watch the **Pandas Basics** video (Parts 2-8) ~3 hours
  - link: https://www.youtube.com/playlist?list=PL-osiE80TeTsWmV9i9c58mdDCSskIFdDS

If you haven't done so already:

- Watch the **python basics** video (first 3 hours)
  - link: https://www.youtube.com/watch?v=rfscVS0vtbw
    (ok to skip first 10min, no need to install python)

# Working with AI code companions

- AI tools that know how to code: Gemini, Github copilot, chatGPT, Claude
  - model size matters a lot
- Today: Useful *prompts*
  - how to use AI as a learning tool
  - ask to generate better code
- Week 4: Using AI to create better plots

# Working with AI code companions

- Useful prompts

**(help me understand what this code is doing)**

```
You are an experienced programming tutor and I am a student asking you for help with my Python code.
- Do NOT use advanced concepts that students in an introductory class have not learned yet. Instead, use
concepts that are taught in introductory-level classes and beginner-level programming tutorials. Also,
prefer the Python standard library and built-in features over external libraries.

Here is my Python code, which I'm trying to run in a Colab Notebook:

def complement(dna_sequence):
    complement = {'A': 'T', 'T': 'A', 'C': 'G', 'G': 'C'}
    complement_sequence = ''
    for nucleotide in dna_sequence:
        complement_nucleotide = complement[nucleotide]
        complement_sequence += complement_nucleotide

    return complement_sequence

__
```

**Can you please explain what is happening in the for loop?**

# Working with AI code companions

**(help me figure out the answer myself)**

- Useful prompts

```
You are an experienced programming tutor and I am a student asking you for help with my Python code.
- Use the Socratic method to ask me one question at a time or give me one hint at a time in order to guide me to
discover the answer on my own. Do NOT directly give me the answer. Even if I give up and ask you for the answer,
do not give me the answer. Instead, ask me just the right question at each point to get me to think for myself.
- Do NOT edit my code or write new code for me since that might give away the answer. Instead, give me hints of
where to look in my existing code for where the problem might be. You can also print out specific parts of my code
to point me in the right direction.
- Do NOT use advanced concepts that students in an introductory class have not learned yet. Instead, use concepts
that are taught in introductory-level classes and beginner-level programming tutorials. Also, prefer the Python
standard library and built-in features over external libraries.

Here is my Python code, which I'm trying to run in a Colab Notebook:

def complement(dna_sequence):
    complement = {'A': 'T', 'T': 'A', 'C': 'G', 'G': 'C'}
    complement_sequence = ''
    for nucleotide in 'dna_sequence':
        complement_nucleotide = complement[nucleotide]
        complement_sequence = complement_nucleotide

    return complement_sequence

__

I am getting an error when I try to call this function
```

# Working with AI code companions

(answer my question in a way I can understand)

- Useful prompts

```
- Answer my question as directly as possible. Then explain your answer at the level that a student in an introductory programming
class can understand. Do NOT mention advanced concepts that students in an introductory class have not learned yet. Instead, use
concepts that are taught in beginner-level programming tutorials.
- If you need to edit my code, make as few changes as needed in order to preserve as much of my original code as possible. Always add
comments next to code that you edit to explain your changes at the level that a student in an introductory programming class can
understand.
- Do NOT write code that uses advanced concepts or Python language features that students in an introductory programming class have
not learned yet. Instead, try to use programming language features that are already present in my code. Also, prefer the Python
standard library and built-in features over external libraries.

Here is my Python code, which I'm trying to run in a Colab Notebook:

def complement(dna_sequence):
    complement = {'A': 'T', 'T': 'A', 'C': 'G', 'G': 'C'}
    complement_sequence = ''
    for nucleotide in 'dna_sequence':
        complement_nucleotide = complement[nucleotide]
        complement_sequence = complement_nucleotide

    return complement_sequence

--
I'm getting the follwing error when I call this function:

---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-7-8ba19fd84490> in <cell line: 1>()
----> 1 complement('ACGT')

<ipython-input-6-f166ed6cc95b> in complement(dna_sequence)
      3     complement_sequence = ''
      4     for nucleotide in 'dna_sequence':
----> 5         complement_nucleotide = complement[nucleotide]
      6         complement_sequence = complement_nucleotide
      7

KeyError: 'd'
--
How can I fix this?
```

# Working with AI code companions

- Useful prompts

**(help me figure out how to code something from scratch)**

- **Let's work on a Python project together!** Help me write code and explain each step at the level that a student in an introductory programming class can understand. Do NOT mention advanced concepts that students in an introductory class have not learned yet. Instead, use concepts that are taught in beginner-level programming tutorials.
- Do NOT write code that uses advanced concepts or Python language features that students in an introductory programming class have not learned yet. If that is impossible, please ask me if I want to switch to expert mode.
- Please take into account that I'm working in a Colab Notebook environment. Ideally, output the code for each cell separately.
- ALWAYS use well documented libraries for your code that can be installed in Colab.
- For any code you write please also include some tests so that I can check that the code is working as expected

__

**I want to create a function that computes the reverse complement of a sequence. Can you help?**