Python

Input/Output, Variables, Types, Assignment

In this lecture

- Introduction to Python
- Output
- Input
- Types
- Declaring Variables
- Assigning values to variables

What is a Computer?

Computer

- A computer program (or application) is a collection of **instructions** that perform a specific task.

Introduction to Python

What is Python?

“Python is a high level programming language, and its core design philosophy is all about code readability and a syntax which allows programmers to express concepts in a few lines of code.”

Guido van Rossum

History

Python was originally developed by Dutch programmer **Guido van Rossum** throughout the 1980s. Guido named the language after UK comedy series Monty Python.

Python 0.9.0 was released in 1991

Python 2 was released in 2000

Python 3.0 was released in 2008

Python has gained in popularity thanks data scienc
and national school curriculums.

Philosophy

First posted in 1999, and later published in the *Zen of Python*:

- Beautiful is better than ugly.
- Explicit is better than implicit.
- Simple is better than complex.
- Complex is better than complicated.
- Readability counts.

Advantages

- **Readable**: intuitive and strict syntax
- **Productive**: saves a lot of code
- **Portable**: for every operating system
- **Reloaded**: it comes with many libraries

Python uses

- **Web Development**: Frameworks such as Django and Flask
- **Data Analysis:** Libraries such as NumPy, Pandas, Matplotlib
- **Internet of Things**: Raspberry Pi + Python
- **Web Scraping:** e.g. Scrapy
- **Computer Vision:** e.g. OpenCV
- **Machine Learning:** Sci-kit Learn, NLTK, TensorFlow
- **Game Development:** PyGame

Output in Python

First program

Print (1)

In[ ]:

```python
print("Hello World")

```

Print (1) running

In[ ]:

```python
print("Hello World")

```

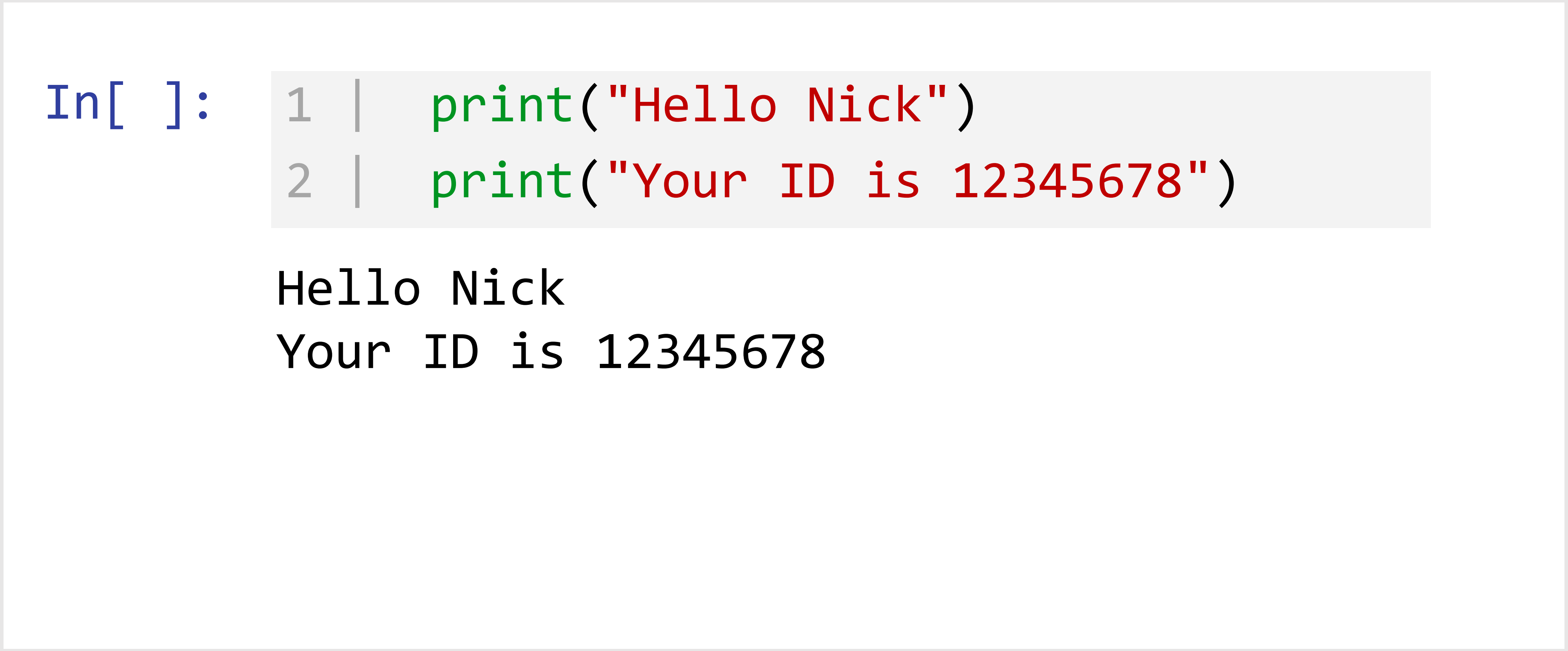
```
Hello World
```

Print (2)

In[ ]:

```python
print("Hello Nick")
print("Your ID is 12345678")
```

Print (2) running



```python
print("Hello Nick")
print("Your ID is 12345678")
```

```
Hello Nick
Your ID is 12345678
```

Variables

What are Variables?

- Variables are a way to store data temporarily in our programs.
- Just like the word variable implies, this data can vary; it can change throughout the course of the program.
- We can store data (value) by assigning it (=) to a memory location. That memory location is then referred to by a variable name.
- The value is stored at a memory address (RAM, which is temporary memory) and can then be accessed through the variable name)
- The opposite of a value changing throughout the program would be a constant, which never changes; it has a fixed value.

Variables in C and Java

- In C, C++, C# and Java, the programmer would have to **first define the data type**, then a meaningful name for the variable:

```
String name = "Nick";
```

data type

name that refers to memory location

assignment operator

value

Variables in Python

- **Python does not require the type to be stated!** Although it is good to be aware of types…
- It is good practice to consider the name of your variable (storage area) and of course what value you will store there:

`name`	`=`	`"Nick"`
name that refers to memory location	assignment operator	value

Types and Names

Numeric types

Types are built as classes (more on this concept later).

The two main numeric types are int and float.

int deals with whole numbers: 10, 60, 0, -9

float deals with decimal numbers: 10.6, 7.25,

Other types

Types are built as classes (more on this concept later).

str (string) can store a combination of characters: "n", "Nick",
It is also possible to use single quotes: 'Nick’

bool can store values True or False

Types

Type	Declaration	Example	Usage
Integer	int	x = 30	Whole numbers
Float	float	x = 3.14	Decimal points
String	str	x = "Hello"	Text
Boolean	bool	x = True	Binary results
NoneType	None	x = None	Empty / Null var

Variable Names

- Important to select meaningful names that describe the value
- Best practice to start variable names with a letter or '_'
- Cannot have spaces in the variable name (Python convention is '_')[1]
- Cannot be a reserved word (int, for, if etc).
- Beware of casing: 'NAME' and 'name' are different variables!

Assignment in Python

Assignment (=)

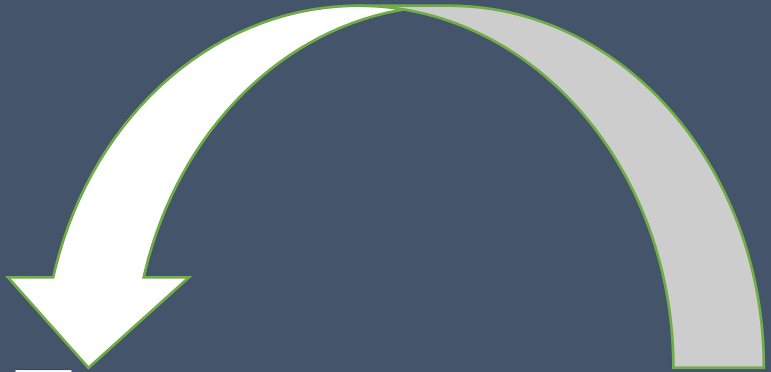
We can store data (value) by assigning it (=) to a memory location. That memory location is then referred to by a variable name.

Remember that the = sign in programming is different to mathematics (which compares two sides of the equation)!

Assignmen t

```
variable = value
```

Assignment



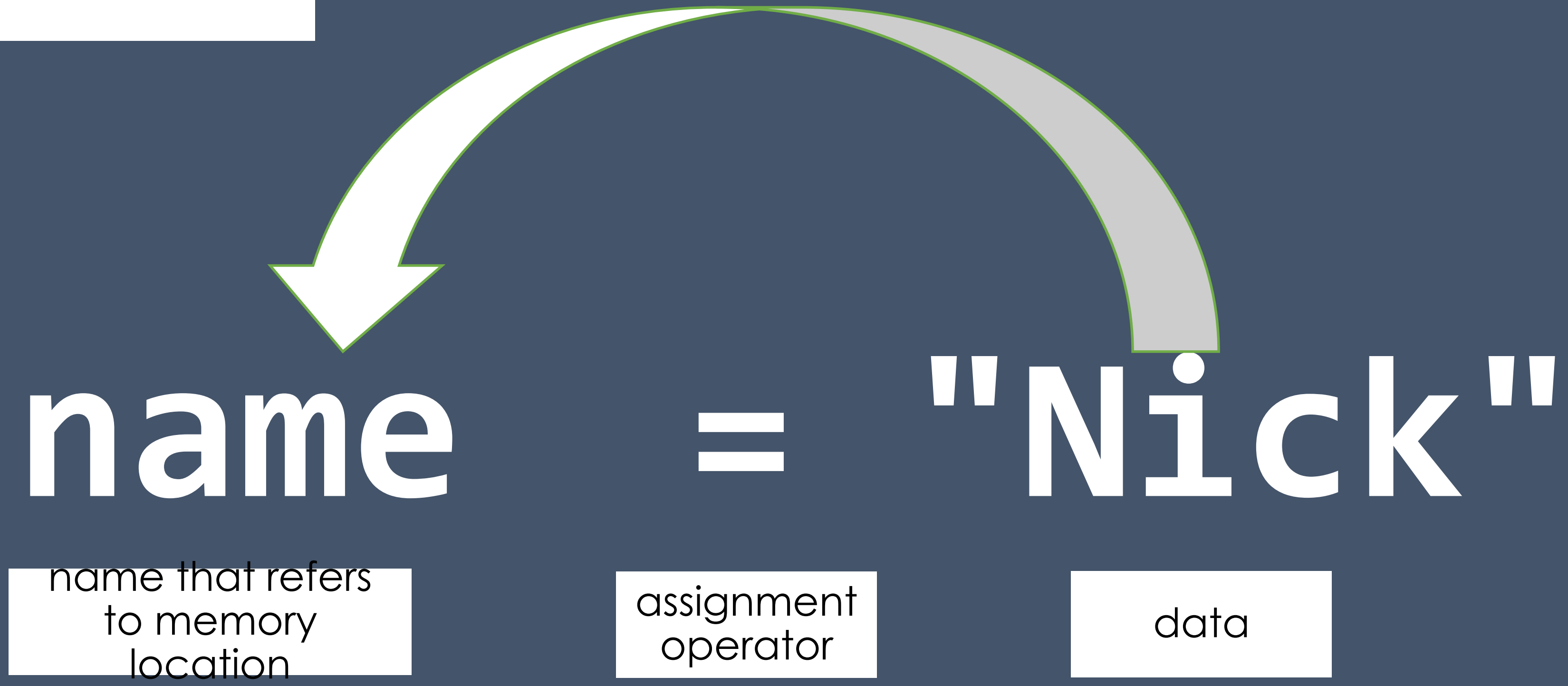
```
variable = value
```

name that refers to memory location

assignment operator

data

Assignment



```
name = "Nick"
```

Variables (1)

In[ ]:

```python
name = "Nick"

```

Variables (1)

In[ ]:

```python
name = "Nick"
print("Hello", name)
```

Variables (1)

```
In[ ]:
```

```python
name = "Nick"
print("Hello", name)
```

```
Hello Nick
```

Concatenation

In[ ]:

```python
name = "Nick"
print("Hello" + name)
```

Concatenation

```
In[ ]:
name = "Nick"
print("Hello" + name)
```

```
HelloNick
```

Concatenation

```
In[ ]:  name = "Nick"
        print("Hello " + name)
```

Concatenation

```
In[ ]:  name = "Nick"
        print("Hello " + name)
```

```
Hello Nick
```

Alternatively

In[ ]:

```python
name = "Nick"
name
```

Alternatively

```
In[ ]:   name = "Nick"
         name

         Nick
```

Variables (2)

In[ ]:

```
name = "Nick"
age = 30
print("Hello", name, "you are", age)
```

Variables (2)

In[ ]:

```python
name = "Nick"
age = 30
print("Hello", name, "you are", age)
```

```
Hello Nick you are 30
```

String formatting

d, i	Integer
u	Unsigned Integer
f	Floating point as m.ddddd
e	Floating point as m.ddddde+/-xx
E	Floating point as m.dddddE+/-xx
g	Use %e for exponents <-4 or >+5, otherwise use %f
c	Single character
s	String, or any object that be converted via **str()**
%	Insert a literal % character

String formatting

```
In[ ]: print("%s is %d years old" % (name, age) )
```

```
Nick is 30 years old
```

More on Types

Static and Dynamic

Python is dynamically typed (type checked at run-time). Variables can be initially assigned a value of one type, and then be reassigned a value of a different type!

Statically typed languages (such as C, C++, C# and Java, which check type at compile-time) only allow values of the same type to be assigned.

Redefining age

In[ ]:

```python
age = 30
print("You are", age)
age = "Nick"
print("You are", age)
```

int and str

In[ ]:

```python
age = 30
print("You are", age)
age = "Nick"
print("You are", age)
```

```
You are 30
You are Nick
```

type() function

In[ ]:

```
age = 30
print(type(age))
name = "Nick"
print(type(name))
```

type() function

In[ ]:

```python
age = 30
print(type(age))
name = "Nick"
print(type(name))
```

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

Type Casting

Arithmetic

```
In[ ]:
first_mark = 60
second_mark = "40"
first_mark + second_mark

```

Arithmetic

```
In[ ]:
first_mark = 60
second_mark = "40"
first_mark + second_mark

```

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

Casting

```
In[ ]:
first_mark = 60
second_mark = int("40")
first_mark + second_mark

```

Casting

In[ ]:

```python
first_mark = 60
second_mark = int("40")
first_mark + second_mark

```

```
100
```

Casting

In[ ]:

```
age = int(30)
print(age)
age = float(30)
print(age)
```

Casting

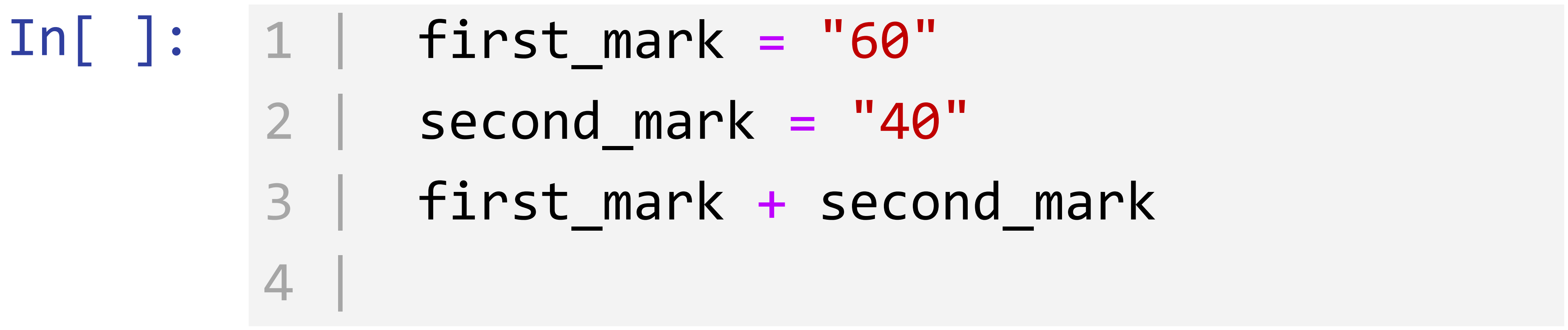
In[ ]:

```python
age = int(30)
print(age)
age = float(30)
print(age)
```

```
30
30.0
```

Quick Quiz!

What will the output be?


In[ ]:

```python
first_mark = "60"
second_mark = "40"
first_mark + second_mark

```

Str concat vs arithmetic

```
In[ ]:
first_mark = "60"
second_mark = "40"
first_mark + second_mark

```

```
'6040'
```

What will the output be?

```
In[ ]:
str1 = "Nick is"
str2 = 40
str3 = "years old"
print(str1 + str2 + str3)
```

Mixture of types

```
In[ ]:
str1 = "Nick is"
str2 = 40
str3 = "years old"
print(str1 + str2 + str3)
```

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

Commas instead of +

In[ ]:

```python
str1 = "Nick is"
str2 = 40
str3 = "years old"
print(str1, str2, str3)
```

```
Nick is 40 years old
```

Case Sensitivity

Case Sensitive

In[ ]:

```python
age = 30
print(age)
Age = 35
print(Age)
```

Case Sensitive

```
In[ ]:
age = 30
print(age)
Age = 35
print(Age)
```

```
30
35
```

Input in Python

Input - strings

In[ ]:

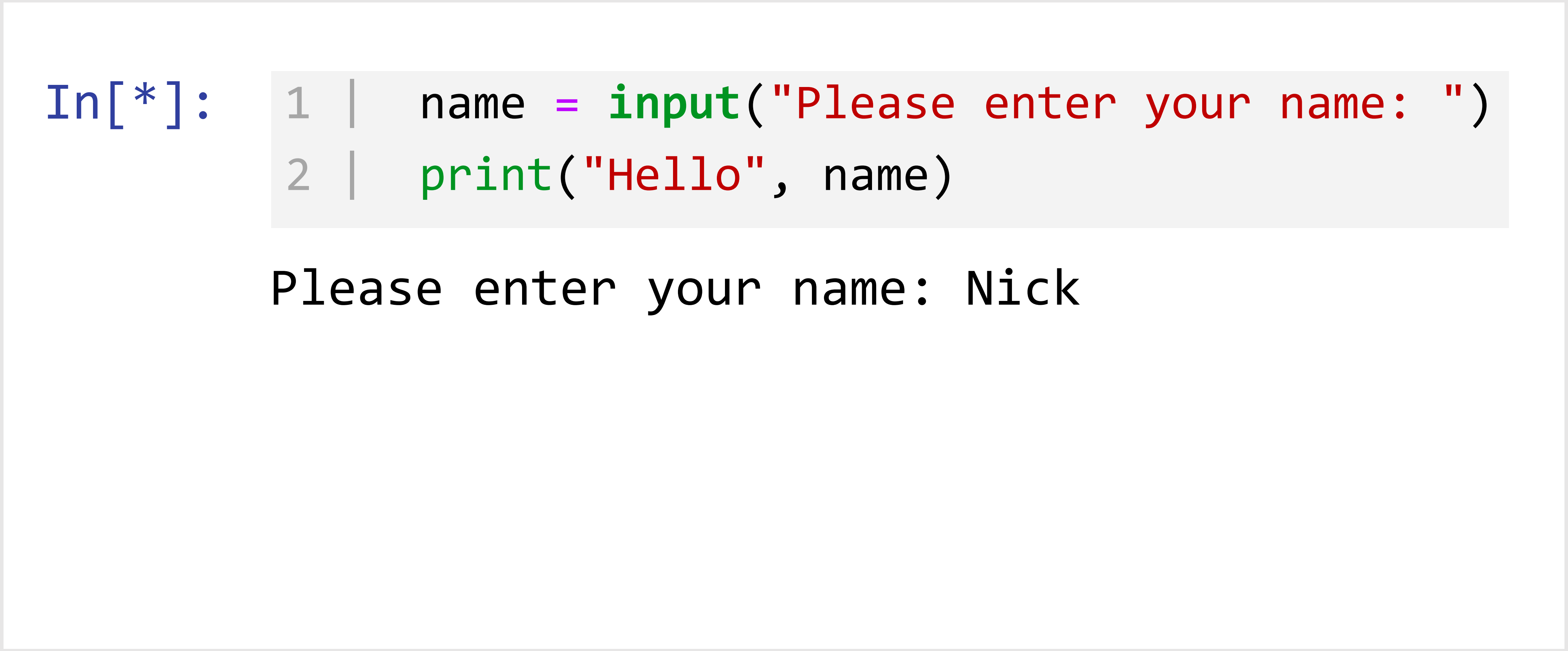
```python
name = input("Please enter your name: ")
print("Hello", name)
```

Input - running

```
In[*]:
name = input("Please enter your name: ")
print("Hello", name)
```

```
Please enter your name:
```

Input - running



```
In[*]:
name = input("Please enter your name: ")
print("Hello", name)
```

```
Please enter your name: Nick
```

Input - running

In[ ]:

```python
name = input("Please enter your name: ")
print("Hello", name)
```

```
Please enter your name: Nick
Hello Nick
```

Legacy Python

Python 2 used to have a function called **raw_input()**.

However, this was confusing to use, so Python 3 introduced a simplified **input()** method.

Do not attempt to use the **input()** function if you are using Python 2, as values return will depend on names of variables used… this caused problems and therefore was largely avoided.

Input - numbers

In[ ]:

```python
age = int(input("Please enter your age: "))
print("You are", age, "years old")
```

Input - numbers

```
In[*]:  age = int(input("Please enter your age: "))
        print("You are", age, "years old")
```

```
Please enter your age:
```

Input - numbers

```
In[*]:  age = int(input("Please enter your age: "))
        print("You are", age, "years old")
```

```
Please enter your age: 30
```

Input - numbers

```
In[ ]:
age = int(input("Please enter your age: "))
print("You are", age, "years old")
```

```
Please enter your age: 30
You are 30 years old
```

Comments in Python

Comments

Code comments are useful for documentation purposes. They are known as docstrings in Python.

They can also be used for beginners to test and debug their code.

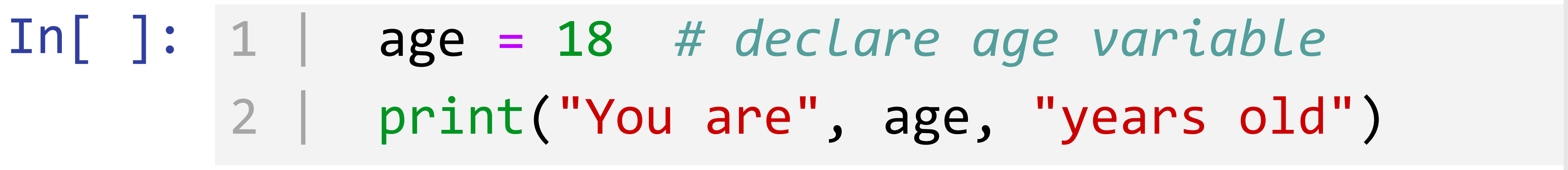
'""' for block summary comments (three quotation marks)

# for single line comments

Comments

```
In[ ]:
# age = 18
# print("You are", age, "years old")
```

Comments



```
In[ ]:
age = 18  # declare age variable
print("You are", age, "years old")
```

Comments

```
In[ ]:
age = 18  # declare age variable
print("You are", age, "years old")
```

```
You are 18 years old
```

Comments

In[ ]:

```
"""
First line
This is a block comment for function
Third line
"""
```

Comments

```
In[ ]:
def get_input():
  """This is a class method comment."""
  return input("Please enter your name: ")

```

Next Lecture

Sequence, Selection, Iteration

- **Sequence** mandates that statements be executed in order (line by line)
- **Selection** (conditional) statements will execute a **block of code once** when the condition is true
- **Iteration** allows us to **repeat** statements within a block **whilst** the condition is **true**