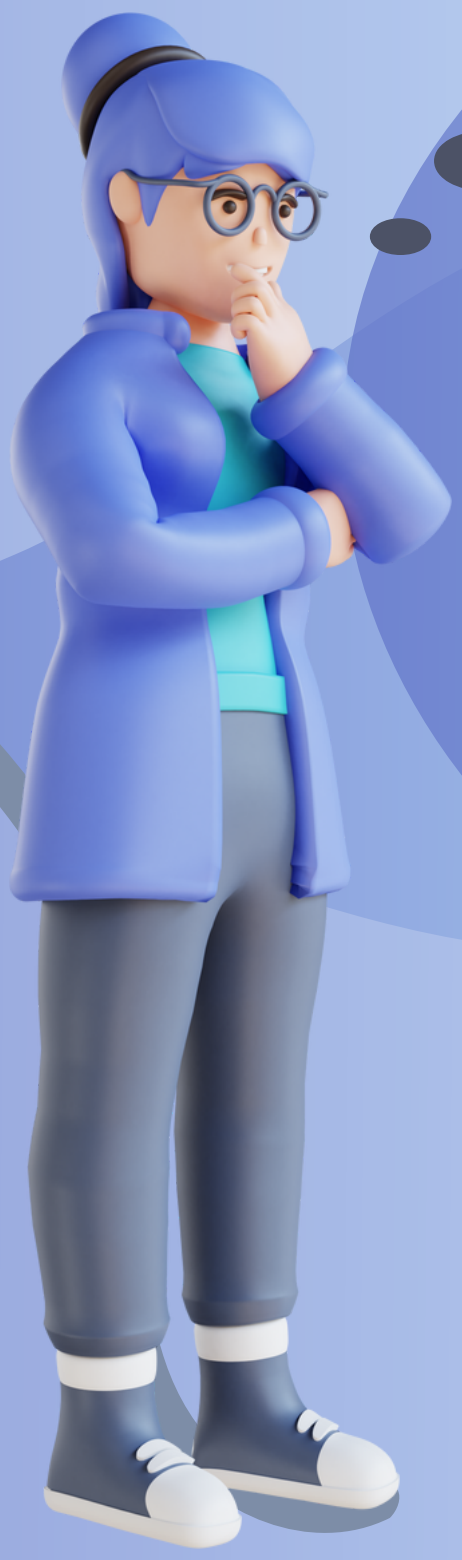
DANIAL SOLEIMANY

1-16

OBJECT-ORIENTED PROGRAMMING



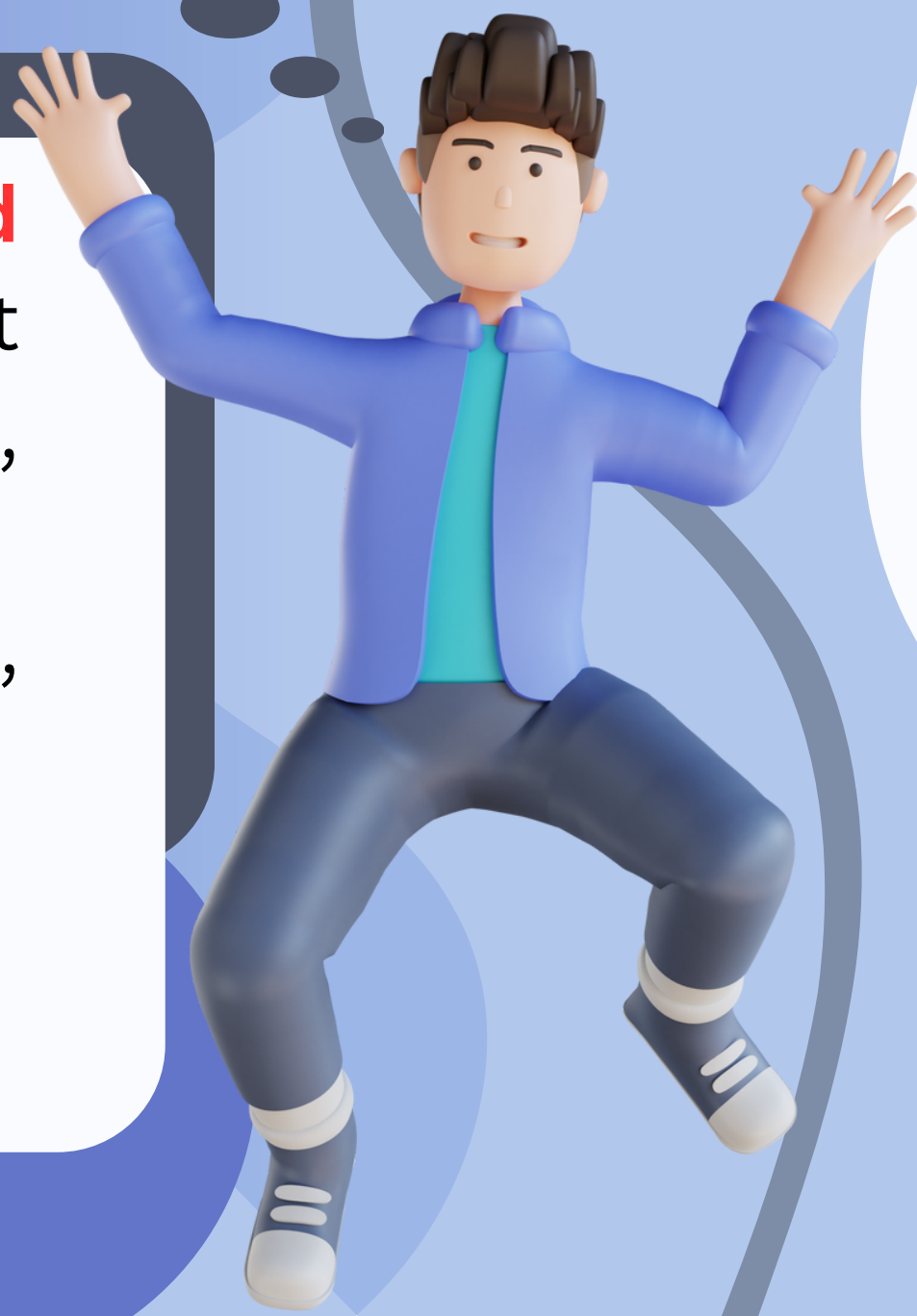
LET'S TALK ABOUT OOP IN PYTHON

DANIAL SOLEIMANY

LET'S START

CLASSES/OBJECTS

Python is an **object oriented** programming language .Almost everything in Python is an object, with its properties and methods.
A Class is like an object constructor, or a "blueprint" for creating objects.



CREATE A CLASS

Create a class named "Person", with a property named "favorite":

EX.

```
class Person:
    favorite = "Programming"
```

CREATE OBJECT

Now we can use the class named Person to create objects

```
# Create an object named p1, and print the value of "favorite"
```

EX.

```
p1 = Person()
print(p1.favorite)
```

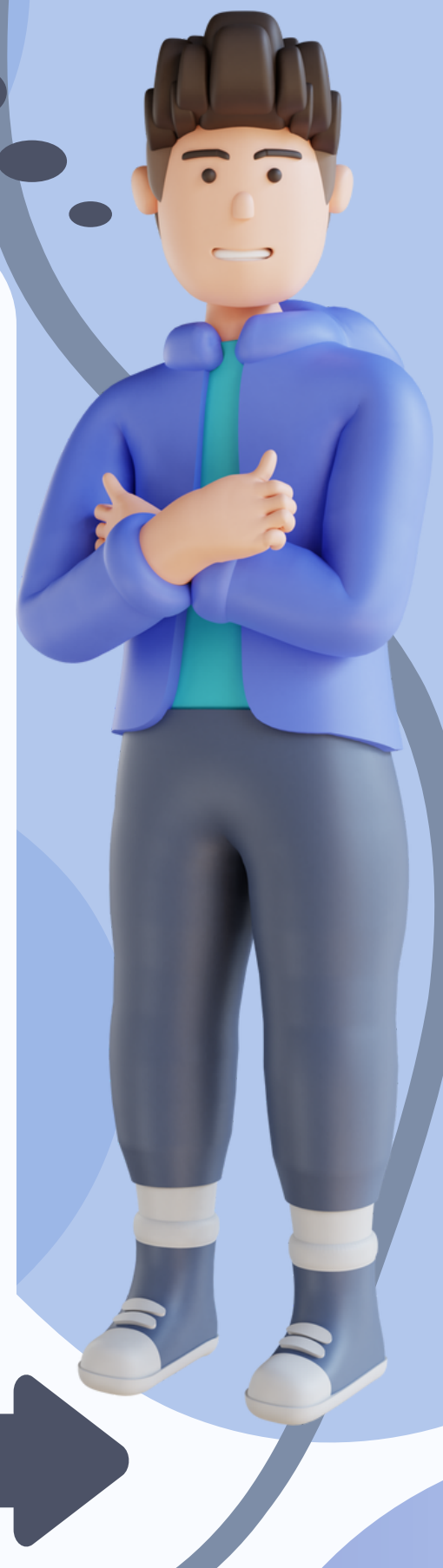
OUTPUT

'Programming'

GREAT JOB!

THE __INIT__() FUNCTION

The examples above are classes and objects in their simplest form, and are not really useful in real life applications. To understand the meaning of classes we have to understand the built-in __init__() function. All classes have a function called __init__(), which is always executed when the class is being initiated. Use the __init__() function to assign values to object properties, or other operations that are necessary to do when the object is being created.



DANIAL SOLEIMANY

THE SELF PARAMETER

The self parameter is a reference to the **current** instance of the class, and is used to **access** variables that belongs to the class.

It does **not** have to be named self , you can call it whatever you like, but it has to be the **first** parameter of any function in the class

Create a class named Person, use the __init__() function to assign values for name and age.

EX.

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age
p1 = Person("Danial", 23)
print(p1.name)
print(p1.age)
```

OUTPUT

```
'Danial'
23
```

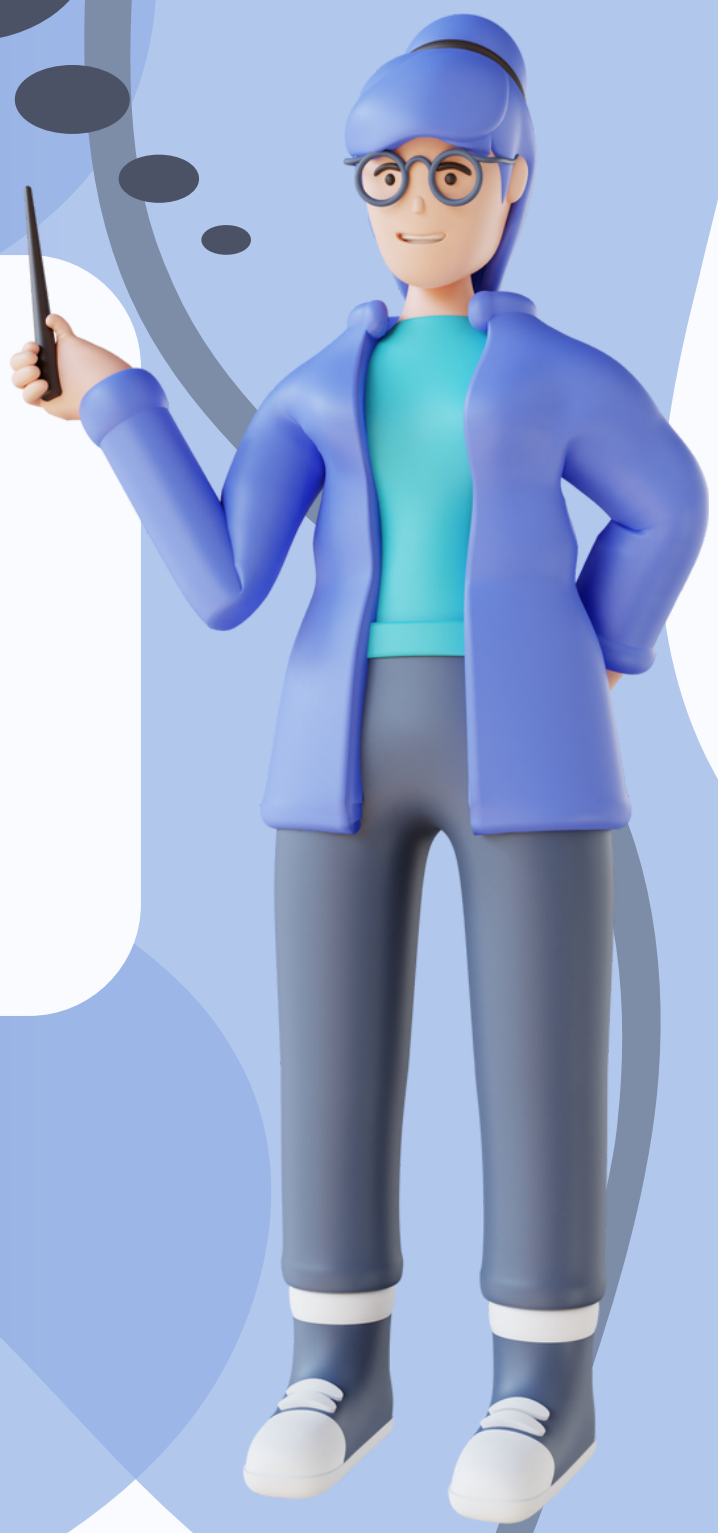
Note: The __init__() function is called automatically every time the class is being used to create a new object.

DANIAL SOLEIMANY

WANNA MORE?

THE __STR__() FUNCTION

The __str__() function controls what should be returned when the **class object** is represented as a string.
If the __str__() function is not set, the string representation of the object is returned.



The string representation of an object WITHOUT the __str__() function

EX.

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age
p1 = Person("Danial", 23)
print(p1)
```

OUTPUT

```
<__main__.Person object at 0x0000022CCF3B7D60>
```

The string representation of an object WITH the __str__() function

EX 2.

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age
    def __str__(self):
        return f"{self.name}, {self.age}"
p1 = Person("Danial", 23)
print(p1)
```

OUTPUT

```
Danial, 23
```

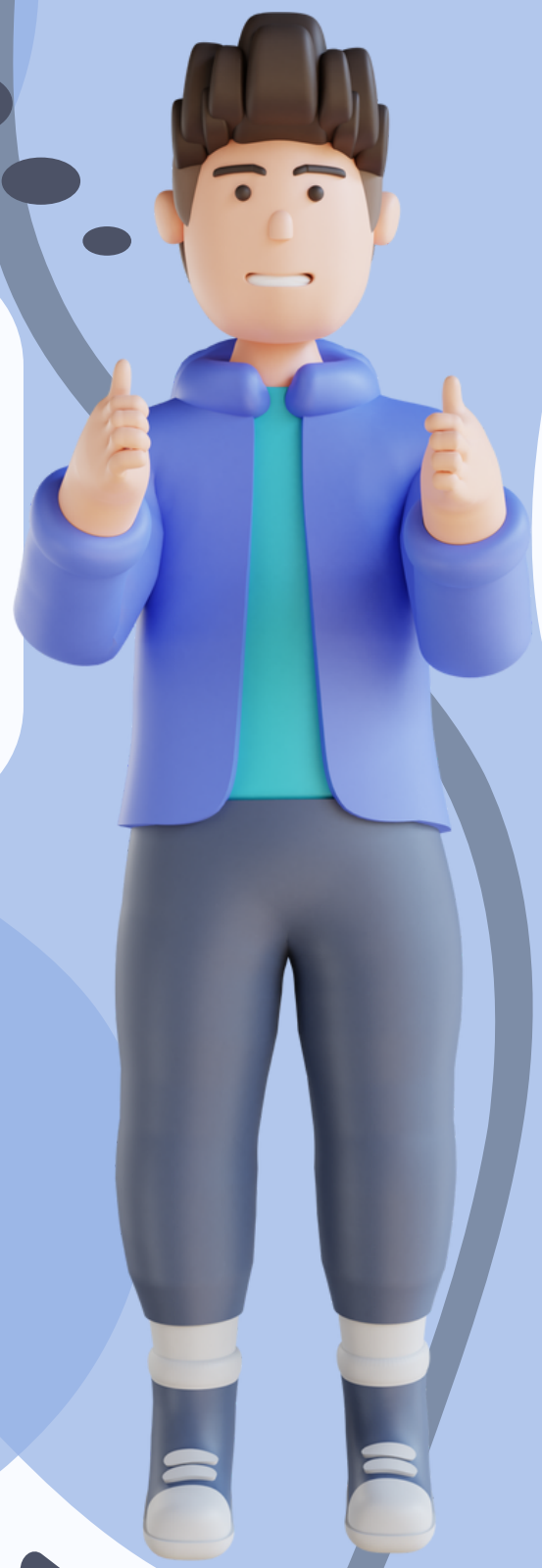
DANIAL SOLEIMANY

EASY!

11-16

OBJECT METHODS

Objects can also contain methods. Methods in objects are functions that belong to the object.



LET'S CREATE A METHOD IN THE PERSON CLASS!

```
# Insert a function that prints a greeting, and execute it on the p1 object
```

EX

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age
    def myfunc(self):
        print("Hello my name is " + self.name)
p1 = Person("Danial", 23)
p1.myfunc()
```

OUTPUT

```
'Hello my name is Danial'
```

Note: The self parameter is a reference to the current instance of the class, and is used to access variables that belong to the class.

Use the words mysillyobject and abc instead of self

EX 2.

```
class Person:
    def __init__(mysillyobject, name, age):
        mysillyobject.name = name
        mysillyobject.age = age
    def myfunc(abc):
        print("Hello my name is " + abc.name)
p1 = Person("Danial", 23)
p1.myfunc()
```

OUTPUT

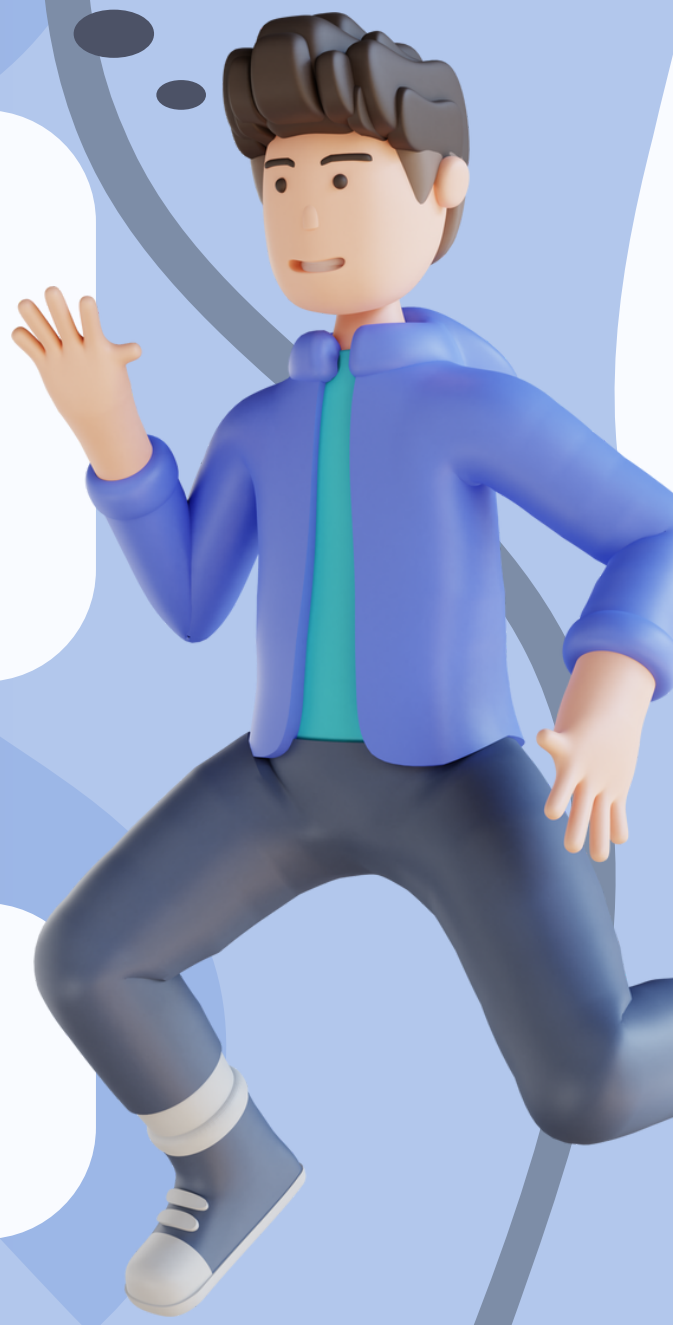
'Hello my name is Danial'

DANIAL SOLEIMANY

HAHA!

MODIFY OBJECT PROPERTIES

```
p1.age = 40
print(p1.age)
p1.name = "Jack"
print(p1.name)
```



OUTPUT

```
40  # age value changed from 23 to 40
'Jack'  # name value changed from 'Danial' to 'Jack'
```

DELETE OBJECT PROPERTIES

```
# You can delete properties on objects by using the del keyword
del p1.age
print(p1.age)
```

OUTPUT

```
AttributeError: 'Person' object has no attribute 'age'
```

DELETE OBJECTS

```
# You can delete objects by using the del keyword
del p1
print(p1)
```

OUTPUT

```
NameError: name 'p1' is not defined
```

DANIAL SOLEIMANY

16-16

FOLLOW FOR MORE