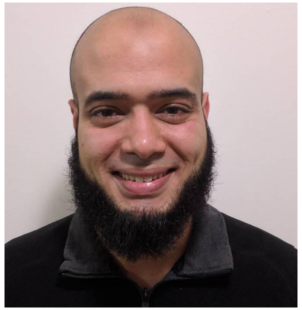
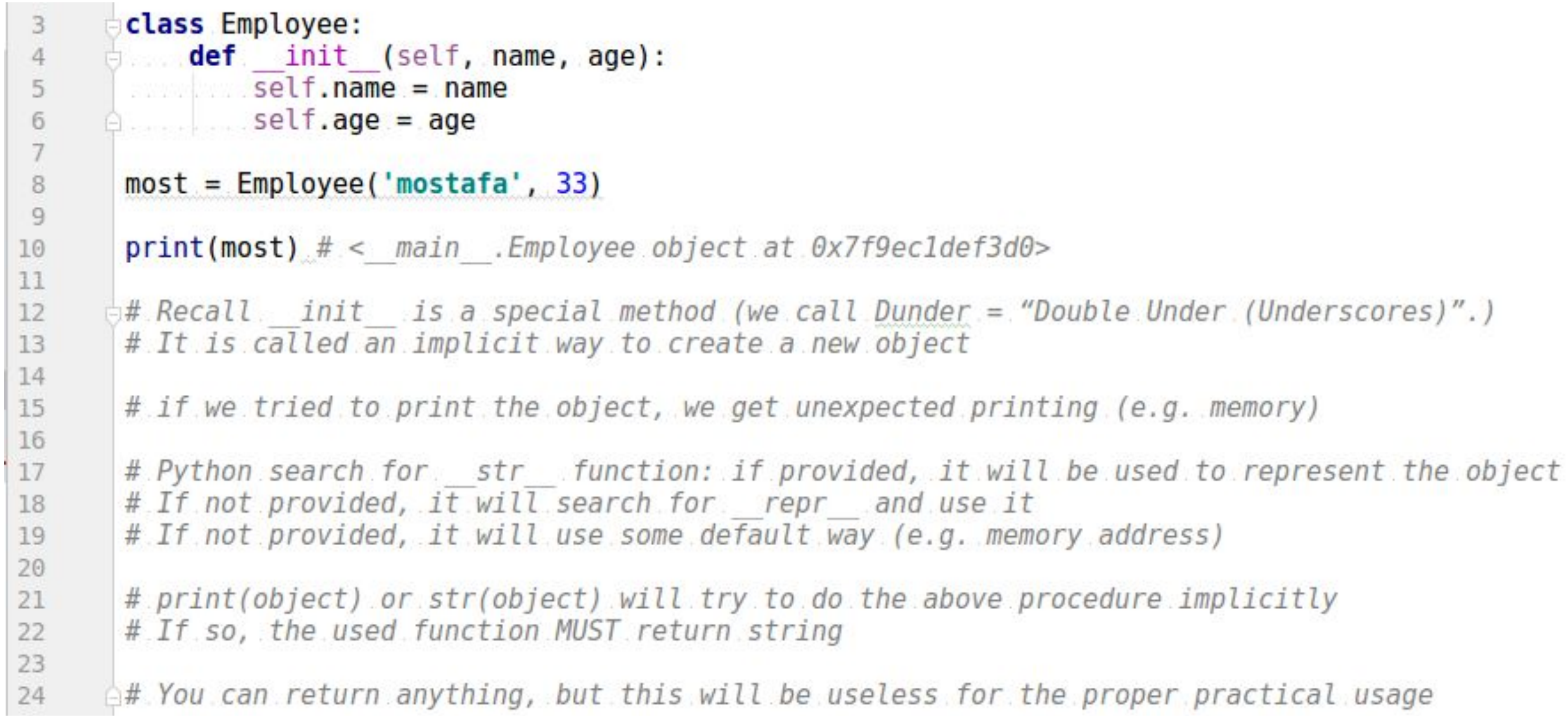
Python Programming
str and repr for Class

Mostafa S. Ibrahim
Teaching, Training and Coaching since more than a decade!

Artificial Intelligence & Computer Vision Researcher
PhD from Simon Fraser University - Canada
Bachelor / Msc from Cairo University - Egypt
Ex-(Software Engineer / ICPC World Finalist)

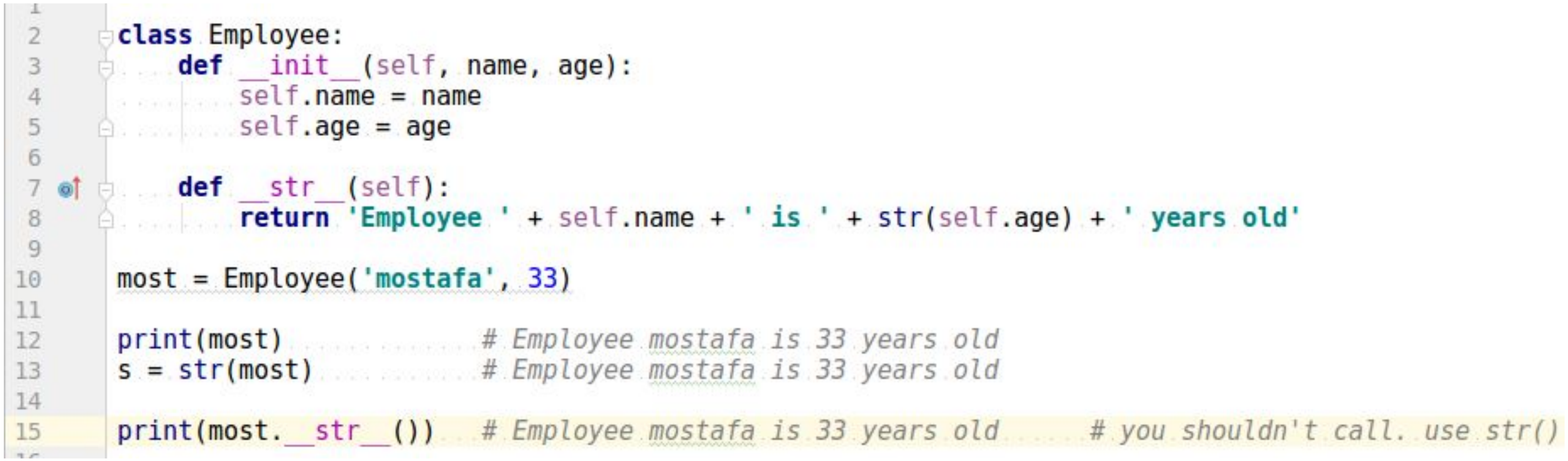


Dunder init



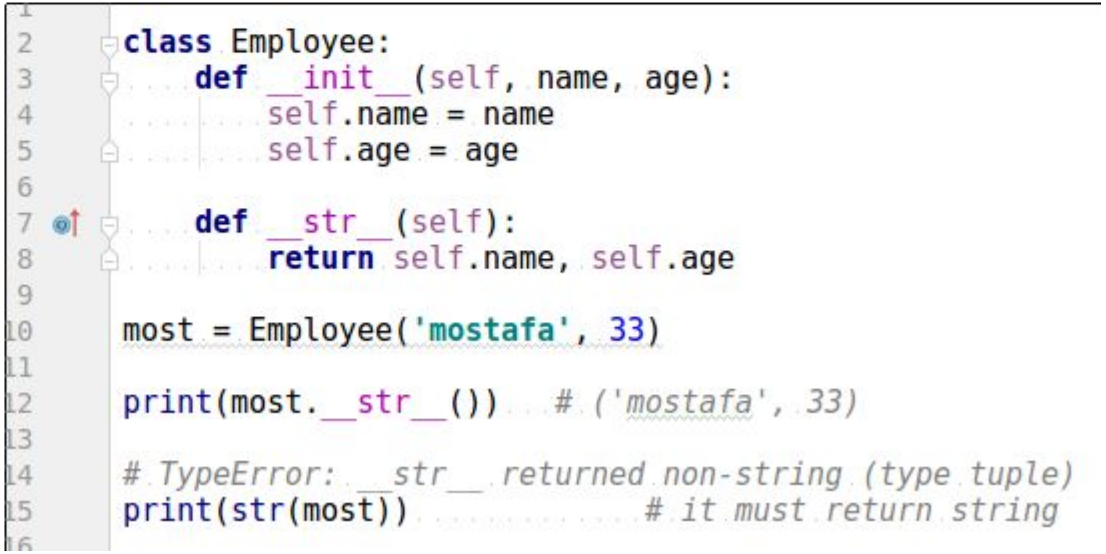
```python
class Employee:
    def __init__(self, name, age):
        self.name = name
        self.age = age

most = Employee('mostafa', 33)

print(most) # <__main__.Employee object at 0x7f9ec1def3d0>

# Recall __init__ is a special method (we call Dunder = “Double Under (Underscores)”.)
# It is called an implicit way to create a new object

# if we tried to print the object, we get unexpected printing (e.g. memory)

# Python search for __str__ function: if provided, it will be used to represent the object
# If not provided, it will search for __repr__ and use it
# If not provided, it will use some default way (e.g. memory address)

# print(object) or str(object) will try to do the above procedure implicitly
# If so, the used function MUST return string

# You can return anything, but this will be useless for the proper practical usage
```

Dunder str



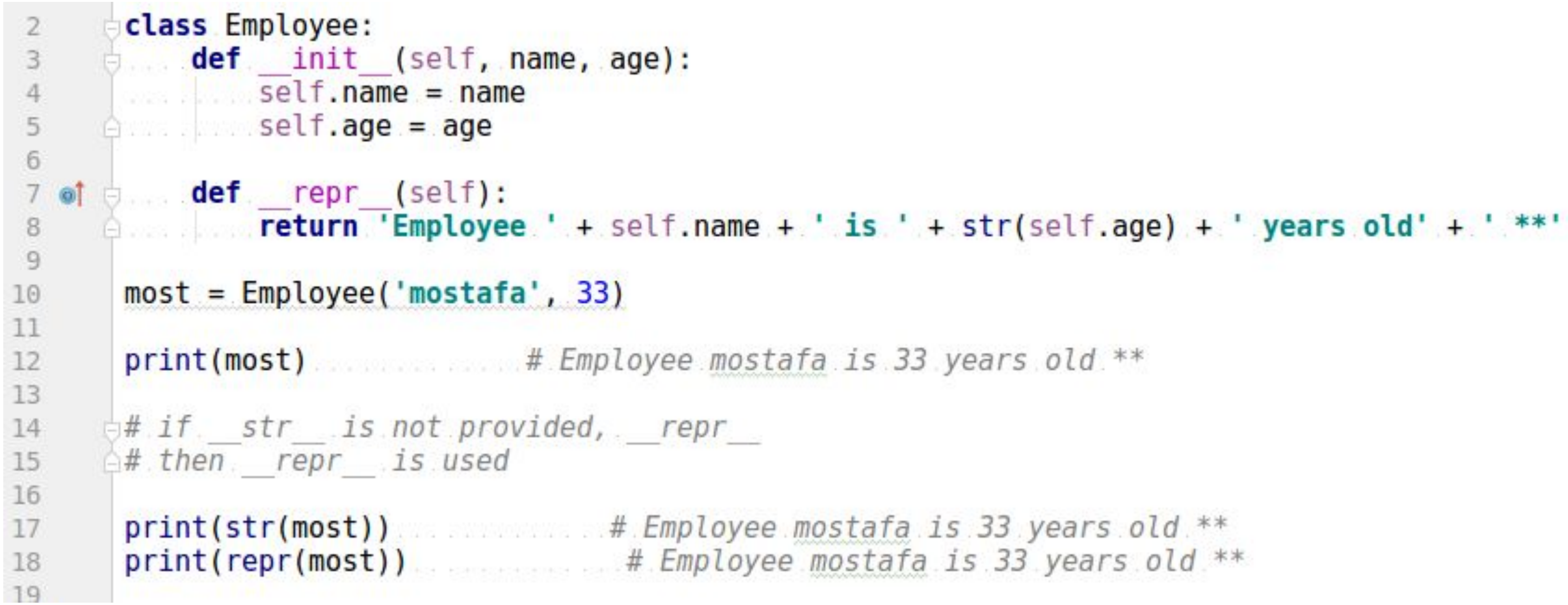
```python

class Employee:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def __str__(self):
        return 'Employee ' + self.name + ' is ' + str(self.age) + ' years old'

most = Employee('mostafa', 33)

print(most)             # Employee mostafa is 33 years old
s = str(most)           # Employee mostafa is 33 years old

print(most.__str__())   # Employee mostafa is 33 years old      # you shouldn't call. use str()
```

Dunder str: must return string



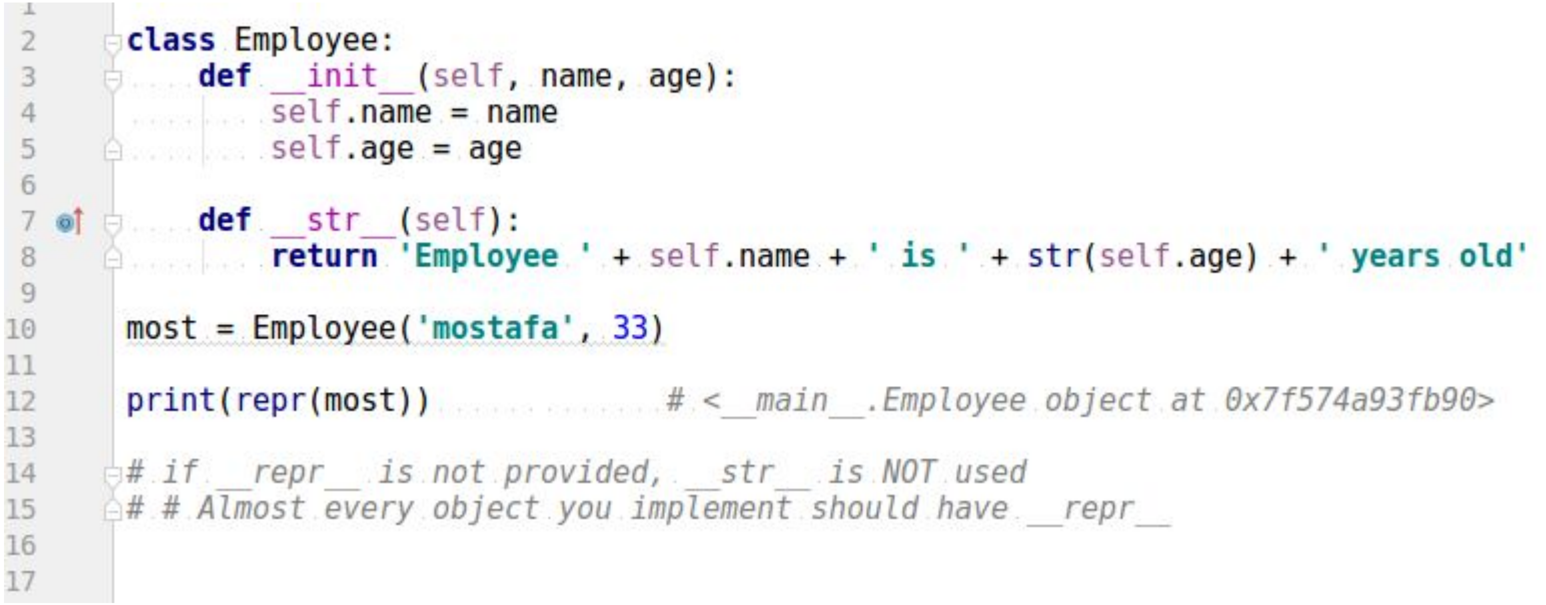
```
class Employee:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def __str__(self):
        return self.name, self.age

most = Employee('mostafa', 33)

print(most.__str__())   # ('mostafa', 33)

# TypeError: __str__ returned non-string (type tuple)
print(str(most))             # it must return string
```

Dunder repr



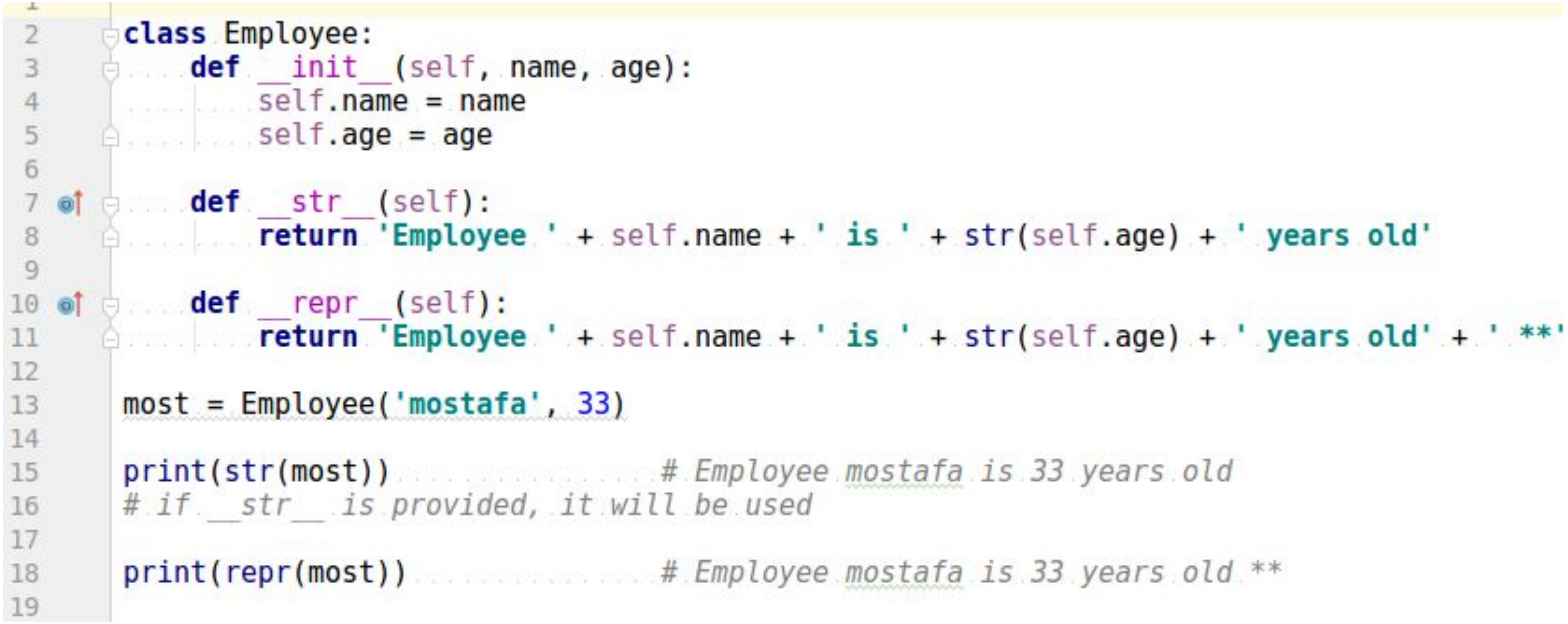
```
class Employee:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def __repr__(self):
        return 'Employee ' + self.name + ' is ' + str(self.age) + ' years old' + ' **'

most = Employee('mostafa', 33)

print(most)                # Employee mostafa is 33 years old **

# if __str__ is not provided, __repr__
# then __repr__ is used

print(str(most))            # Employee mostafa is 33 years old **
print(repr(most))            # Employee mostafa is 33 years old **
```

Dunder repr: if not provided, NO call for str

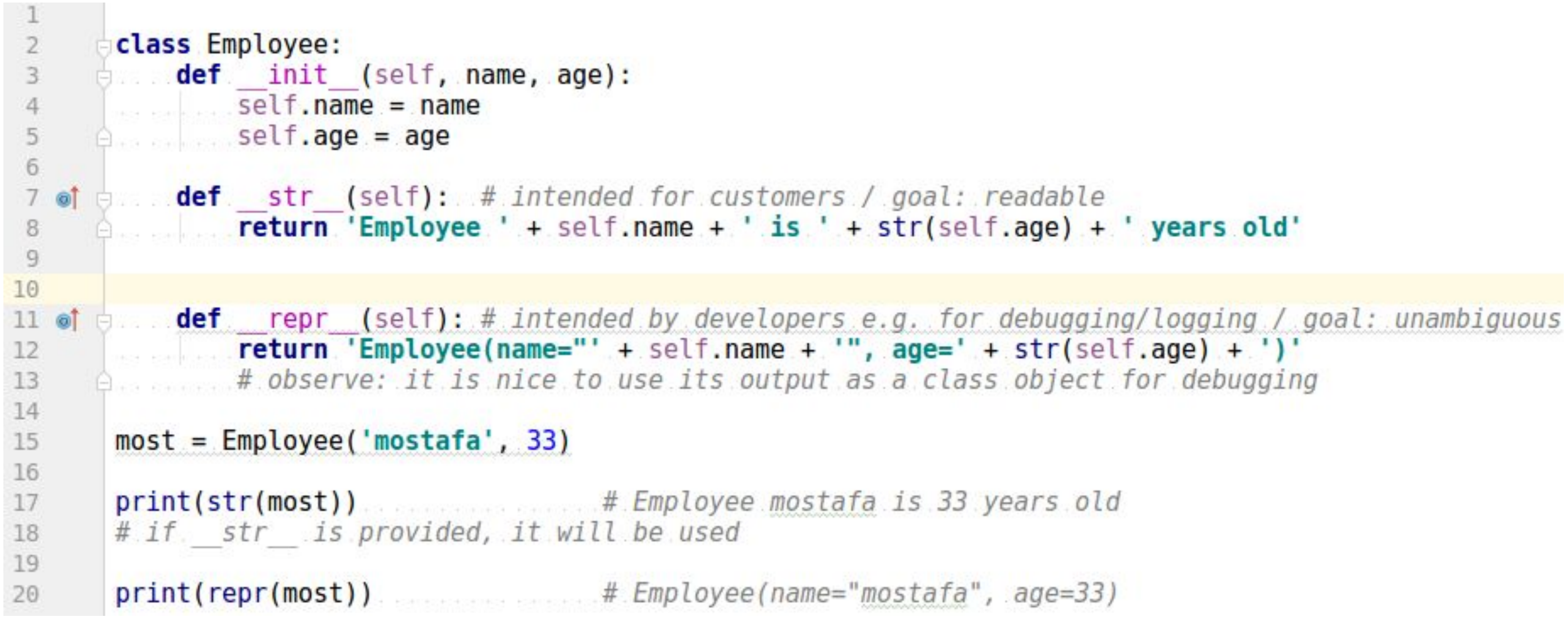


```python

class Employee:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def __str__(self):
        return 'Employee ' + self.name + ' is ' + str(self.age) + ' years old'

most = Employee('mostafa', 33)

print(repr(most))           # <__main__.Employee object at 0x7f574a93fb90>

# if __repr__ is not provided, __str__ is NOT used
# # Almost every object you implement should have __repr__


```

If provided, use it

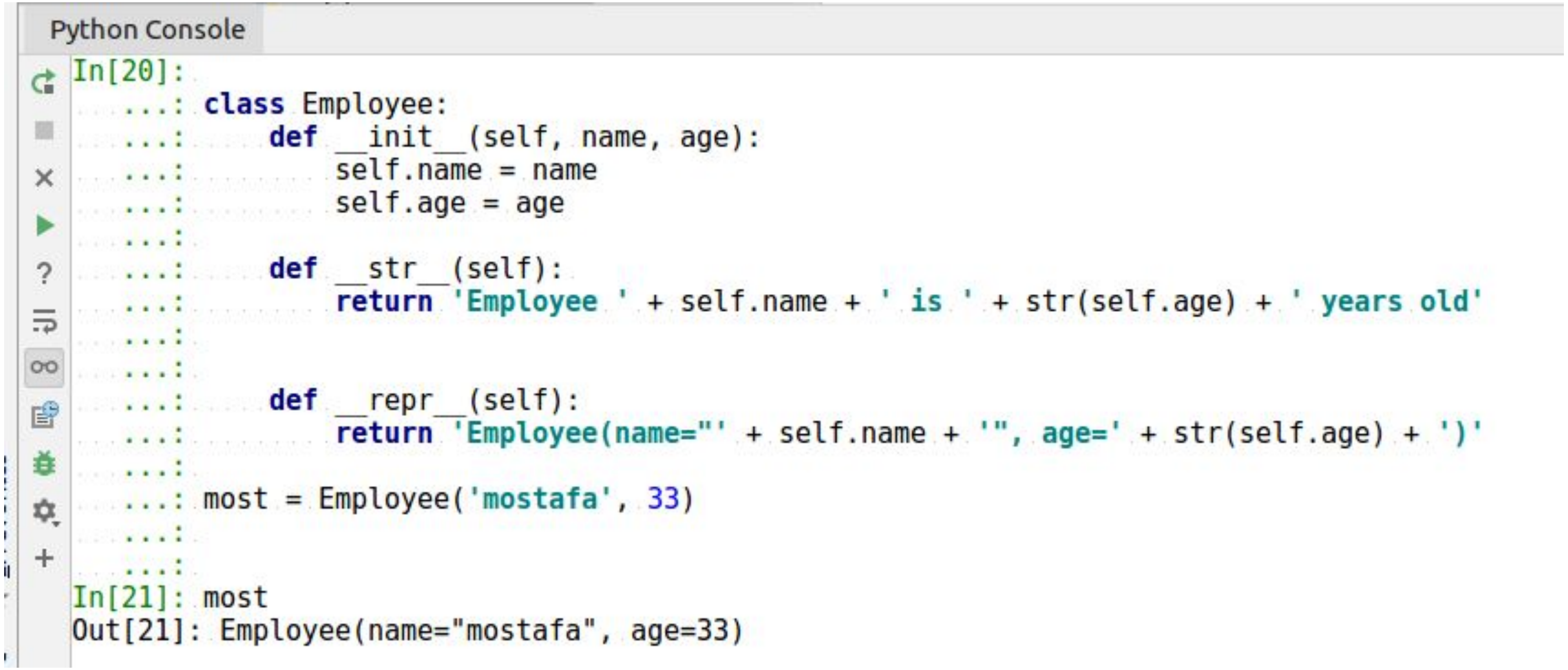

```python

class Employee:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def __str__(self):
        return 'Employee ' + self.name + ' is ' + str(self.age) + ' years old'

    def __repr__(self):
        return 'Employee ' + self.name + ' is ' + str(self.age) + ' years old' + ' **'

most = Employee('mostafa', 33)

print(str(most))                # Employee mostafa is 33 years old
# if __str__ is provided, it will be used

print(repr(most))               # Employee mostafa is 33 years old **

```

Why both str and repr?



```python

class Employee:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def __str__(self):  # intended for customers / goal: readable
        return 'Employee ' + self.name + ' is ' + str(self.age) + ' years old'

    def __repr__(self): # intended by developers e.g. for debugging/logging / goal: unambiguous
        return 'Employee(name="' + self.name + '", age=' + str(self.age) + ')'
        # observe: it is nice to use its output as a class object for debugging

most = Employee('mostafa', 33)

print(str(most))                # Employee mostafa is 33 years old
# if __str__ is provided, it will be used

print(repr(most))               # Employee(name="mostafa", age=33)
```

From Console



```
In[20]: 
   ...: class Employee:
   ...:     def __init__(self, name, age):
   ...:         self.name = name
   ...:         self.age = age
   ...: 
   ...:     def __str__(self):
   ...:         return 'Employee ' + self.name + ' is ' + str(self.age) + ' years old'
   ...: 
   ...: 
   ...:     def __repr__(self):
   ...:         return 'Employee(name="' + self.name + '", age=' + str(self.age) + ')'
   ...: 
   ...: most = Employee('mostafa', 33)
   ...: 
   ...: 
In[21]: most
Out[21]: Employee(name="mostafa", age=33)
```

"Acquire knowledge and impart it to the people."

"Seek knowledge from the Cradle to the Grave."