

(U4284) Python程式設計

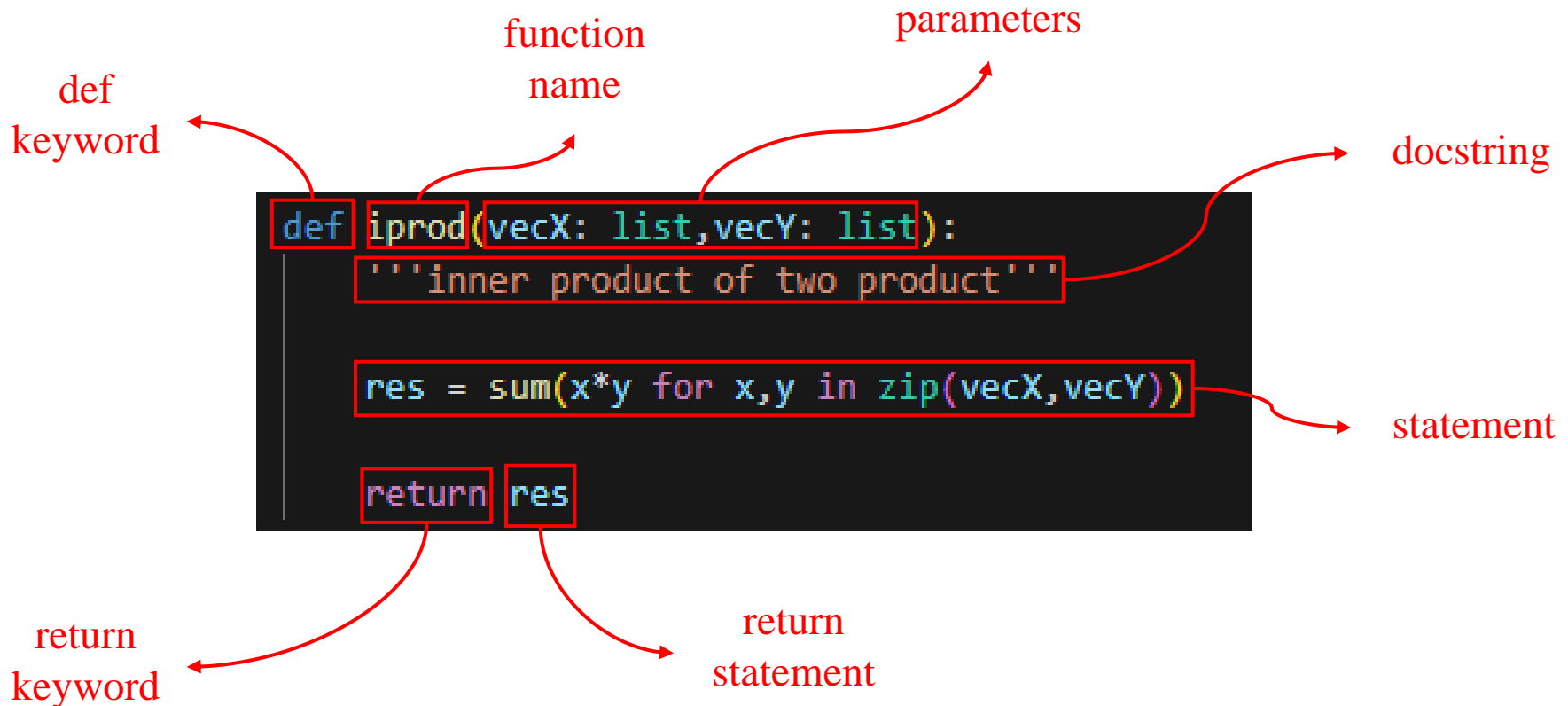
Function, Class

Speaker: 吳淳硯



Function

- There are Two things with a function.
 - *Define* it with zero or more parameters
 - *Call* it and get zero or more results
- To define a Python function, type **def**



Arguments

- Positional Argument
 - Values are copied to their corresponding parameters **in order**.
- Keyword Argument
 - Specify arguments by the names of their corresponding parameters, even **in a different order** from their definition in the function.

Keyword Arguments

```
def my_func(name, age):  
    pass
```

```
my_func(name="Chetan", age=33)
```

These are **keyword arguments** as arguments are passed to function by keywords

Positional Arguments

```
def my_func(name, age):  
    pass
```

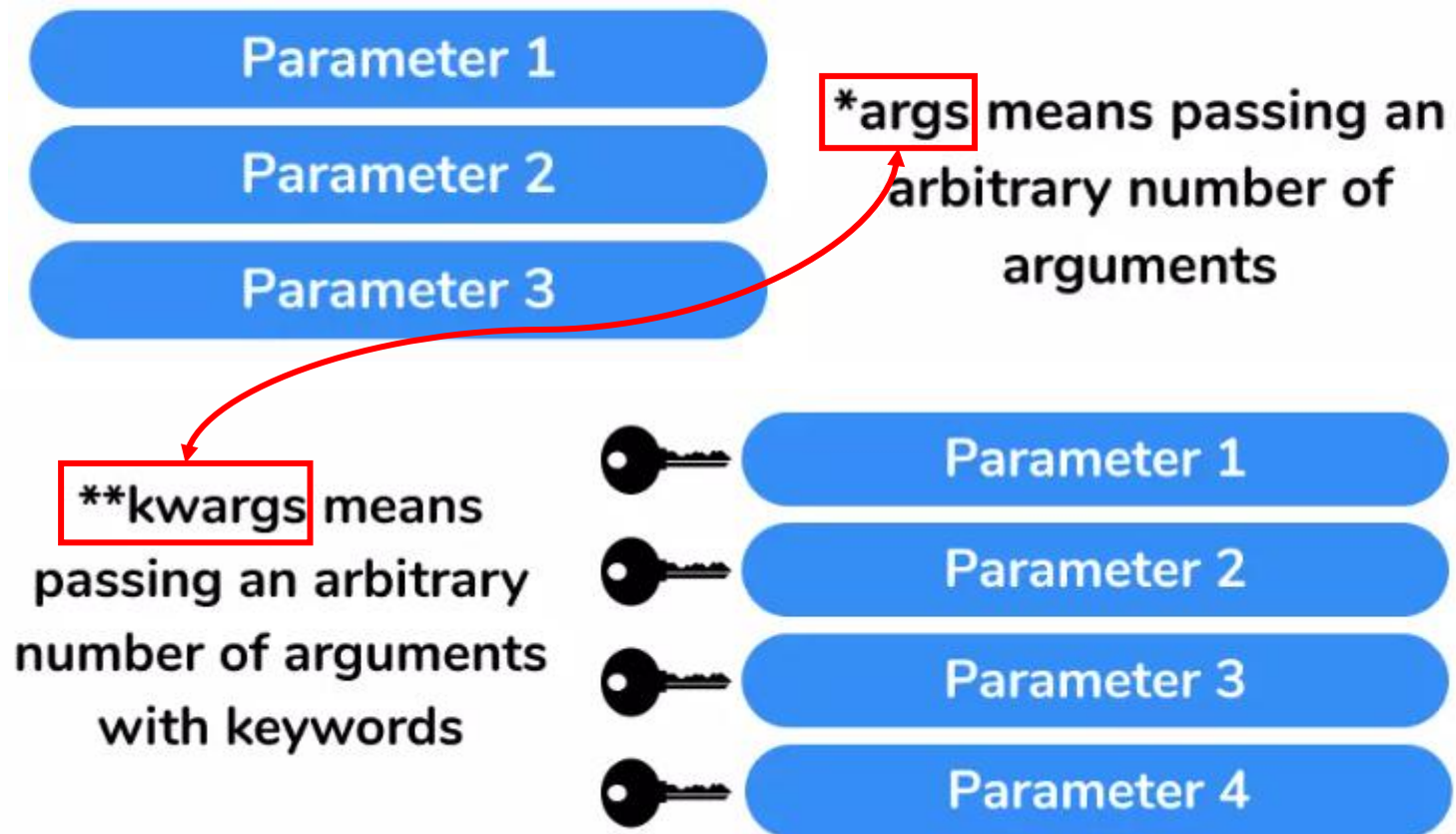
```
my_func("Chetan", 33)
```

These are **positional arguments** as arguments are passed to function by position

Default & asterisk (* & **)

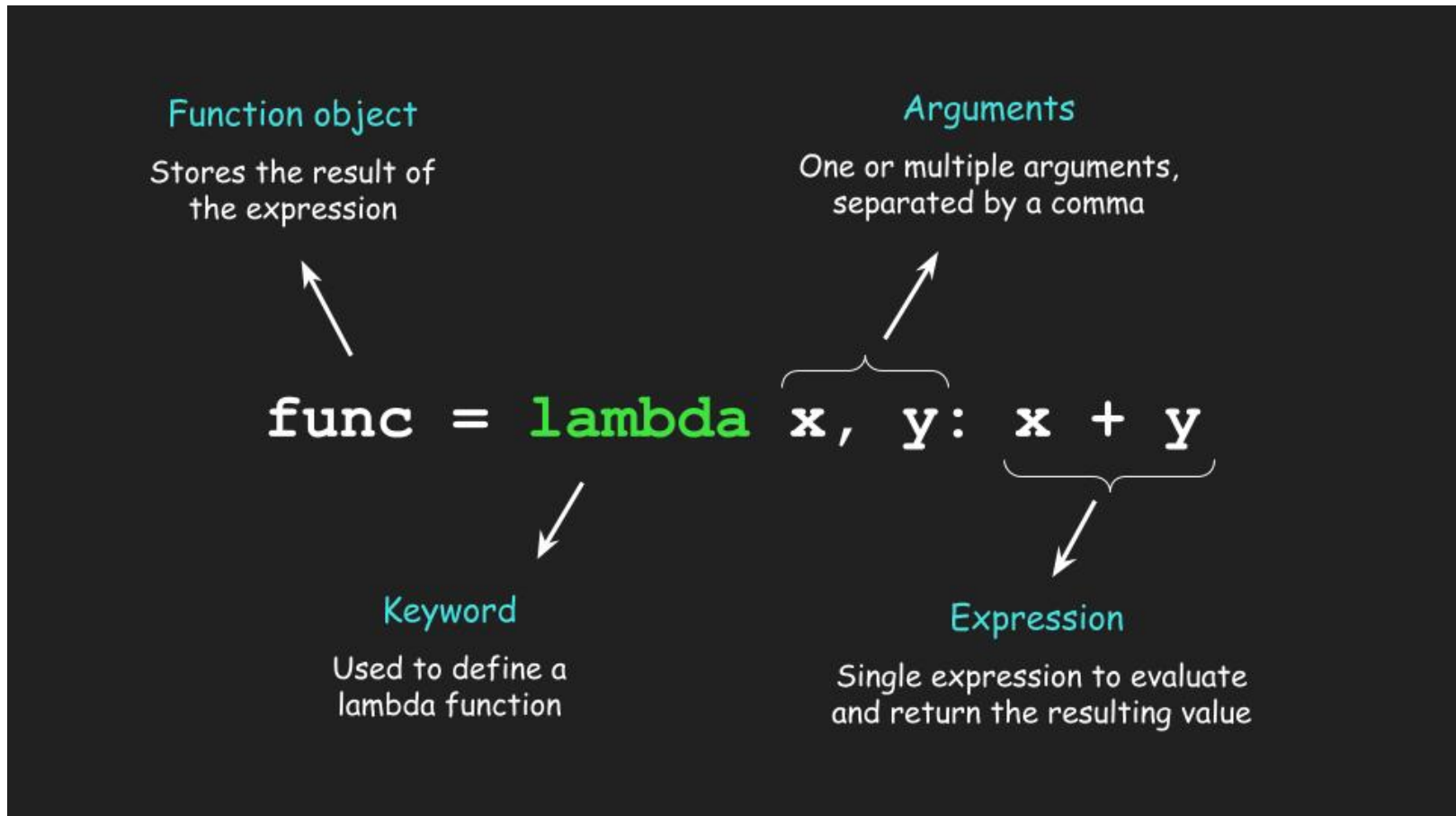
- Specify **default values** for parameters

```
def menu(wine, entree, dessert='pudding'):  
    return {'wine': wine, 'entree': entree, 'dessert': dessert}
```



Anonymous Functions

- A Python *lambda function* is an anonymous function expressed as a single statement.



Dynamic typing

- What is **dynamic typing**?
 - In **static-typed** languages like Java or C++, you have to declare the type of variables before using them.
 - In Python, you can code without giving a second thought to data types until runtime—which is one of the features that make Python particularly beginner-friendly.
- However, this dynamic nature can also lead to bugs that are difficult to debug, especially in large codebases or complex data processing pipelines, where the data flow might not be immediately obvious.

```
>>> x = 1234      # x is an integer.
>>> print(x + 1)
1235
>>> x = str(x)    # x is now a string.
>>> print(x[0])
1
>>> x = list(x)   # x is now a list.
>>> x
['1', '2', '3', '4']
```

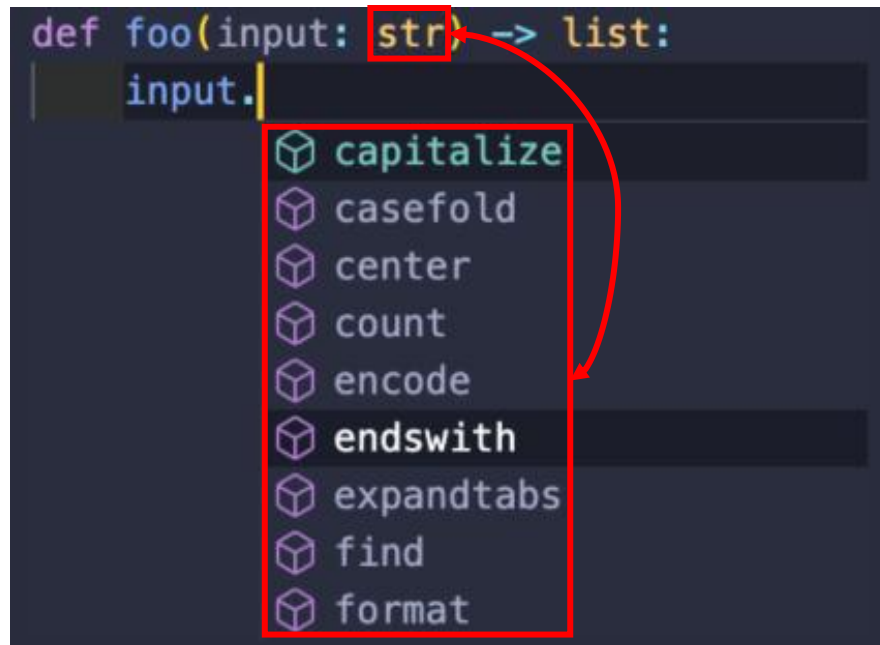
Type Hints

- Improved Code Readability
- Improved Code Quality
- Enhanced Development Experience
- Collaboration and Maintainability

```
## 沒有 Type Hints 型別註釋
## 每個引數要怎麼用，大概要看完函式主體才知道
def send_request(request_data,
                  headers,
                  user_id,
                  as_json):
    pass
```

```
## 有 Type Hints 型別註釋
## 函式主體還沒寫、可以略知一二每個引數的角色
def send_request(request_data : Any, # 可以是任何型別
                  headers: Optional[dict[str, str]], # 可以是 None 或者是從 str 映射到 str 的 dict
                  user_id: Optional[UserId] = None, # 可以是 None 或者是 UserId 類別
                  as_json: bool = True): # 必須是 bool
    pass
```

Type Checking



- Large codebases often encounter runtime bugs due to unexpected data input, which can be challenging to debug, requiring tracing through multiple layers.
- **Static typing** offers a solution by enforcing type constraints during development, preventing these bugs and saving valuable development time.

Static Type checking - mypy

- **mypy** is a static type checker for Python.
- Type checkers help ensure that you're using variables and functions in your code correctly. It finds bugs in your programs without even running them.
- Adding type hints for mypy does not interfere with the way your program would otherwise run. Think of type hints as similar to comments!
- You can always use the Python interpreter to run your code, even if mypy reports errors.

```
PS C:\Users\User\Desktop\北大統計-授課資料\113-2 日間部-Python程式設計> py -m mypy '.\Section 3.1 - case test.py'
Section 3.1 - case test.py:12: error: Incompatible return value type (got "float", expected "str | None") [return-value]
Section 3.1 - case test.py:16: error: Incompatible return value type (got "float", expected "str | int") [return-value]
Found 2 errors in 1 file (checked 1 source file)
```

Atomic vs Composite

- **Atomic** types :
 - such as **int**, **float**, and **str**, are simple and indivisible, and their type annotations can be provided directly using the type itself, like **str**.
- **Composite** types :
 - such as **List** and **Dict** are composed of other types, they often required importing specific definitions from the typing module, such as **Dict[str, int]** for a dictionary with string keys and float values.
- **User-Defined** types
 - Create a class and use it as a type hint
 - **Type** from the **typing** module
 - **NewType** from the **typing** module

Recurrence Relation

- A **recurrence relation** is an equation according to which the n th term of a sequence of numbers is equal to some combination of the previous terms.
- Often, only k previous terms of the sequence appear in the equation, for a parameter k that is independent of n ; this number k is called the **order of the relation**.

- (Def)

A recurrence relation of **order k** has the form

$$a_n = f(n, a_{n-1}, \dots, a_{n-k}) \text{ for } n \geq k$$

where

$$f : \mathbb{N} \times \mathcal{D}^k \rightarrow \mathcal{D}^k$$

is a function that involves k consecutive elements of sequence. In this case, k initial values are needed for defining a sequence

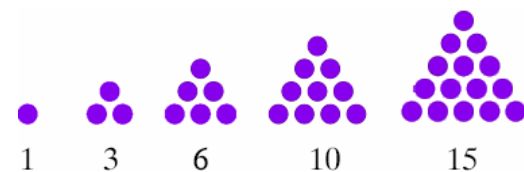
Examples of recursive relation

- Factorial number

$$n! = n \times (n - 1)!, \quad 0! = 1$$

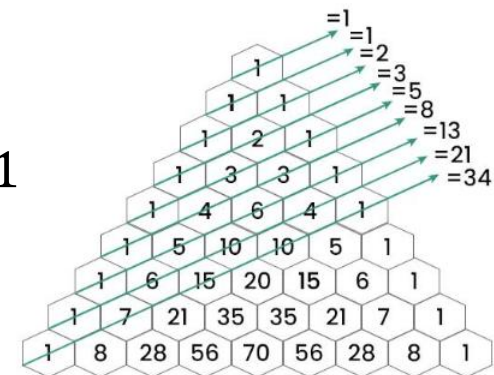
- Triangular number

$$T_n = T_{n-1} + n, \quad T_1 = 1$$



- Fibonacci numbers

$$F_n = F_{n-1} + F_{n-2}, \quad F_0 = 0 \text{ and } F_1 = 1$$

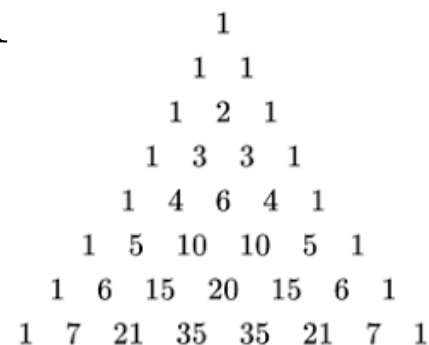


- Binomial coefficients

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}, \quad \binom{n}{0} = \binom{n}{n} = 1$$

or computed with 1-dim recurrence

$$\binom{n}{k} = \binom{n}{k-1} \left(\frac{n-k+1}{k} \right)$$



Golden Ratio

- Since $F_n < F_{n-1}$

$$1 \leq G_n = \frac{a_{n+1}}{a_n} = \frac{a_n + a_{n-1}}{a_n} \leq \frac{a_n + a_n}{a_n} = 2$$

$\{G_n\}$ is bounded and

$$G_n = 1 + \frac{1}{G_{n-1}}$$

observe $G_1 = 1, G_2 = 2$

$$G_3 = 1 + \frac{1}{G_2} > G_1, \quad G_5 = 1 + \frac{1}{G_4} > 1 + \frac{1}{G_2} = G_3$$

$$G_4 = 1 + \frac{1}{G_3} < 1 + \frac{1}{G_1} = G_2, \quad G_6 = 1 + \frac{1}{G_5} < 1 + \frac{1}{G_3} = G_4$$

both $\{G_{2n}\}$ and $\{G_{2n+1}\}$ are **monotone bounded** \Rightarrow both converge

$$\begin{cases} \alpha = \lim_{n \rightarrow \infty} G_{2n} = \lim_{n \rightarrow \infty} 1 + \frac{1}{G_{2n-1}} = 1 + \frac{1}{\beta} \\ \beta = \lim_{n \rightarrow \infty} G_{2n+1} = \lim_{n \rightarrow \infty} 1 + \frac{1}{G_{2n}} = 1 + \frac{1}{\alpha} \end{cases} \Rightarrow \alpha = \beta$$

$\{G_n\}$ is converge

$$g = \lim_{n \rightarrow \infty} G_n = \lim_{n \rightarrow \infty} 1 + \frac{1}{G_{n-1}} = 1 + \frac{1}{g} \Rightarrow g = \frac{1 + \sqrt{5}}{2}$$

Linear Recurrence Relation

- (Thm)

For the recurrence relation of order k

$$a_n = c_1 a_{n-1} + \cdots + c_k a_{n-k}$$

if its *characteristic equation*

$$t^k - c_1 t^{k-1} - \cdots - c_{k-1} t - c_k = 0$$

has distinct roots r_1, \dots, r_k then the general solution is

$$a_n = \alpha_1 r_1^n + \cdots + \alpha_k r_k^n$$

where $\alpha_1, \dots, \alpha_k$ are arbitrary constants.

- (Cor)

If the characteristic equation has *repeated roots* r_1, \dots, r_s with multiplicities m_1, \dots, m_s respectively, then the general solution is a linear combination of the solutions

$$\begin{array}{cccc} r_1^n & nr_1^n & \cdots & n^{m_1-1} r_1^n \\ r_2^n & nr_2^n & \cdots & n^{m_2-1} r_2^n \\ \vdots & \vdots & \ddots & \vdots \\ r_s^n & nr_s^n & \cdots & n^{m_s-1} r_s^n \end{array}$$

General Solution - 1

- Fibonacci number

$$F_n = F_{n-1} + F_{n-2}$$

the characteristic equation is

$$t^2 - t - 1 = 0 \Rightarrow r_1 = \frac{1 + \sqrt{5}}{2}, \quad r_2 = \frac{1 - \sqrt{5}}{2}$$

then the general solution is given by

$$f_n = c_1 \left(\frac{1 + \sqrt{5}}{2} \right)^n + c_2 \left(\frac{1 - \sqrt{5}}{2} \right)^n$$

Set

$$\begin{cases} c_1 + c_2 = 0 \\ c_1 \left(\frac{1 + \sqrt{5}}{2} \right) + c_2 \left(\frac{1 - \sqrt{5}}{2} \right) = 1 \end{cases} \Rightarrow c_1 = -c_2 = \frac{1}{\sqrt{5}}$$

Thus

$$f_n = \frac{1}{\sqrt{5}} \left(\frac{1 + \sqrt{5}}{2} \right)^n - \frac{1}{\sqrt{5}} \left(\frac{1 - \sqrt{5}}{2} \right)^n, \quad n \geq 0$$

General Solution - 2

- $\{T_n\}$ is defined by $T_0 = 1, T_1 = 1, T_2 = 2$ and the recurrence relation

$$T_n = 4T_{n-1} + 3T_{n-2} - 18T_{n-3}$$

the characteristic polynomial is

$$t^3 - 4t^2 - 3t + 18 = (t - 3)^2(t + 2)$$

$$\Rightarrow t = 3 \text{ with multiplicity } 2, t = -2$$

then the general solution

$$T_n = c_1 3^n + c_2 n 3^n + c_3 (-2)^n$$

set

$$\begin{cases} T_0 = c_1 + c_3 = 1 \\ T_1 = 3c_1 + 3c_2 - 2c_3 = 1 \\ T_2 = 9c_1 + 18c_2 + 4c_3 = 2 \end{cases}$$

$$\Rightarrow c_1 = \frac{4}{5}, c_2 = -\frac{1}{3}, c_3 = \frac{1}{5}$$

Thus

$$\begin{aligned} T_n &= \left(\frac{4}{5} - \frac{n}{3}\right) 3^n + \frac{1}{5} (-2)^n \\ &= \frac{(12 - 5n)3^n + 3(-2)^n}{15} \end{aligned}$$

Exercise

- X_n is defined by $X_0 = 1, X_1 = 3$ and

$$X_n = 5X_{n-1} + 6X_{n-2}$$

- X_n is defined by $X_0 = -1, X_1 = 0, X_2 = 1$ and

$$X_n = 6X_{n-1} - 12X_{n-2} + 8X_{n-3}$$

☺ Find the general solution of X_n

☺ Compare general solution with the recursive result for first 15 number.

Recursion

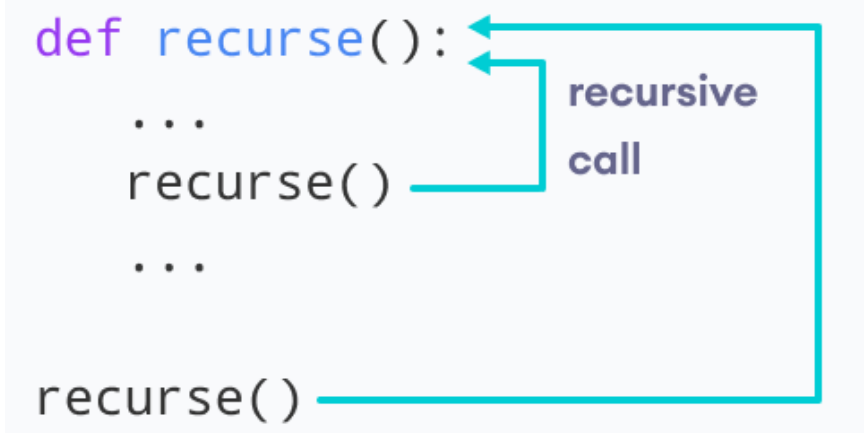
- Function calls itself ? This is called recursion.

```

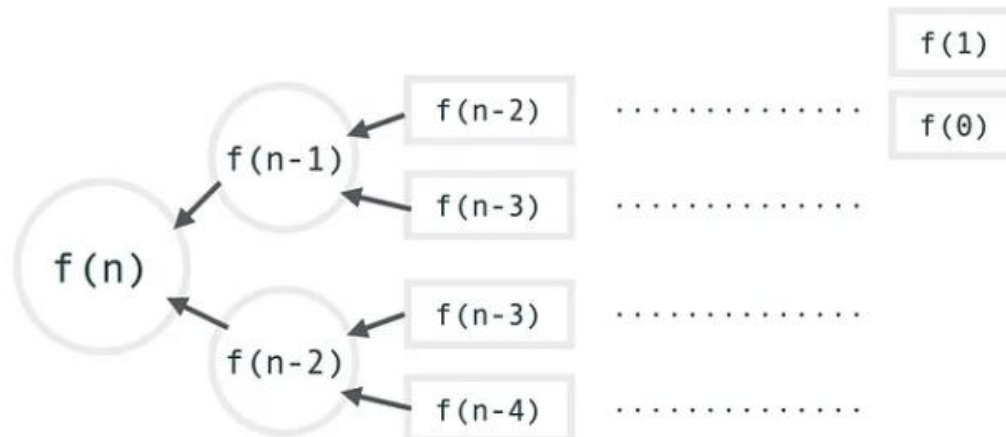
def recurse():
    ...
    recurse()
    ...

recurse()
    
```

recursive call



- Python saves the universe again by raising an exception if you get too deep.



Recursion for Partial correlation

- The partial correlation X and Y given a controlling variable Z is

$$\rho_{XY|Z} = \frac{\rho_{XY} - \rho_{XZ}\rho_{ZY}}{\sqrt{1 - \rho_{XZ}^2}\sqrt{1 - \rho_{ZY}^2}}$$

The zeroth order partial correlation is defined to be the regular correlation coefficient

$$\rho_{XY|\emptyset} = \rho_{XY} = \frac{\text{Cov}(X, Y)}{\sqrt{V(X)}\sqrt{V(Y)}}$$

- For any $z^* \in \mathbf{Z} = (Z_1, \dots, Z_n)$, the partial correlation between X and Y given a set of n controlling variables \mathbf{Z}

$$\rho_{XY|\mathbf{Z}} = \frac{\rho_{XY|\mathbf{Z} \setminus z^*} - \rho_{Xz^*|\mathbf{Z} \setminus z^*}\rho_{z^*Y|\mathbf{Z} \setminus z^*}}{\sqrt{1 - \rho_{Xz^*|\mathbf{Z} \setminus z^*}^2}\sqrt{1 - \rho_{z^*Y|\mathbf{Z} \setminus z^*}^2}}$$

Actually, the n th order partial correlation can be easily computed from three $(n-1)$ th order partial correlations.

Geometry of Partial correlation

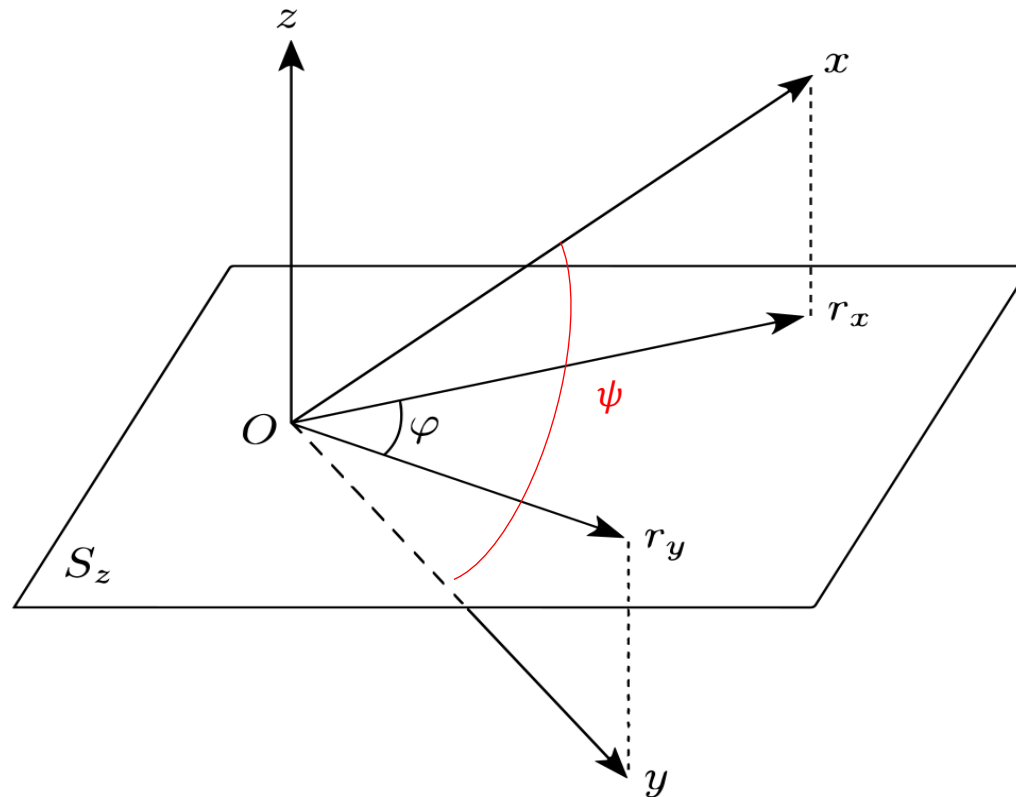
- Consider two regression models

$$Y = \beta_0 + \beta_1 Z + \varepsilon_Y, \quad \hat{\varepsilon}_Y = Y - \hat{Y}$$

$$X = \theta_0 + \theta_1 Z + \varepsilon_X, \quad \hat{\varepsilon}_X = X - \hat{X}$$

$$\cos(\varphi) = \rho_{XY|Z} = \text{Corr}(\hat{\varepsilon}_Y, \hat{\varepsilon}_X)$$

$$\cos(\psi) = \rho_{XY} = \text{Corr}(Y, X)$$



Global vs Local

Local Variable Vs. Global Variable (Primary Differences)

Local Variable



A variable that is declared **inside** a function of a computer program.



Accessible **only within the function** it is declared.



Created when the function starts executing and is destroyed when the execution is complete.



More reliable and secure since other functions cannot change the value.

Global Variable

A variable that is declared **outside** the functions of a computer program.

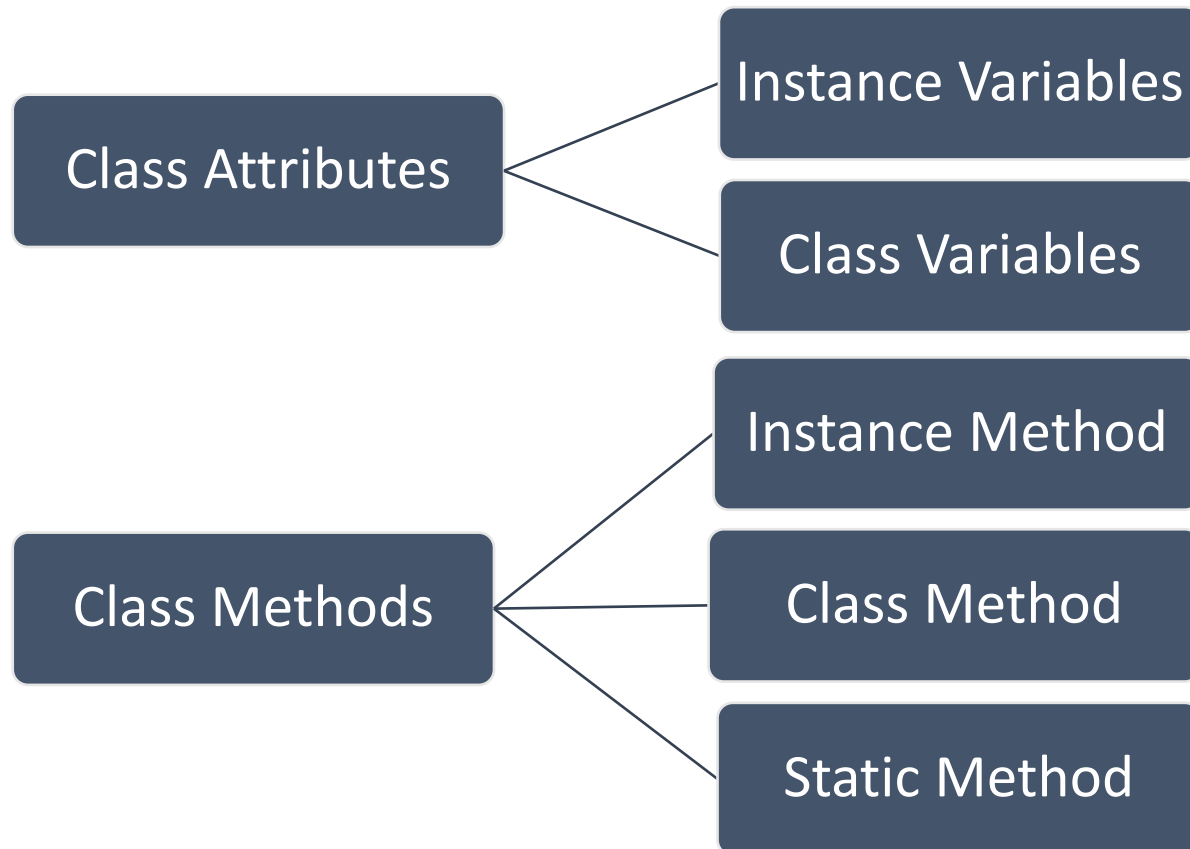
Accessible by **all the functions** in the program.

Remains in existence for the entire time the program is executed.

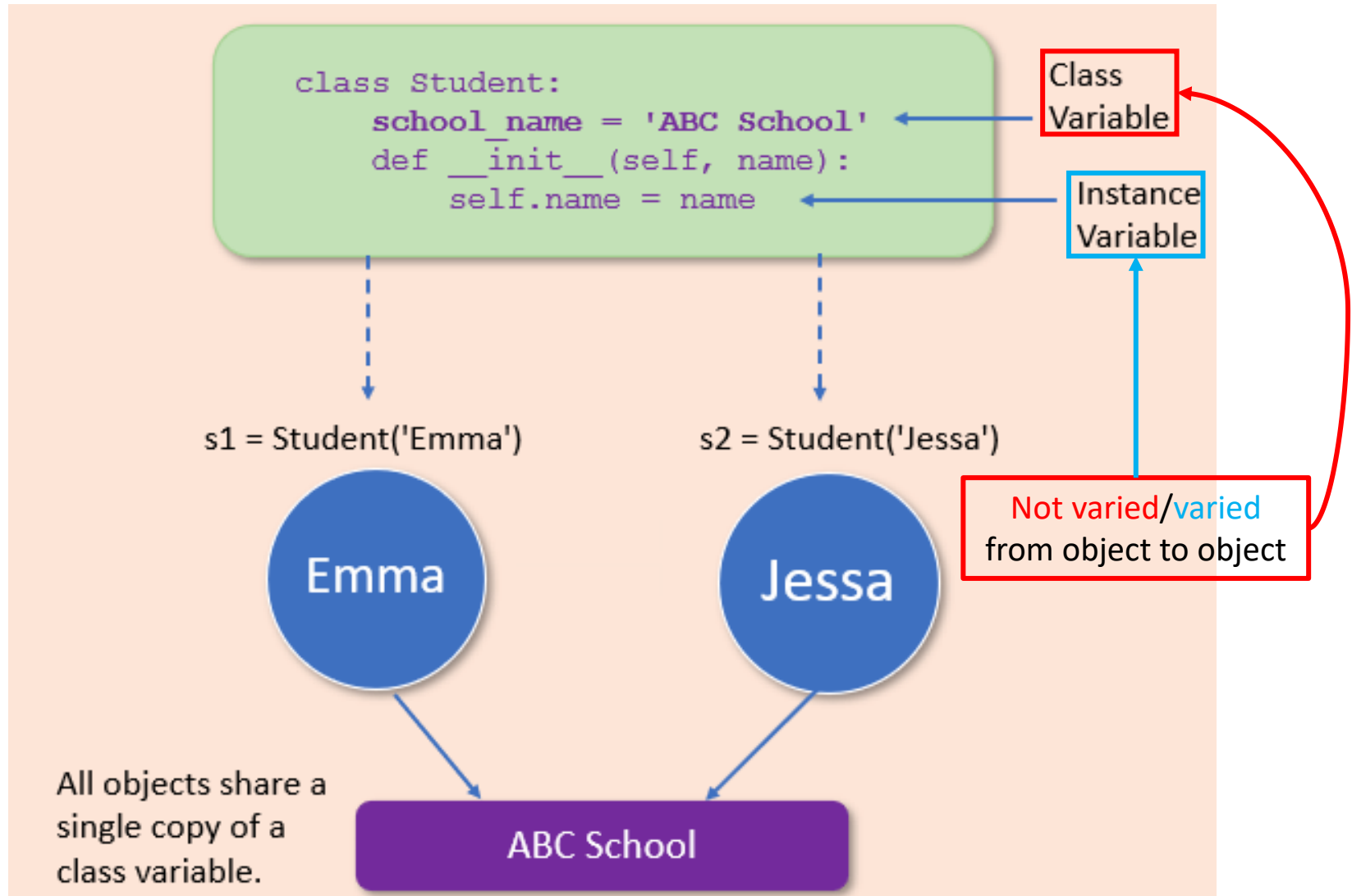
Accessible by multiple functions; therefore, its value can be changed.

Object & Class

- An *object* is a custom data structure containing both
 - data (variables, called *attributes*)
 - code (functions, called *methods*)



Attributes



Instance method & Static method

```
class Student:
    def __init__(self, name, age):
        self.name = name
        self.age = age
    def show(self):
        print('Name:', self.name, 'Age:', self.age)

emma = Student("Jessa", 14)
emma.show()
```

Annotations in the code:

- `__init__`: Constructor to initialize Instance variables
- `self.name` and `self.age`: Instance variables
- `show`: Instance method
- `self` in `show`: `Self` refers to the calling object
- `emma.show()`: Call instance method

Static methods are defined inside a class, and it is pretty similar to defining a regular [function](#). To declare a static method, use this idiom:

```
class C:
    @staticmethod
    def f(arg1, arg2, ...): ...
```


Class method

```
class Student:
```

```
    school_name = 'ABC School' ← Class Variables
```

```
    def __init__(self, name, age): ← Constructor to initialize  
        self.name = name           Instance variables  
        self.age = age
```

Class Method {

```
    @classmethod  
    def change_school(cls, name):  
        print(Student.school_name) ← Access Class Variables  
        Student.school_name = name ← Modify Class Variables
```

cls refer to the Class

```
jessa = Student('Jessa', 14)  
Student.change_school('XYZ School') ← Call Class Method
```

Exercise

- Let $\mathcal{M}_K = (M_1, \dots, M_K)$ with its power set $\mathcal{P}(\mathcal{M}_K)$. $\mathcal{P}(\mathcal{M}_K)$ is isomorphism with the binary digits $\{e_d\}_{d=0}^{2^K-1}$ of numbers from 0 to $2^K - 1$.

| $\mathcal{P}(\mathcal{M}_3)$ | \emptyset | $\{M_1\}$ | $\{M_2\}$ | $\{M_1, M_2\}$ | $\{M_3\}$ | $\{M_1, M_3\}$ | $\{M_2, M_3\}$ | \mathcal{M}_3 |
|------------------------------|------------------|------------------|------------------|------------------|------------------|------------------|------------------|------------------|
| Binary Digits | 000 ₂ | 001 ₂ | 010 ₂ | 011 ₂ | 100 ₂ | 101 ₂ | 110 ₂ | 111 ₂ |
| e_d | (0,0,0) | (0,0,1) | (0,1,0) | (0,1,1) | (1,0,0) | (1,0,1) | (1,1,0) | (1,1,1) |

- The Sierpinski triangle matrix \mathbf{H}_K can be obtained in two way
 - $\mathbf{H}_K = [h_{i,j}]$, for each element $h_{i,j}$

$$h_{i,j} = \begin{cases} 1, & \langle e_i, e_j \rangle = 0 \\ 0, & \langle e_i, e_j \rangle \neq 0 \end{cases}$$

- By recursion

$$\mathbf{H}_K = \begin{bmatrix} \mathbf{H}_{K-1} & \mathbf{H}_{K-1} \\ \mathbf{H}_{K-1} & \mathbf{O}_{K-1} \end{bmatrix}, \quad \mathbf{H}_1 = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$$

- ☺ Write a Python class that includes two different methods and can plot multiple graphs for different K values in a single figure. Draw K=1,2,3,4 for Method 1 and K=1,3,5,7 for Method 2

Inheritance

- Inheritance allows us to define a class that inherits all the methods and properties from another class.
- The **parent/base class/superclass** is the class being inherited from, and the **child/derived class/subclass** is the class that inherits from another class.

Python Inheritance Syntax

```
# define a superclass
class super_class:
    # attributes and method definition

# inheritance
class sub_class(super_class):
    # attributes and method of super_class
    # attributes and method of sub_class
```



Getters and Setters

- Some OOP support private object attributes that can't be accessed directly from the outside.
- Python doesn't have private attributes, but you can write getters and setters with obfuscated attribute names to get a little privacy.

Getter — `@property`:

The `@property` decorator is used to define a method that acts as a "getter" for an attribute. When you access the attribute, the method decorated with `@property` will be called automatically. This is useful for calculations or other actions that need to occur when the attribute is accessed.

Setter — `@<attribute>.setter`:

The `@<attribute>.setter` decorator is used to define a method that acts as a "setter" for an attribute. When you set the attribute, the method decorated with `@<attribute>.setter` will be called automatically. This is useful for validation or other actions that need to occur when the attribute is modified.