



Python Programming on Colab

Prof. Kuan-Ting Lai

2020/3/26

Google Colaboratory

+ Code + Text  Copy to Drive

Connect  Editing 

Table of contents Code snippets Files 

Introducing Colaboratory

Getting Started

More Resources

Machine Learning Examples: Seedbank

 SECTION

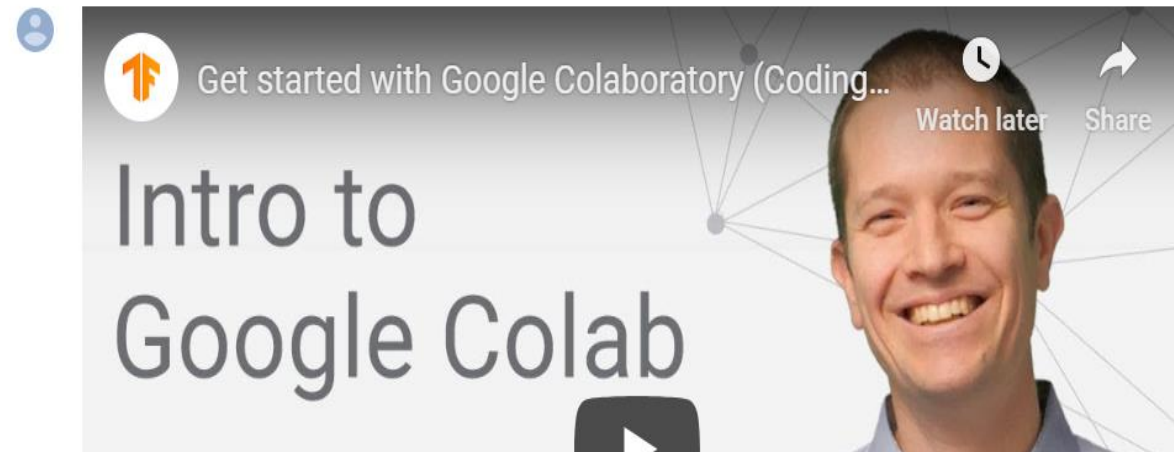
Welcome to Colaboratory!

Colaboratory is a free Jupyter notebook environment that requires no setup and runs entirely in the cloud.

With Colaboratory you can write and execute code, save and share your analyses, and access powerful computing resources, all for free from your browser.

[] **Introducing Colaboratory**

This 3-minute video gives an overview of the key features of Colaboratory:



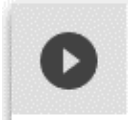
New Python 3 Notebook

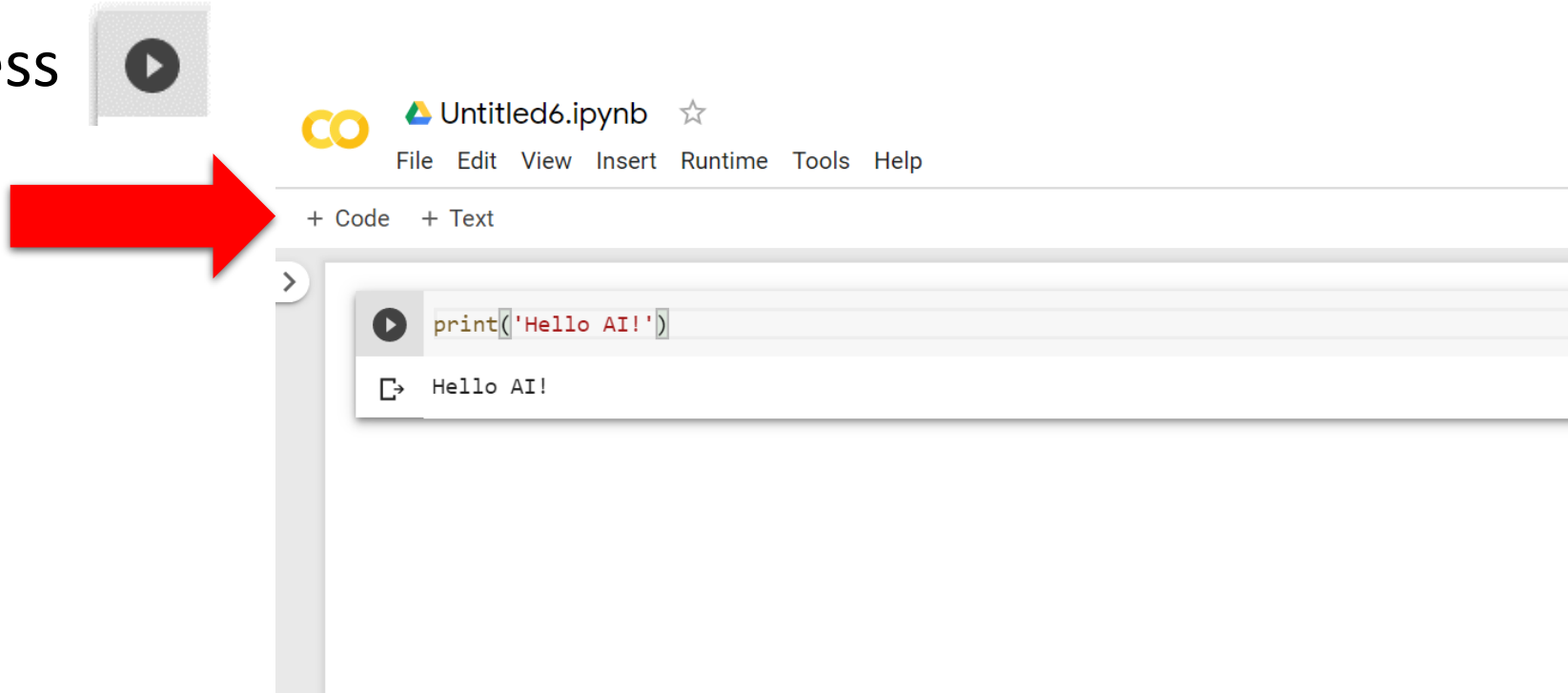
The screenshot shows the Google Colaboratory interface with a modal open for creating a new notebook. The modal has a header with tabs: Examples, Recent, Google Drive, GitHub, and Upload. The 'Recent' tab is active, showing a list of notebooks. A red arrow points to the 'NEW PYTHON 3 NOTEBOOK' button at the bottom of the modal.

Filter notebooks	Title	First opened	Last opened	
	Welcome To Colaboratory	Jan 19, 2019	0 minutes ago	
	boston_housing_price.ipynb	Mar 4, 2019	13 days ago	
	newswires_multi_class.ipynb	Mar 2, 2019	13 days ago	
	Overview of Colaboratory Features	Jan 19, 2019	13 days ago	
	imdb.ipynb	Mar 2, 2019	13 days ago	

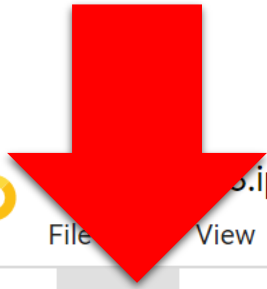
NEW PYTHON 3 NOTEBOOK CANCEL

Hello World!

- Add a code cell by pressing “+ Code”
- Enter code
 - `print("Hello AI!")`
- Press 



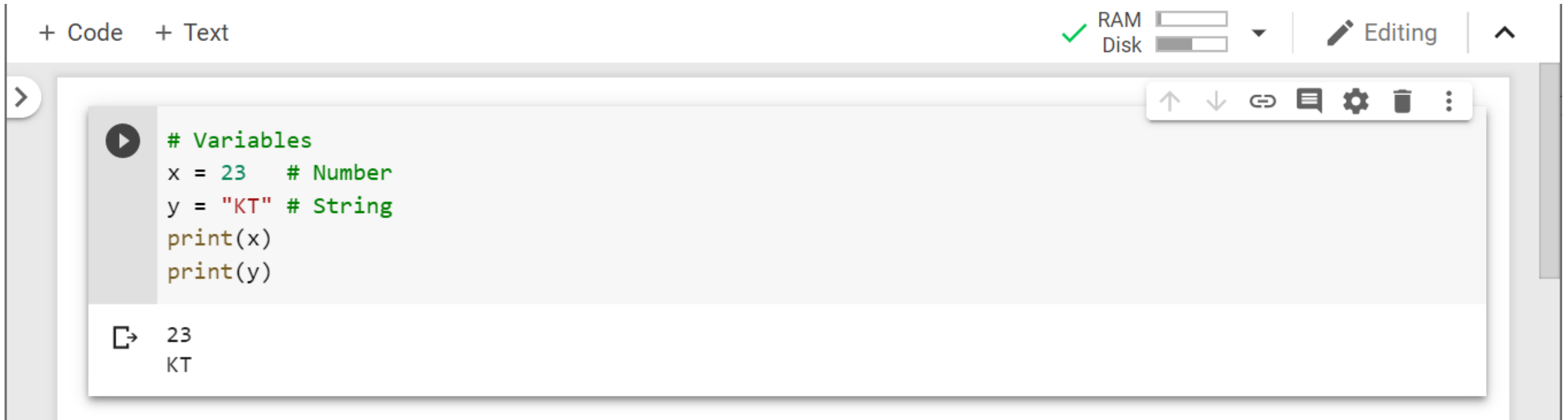
Adding a Text Cell



The screenshot shows the JupyterLab interface. At the top left, the 'co' logo is visible. The main menu includes 'File', 'View', 'Insert', 'Runtime', 'Tools', and 'Help'. A red arrow points to the '+ Text' button in the top left corner. The '+ Code' button is also visible. On the right side, there are 'Comment', 'Share', and a user profile icon. Below the menu, there are 'RAM' and 'Disk' usage indicators, and an 'Editing' mode indicator. The main workspace shows a text cell with the text 'This is a Simple Tutorial for Python!' and a code cell with the code `[1] print('Hello AI!')`. The output of the code cell is 'Hello AI!'.

Creating Variables

- Variables are containers for storing data values.



The screenshot shows a code editor interface. At the top, there are tabs for '+ Code' and '+ Text'. On the right side, there are indicators for 'RAM' (with a green checkmark and a progress bar) and 'Disk' (with a progress bar), and a status 'Editing' with a pencil icon. Below the editor, there is a toolbar with icons for undo, redo, link, comment, settings, and delete. The code editor contains the following Python code:

```
# Variables  
x = 23 # Number  
y = "KT" # String  
print(x)  
print(y)
```

Below the code, there is a terminal output window showing the results of the execution:

```
23  
KT
```

Python Data Types

Text Type:	<code>str</code>
Numeric Types:	<code>int, float, complex</code>
Sequence Types:	<code>list, tuple, range</code>
Mapping Type:	<code>dict</code>
Set Types:	<code>set, frozenset</code>
Boolean Type:	<code>bool</code>
Binary Types:	<code>bytes, bytearray, memoryview</code>

Printing Data Type

- Using `type()`

+ Code + Text

✓ RAM | Disk | Editing ^



```
# Variables
x = 23 # Number
y = "KT" # String
print(type(x))
print(type(y))
```

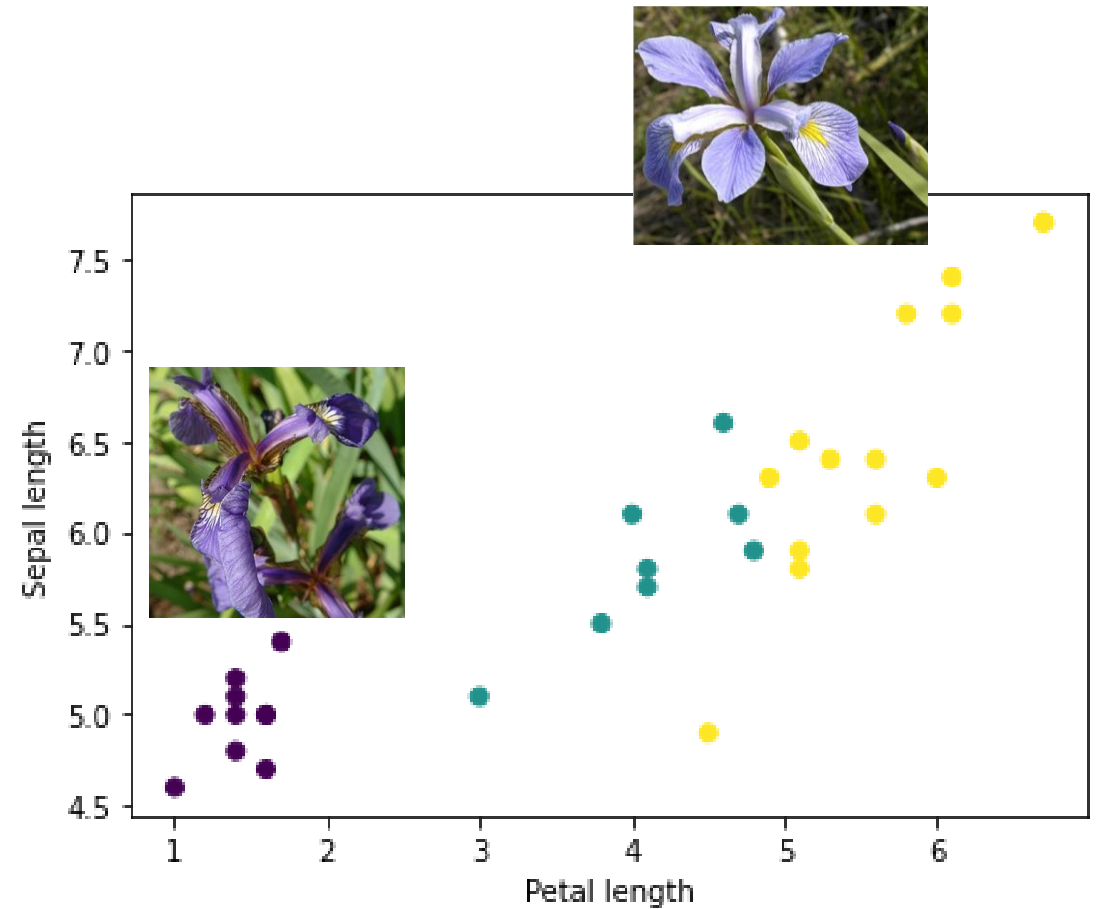


```
<class 'int'>
<class 'str'>
```


Conditional Control (if ... else ...)

```
▶ petal_len=2  
|  
if petal_len > 2.5:  
    print(" Iris Setosa")  
else:  
    if petal_len < 5:  
        print("Iris Versicolor")  
    else:  
        print("Iris Virginica")
```

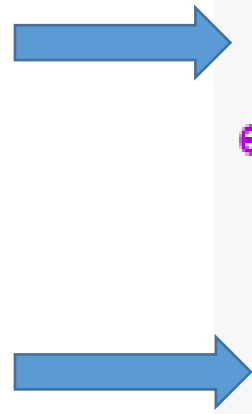
☞ Iris Versicolor



Python Syntax

- Indentation

```
if petal_len > 2.5:
    print(" Iris Setosa")
else:
    if petal_len < 5:
        print("Iris Versicolor")
    else:
        print("Iris Virginica")
```



Python Conditions and If statements

- Equals: `a == b`
- Not Equals: `a != b`
- Less than: `a < b`
- Less than or equal to: `a <= b`
- Greater than: `a > b`
- Greater than or equal to: `a >= b`

Elif

- Else if



The image shows a code editor interface. At the top, there are tabs for '+ Code' and '+ Text'. On the right side of the top bar, there are indicators for 'RAM' and 'Disk' usage, a green checkmark, and a status 'Editing' with a pencil icon. Below the top bar, there is a toolbar with icons for navigation (up, down), linking, commenting, settings, and deleting. The main area contains a code block with a play button icon on the left. The code is as follows:

```
a = 33
b = 33
if b > a:
    print("b is greater than a")
elif a == b:
    print("a and b are equal")
```

Below the code block, there is a terminal or output area showing the result of the execution: a box with a right-pointing arrow containing the text 'a and b are equal'.

Python Loops

- while loops
- for loops

+ Code + Text

```
i = 0  
while i < 10:  
    print(i)  
    i += 1
```

```
0  
1  
2  
3  
4  
5  
6  
7  
8  
9
```

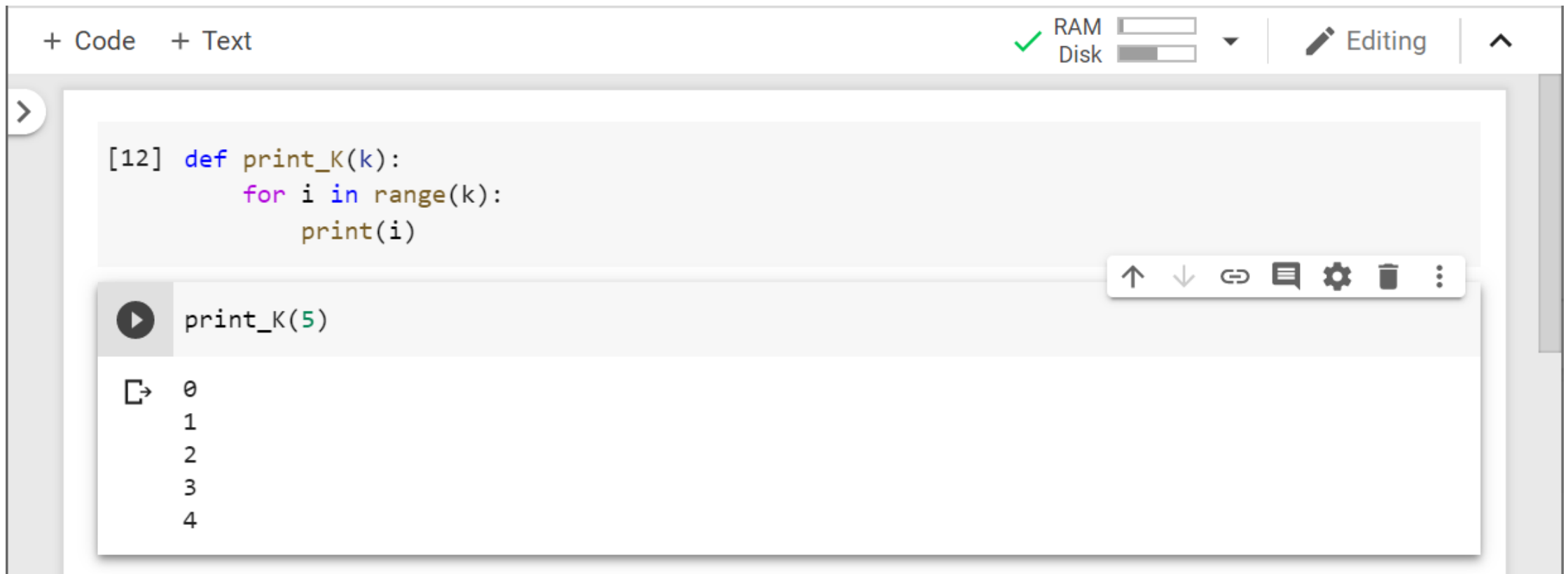
+ Code + Text

```
for i in range(10):  
    print(i)
```

```
0  
1  
2  
3  
4  
5  
6  
7  
8  
9
```

Defining function

- Using the def keyword



The screenshot shows a Jupyter Notebook interface. At the top, there are tabs for '+ Code' and '+ Text', and a status bar indicating 'RAM' and 'Disk' usage, along with an 'Editing' mode indicator. The main area contains a code cell with the following Python code:

```
[12] def print_K(k):  
    for i in range(k):  
        print(i)
```

Below the code cell, there is a run button (a play icon) and the command `print_K(5)`. The output of this command is displayed in a separate cell, showing the numbers 0, 1, 2, 3, and 4, each on a new line.

Python Arrays

+ Code + Text

✓ RAM | Editing ^

↓

↑ ↓ ↻ 🗨 ⚙ 🗑 ⋮

```
▶ cars = ["Benz", "Lexus", "BMW"]  
print(cars[0])
```

📄 Benz

Looping Array Elements

- Code + Text

```
[17] cars = ["Benz", "Lexus", "BMW"]  
      print(cars[0])
```

↳ Benz

```
▶ for x in cars:  
   print(x)
```

↳ Benz
Lexus
BMW

Array Methods

Method	Description
<u>append()</u>	Adds an element at the end of the list
<u>clear()</u>	Removes all the elements from the list
<u>copy()</u>	Returns a copy of the list
<u>count()</u>	Returns the number of elements with the specified value
<u>extend()</u>	Add the elements of a list (or any iterable), to the end of the current list
<u>index()</u>	Returns the index of the first element with the specified value
<u>insert()</u>	Adds an element at the specified position
<u>pop()</u>	Removes the element at the specified position
<u>remove()</u>	Removes the first item with the specified value
<u>reverse()</u>	Reverses the order of the list
<u>sort()</u>	Sorts the list

Class

+ Code + Text

✓ RAM
Disk

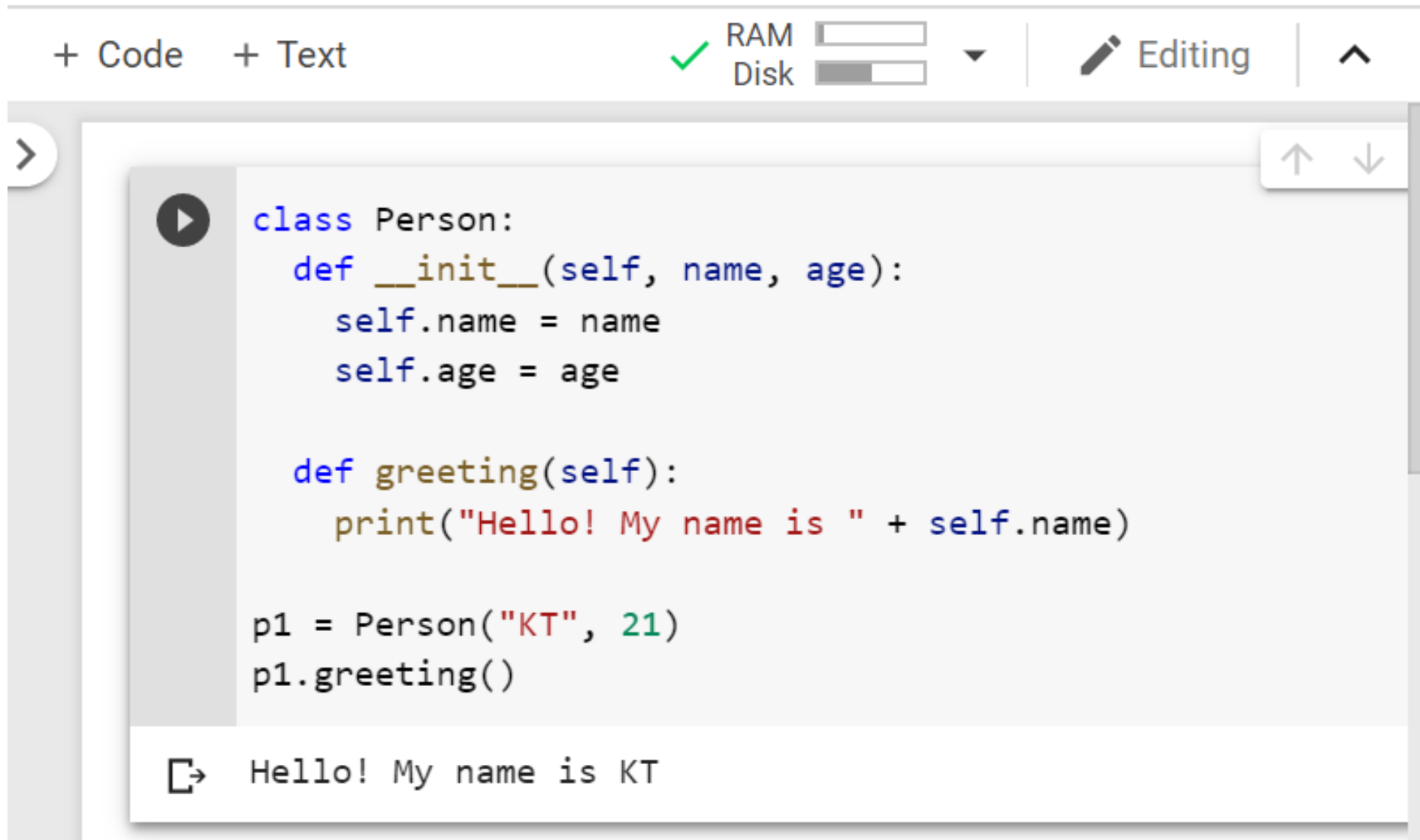
Editing

```
[22] # Define a Class  
class MyClass:  
    x = 23  
    y = 5
```

```
# Create a Object  
p1 = MyClass()  
print(p1.x)
```

23

Constructor and Methods



The image shows a code editor interface with a toolbar at the top. The toolbar includes a green checkmark, RAM and Disk usage indicators, an 'Editing' mode indicator with a pencil icon, and a caret icon. The code editor contains Python code for a class and its execution. The code defines a class 'Person' with an '__init__' method and a 'greeting' method. It then creates an instance 'p1' and calls the 'greeting' method. The output of the execution is displayed in a terminal window at the bottom of the editor.

```
+ Code + Text | RAM [ ] | Disk [ ] | Editing | ^
```

```
> | [ ] [ ]
```

```
▶ class Person:  
    def __init__(self, name, age):  
        self.name = name  
        self.age = age  
  
    def greeting(self):  
        print("Hello! My name is " + self.name)  
  
p1 = Person("KT", 21)  
p1.greeting()
```

```
☞ Hello! My name is KT
```

References

- <https://www.w3schools.com/python>