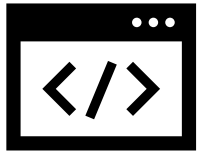
IMPERIAL



BIOE50010 – Programming 2

Computer Lab 3

Binghuan Li Department of Chemical Engineering

Maria Portela Department of Bioengineering

Wenhao Ding Department of Bioengineering

18 October, 2024

Formatting

- **f**-string formatting starts with an **f** before the opening quotation mark
- Each individual variable is enclosed within a pair curly brackets **{}**

Code snippet from `extract_dna2protein.py`

```
TTT   Phe   F Phenylalanine
```

```
data = [['T', 'T', 'T', 'Phe', 'F', 'Phenylalanine'],
        ['T', 'T', 'C', 'Phe', 'F', 'Phenylalanine']]

print(f'{data[0][0]}{data[0][1]}{data[0][2]}\t{data[0][3]}\t{data[0][4]}\t{dat
a[0][5]}')
```

- Alternatively, one can use the **`format`** method

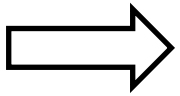
```
print('{}{}{}\t{}\t{}\t{}'.format(data[0][0], data[0][1], data[0][2],
data[0][3], data[0][4], data[0][5])
```

Raw String

- By default, Python treats the backslash (\) as a special character: *e.g.*, `\t`, `\n`

Example

```
myStr = 'Hi\nHello'
print(myStr)
```



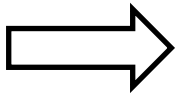
Console

```
>> Hi
Hello
```

- Python **raw string** ('r') treats the backslash as a literal character.

Example

```
myStr = r'Hi\nHello'
print(myStr)
```



Console

```
>> Hi\nHello
```

- …which can be useful with the **open** function

☹

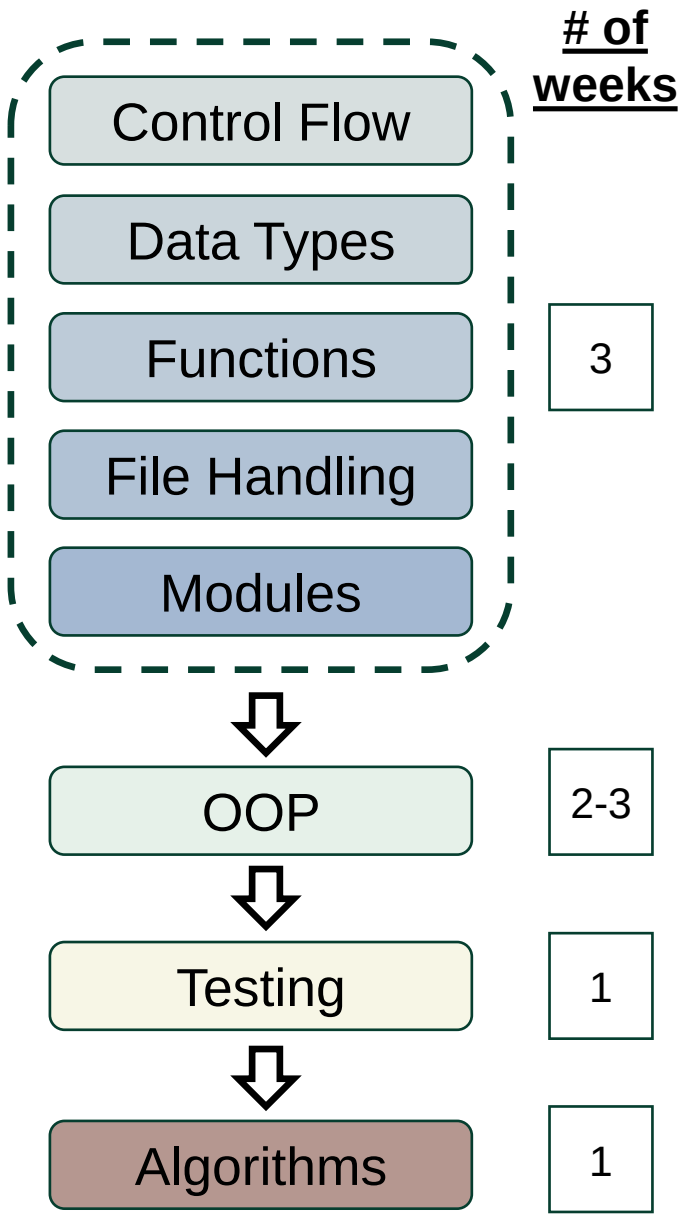
```
f = open('C:\Users\lbing\Desktop\lab2\the_road_not_taken.txt', 'r')
```

☺

```
f = open(r'C:\Users\lbing\Desktop\lab2\the_road_not_taken.txt', 'r')
```

Progress Check

Week 3:
we are here



Questions outside the classroom? ed discussion

Checklist: you should have mastered…

- File I/O: open, read, close
- Loops, recursion: when to terminate reading?
- Function definition and namespaces
- Formatting with `f`-string
- Python build in functions: `count()`, `strip()`, `split()`

General Good Coding Practice

- Code is read much more often than it is written. Code should always be written in a way that promotes readability.
- PEP 8 provides coding style guide for Python programming from the authors' perspectives. Key advice to summarise:
  1. Use **intention-revealing**, **descriptive** names
  2. Adhere to the **proper code layout** (*e.g.*, use consistent 4-space indentation)
  3. Keep **comments**, but good comments do not excuse unclear code

Identifier Type	Example Name	Naming Convention
variables	`playBoard`	Lower Camel Case
functions	`displayBoard`	Lower Camel Case
	`display_board`	Snake Case
classes	`BioengPerson`	Upper Camel Case
constants	`MAX_CAPACITY`	Constant Case

Avoid using names *e.g.*,

- `myList` ← vague
- `data1` ← "noisy"
- `l`, `O` ← 'l' or '1'? 'O' or '0'?

How to Properly Document a Function?

argument annotation: a and b are `float`

return annotation: c is `float`

```
def calculate_pythagoras(a: float, b: float) -> float:
    """
    Calculate the hypotenuse of a right-angled triangle.

    Args:
        a (float): Length of side a.
        b (float): Length of side b.

    Returns:
        float: Length of the hypotenuse.

    Example:
        >>> calculate_pythagoras(3, 4)
        5.0
    """
    c = (a**2 + b**2)**0.5
    return c
```

consistent 4-space indentation

Documentation strings:

- Function description
- Arguments
- Return
- Example usage

Your tasks today

- Three mini tasks on **modular programming** (writing functions)
  - Task 1: Calculate radius and from a pair of polar coordinates

$$\theta = \left(\text{atan}\left(\frac{\text{edge}_y}{\text{edge}_x}\right)/\pi\right) * 180°$$

  - Task 2: Passing an unknown number of arguments into a function
  - Task 3: The Collatz conjecture
  - Task 4: Plotting marks on a user-defined board

To start…

- Read all information and the **sample output** provided in the lab carefully
- Consult the help pages for the string / list methods provided in *Lab 2 slides*
- Study the non-keyword and keyword arguments attached to the slides.

Hint: Non-Keyword and Keyword Args (1/)

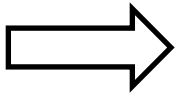
Suppose you are defining a function with arbitrary number of arguments…

- You can use non-keyword arguments (*arg)

Example

```
def call_good_fruits(*fruits):
    for item in fruits:
        print('let us take a', item)

good_fruits('kiwi', 'watermelon', 'durian')
```

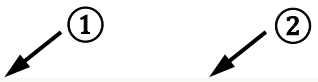


Console

```
let us take a kiwi
let us take a watermelon
let us take a durian
```

Comments

- The asterisk * is known as the unpacking operator.
- All *args are collected and packed into a **tuple** (hence, iterable)
- Positional arguments must come before *args:




```
def call_greeting(greeting, *names)
```

Hint: Non-Keyword and Keyword Args (2/)

- Alternatively, you can use keyword arguments (*kwarg)

Example

```
def call_good_fruits(**fruits):
    for fruit, attribute in kwargs.items():
        print(f"Let us take a {fruit}, which is {attribute}.")

good_fruits(kiwi="green", watermelon="large", durian="spiky")
```

Console

```
Let us take a kiwi, which is green.
Let us take a watermelon, which is large.
Let us take a durian, which is spiky.
```

Comments

- All *kwargs are collected and packed into a **dictionary** ({key}:{value})
- Positional arguments and *args must come before **kwargs.

Hint: Iterations with enumerate and range

- `range(start=0, stop, step=1)` - iterate through a sequence of numbers
- `enumerate(iterable, start=0)` - iterate through an iterable object and keep track of both the index and the number.

Example

```
good_friuts = ['kiwi', 'watermelon', 'durian']

# using range()
for idx in range(0, len(good_friuts)):
    print(f'{idx}\t{good_friuts[idx]}')

# using enumerate()
for idx, fruit in enumerate(good_friuts):
    print(f'{idx}\t{fruit}’)
```

same printout!

Console

```
0       kiwi
1       watermelon
2       durian
```

Questions?

That's it for now.

You can now proceed to the Lab 3 exercises.