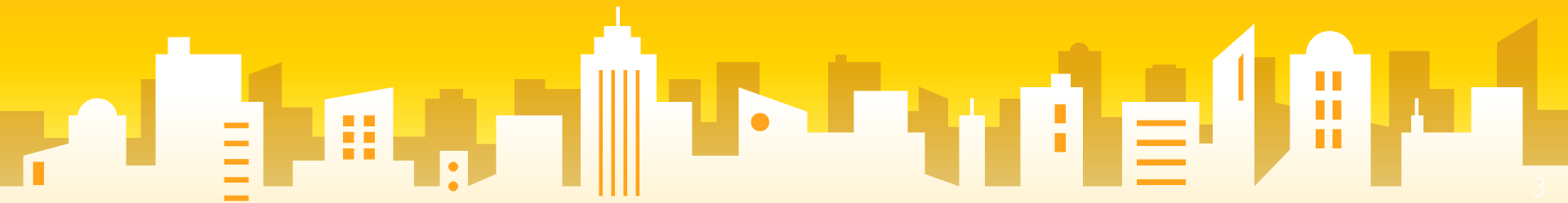
@Property in Python

First Lesson. (Most Important)



Python @property

- Python has a great concept called property which makes the life of an object oriented programmer much simpler.

- Before defining and going into details of what @property is, let us first build an intuition on why it would be needed in the first place.

An Example To Begin With

Let us assume that you decide to make a class that could store the temperature in degree Celsius. It would also implement a method to convert the temperature into degree Fahrenheit. One way of doing this is as follows.

Example

```
class Celsius:
    def __init__(self, temperature = 0):
        self.temperature = temperature

    def to_fahrenheit(self):
        return (self.temperature * 1.8) + 32
```

Variable Manipulation

```
>>> # create new object
>>> man = Celsius()

>>> # set temperature
>>> man.temperature = 37

>>> # get temperature
>>> man.temperature
37

>>> # get degrees Fahrenheit
>>> man.to_fahrenheit()
98.60000000000001
```

Using Getters and Setters

```
class Celsius:
    def __init__(self, temperature = 0):
        self.set_temperature(temperature)
    def to_fahrenheit(self):
        return (self.get_temperature() * 1.8) + 32
    def get_temperature(self):
        return self._temperature
    def set_temperature(self, value):
        if value < -273:
            raise ValueError("Temperature below -273 is not possible")
        self._temperature = value
```

Execution

```
>>> c = Celsius(-277)
Traceback (most recent call last):
...
ValueError: Temperature below -273 is not possible

>>> c = Celsius(37)
>>> c.get_temperature()
37
>>> c.set_temperature(10)

>>> c.set_temperature(-300)
Traceback (most recent call last):
...
ValueError: Temperature below -273 is not possible
```

The Power of @property

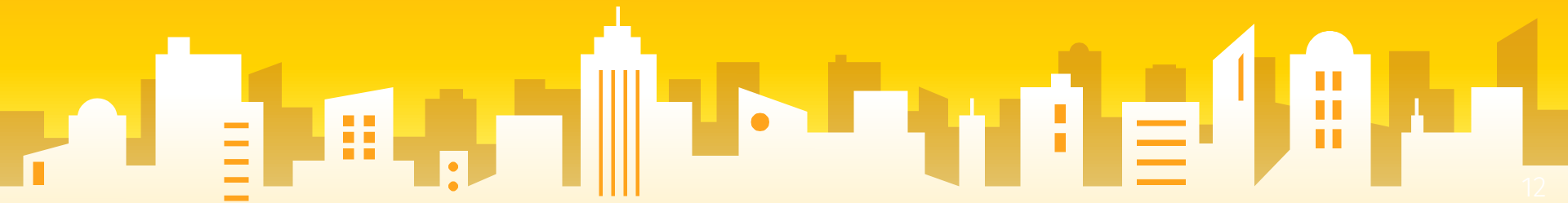
```
class Celsius:
    def __init__(self, temperature = 0):
        self.temperature = temperature

    def to_fahrenheit(self):
        return (self.temperature * 1.8) + 32

    def get_temperature(self):
        print("Getting value")
        return self._temperature

    def set_temperature(self, value):
        if value < -273:
            raise ValueError("Temperature below -273 is not possible")
        print("Setting value")
        self._temperature = value

    temperature = property(get_temperature,set_temperature)
```

Another Way for Property

```
class Celsius:
    def __init__(self, temperature = 0):
        self._temperature = temperature

    def to_fahrenheit(self):
        return (self.temperature * 1.8) + 32

    @property
    def temperature(self):
        print("Getting value")
        return self._temperature

    @temperature.setter
    def temperature(self, value):
        if value < -273:
            raise ValueError("Temperature below -273 is not possible")
        print("Setting value")
        self._temperature = value
```

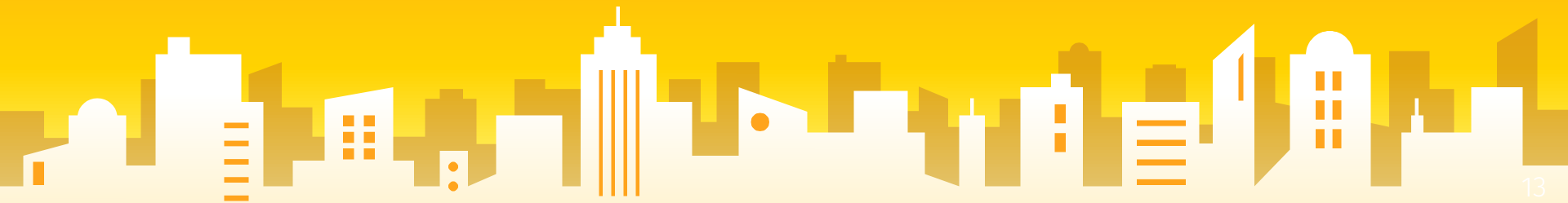
Property for temprature.

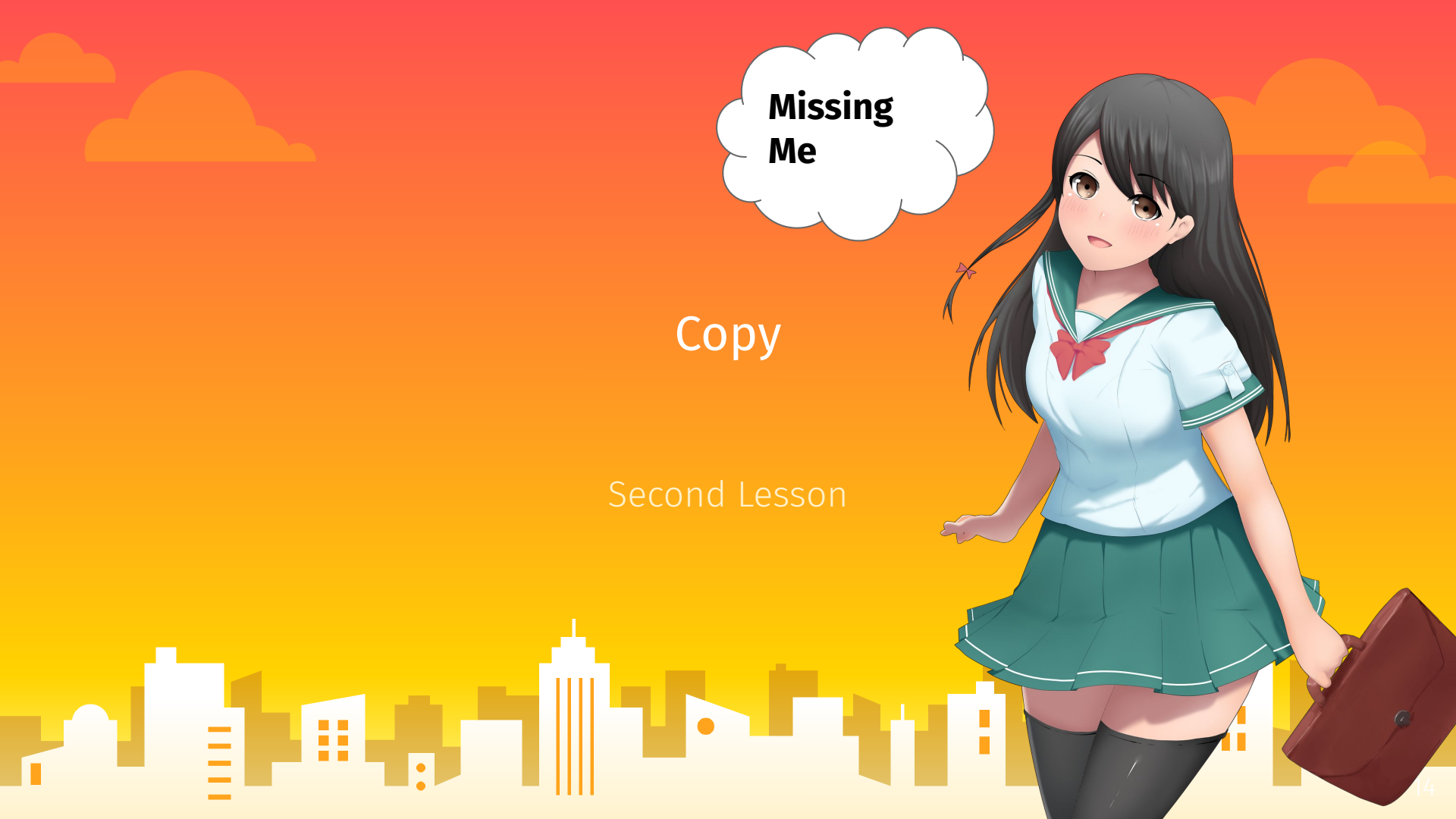
Assignment



Property for mathematics

Assignment





Copy

Second Lesson

Copy an Object in Python

- In Python, we use = operator to create a copy of an object. You may think that this creates a new object; it doesn't. It only creates a new variable that shares the reference of the original object.

- Let's take an example where we create a list named old_list and pass an object reference to new_list using = operator.

Example

```
old_list = [[1, 2, 3], [4, 5, 6], [7, 8, 'a']]
new_list = old_list

new_list[2][2] = 9

print('Old List:', old_list)
print('ID of Old List:', id(old_list))

print('New List:', new_list)
print('ID of New List:', id(new_list))
```

Shallow Copy

- A shallow copy creates a new object which stores the reference of the original elements.

- So, a shallow copy doesn't create a copy of nested objects, instead it just copies the reference of nested objects. This means, a copy process does not recurse or create copies of nested objects itself.

Create a copy using shallow copy

```
import copy

old_list = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
new_list = copy.copy(old_list)

print("Old list:", old_list)
print("New list:", new_list)
```

Deep Copy

- A deep copy creates a new object and recursively adds the copies of nested objects present in the original elements.

- Let’s continue with example 2. However, we are going to create deep copy using deepcopy() function present in copy module. The deep copy creates independent copy of original object and all its nested objects.

Example

```
import copy

old_list = [[1, 1, 1], [2, 2, 2], [3, 3, 3]]
new_list = copy.deepcopy(old_list)

print("Old list:", old_list)
print("New list:", new_list)
```

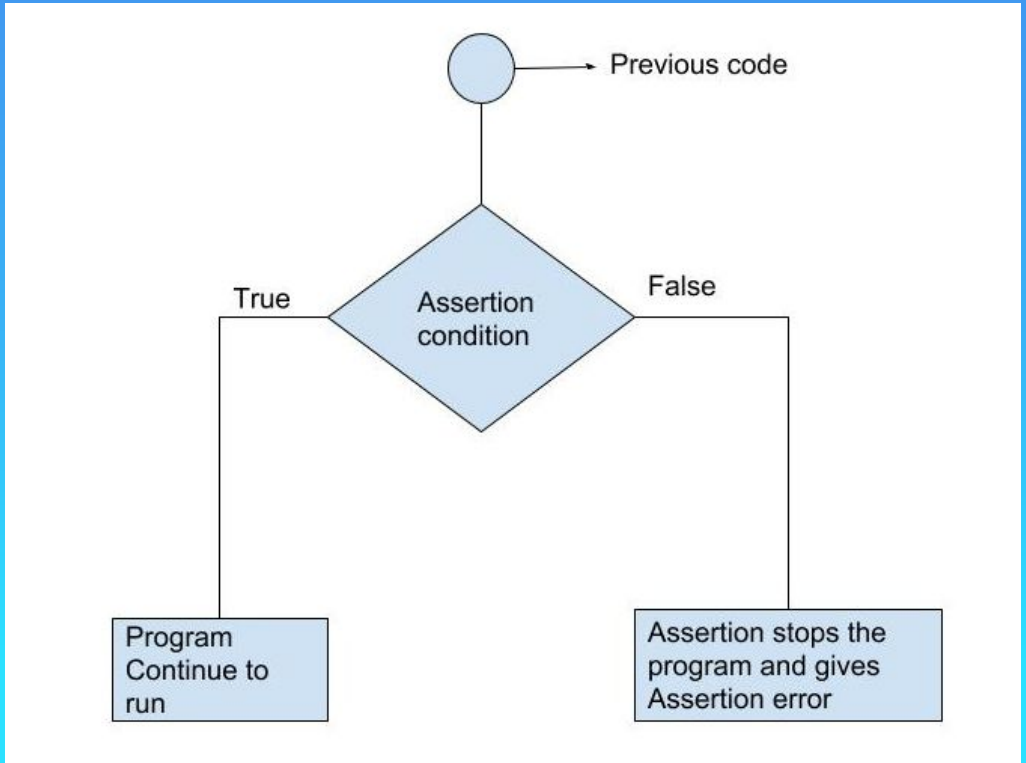
Assertion

Second Lesson

What is Assertion?

- Assertions are statements that assert or state a fact confidently in your program. For example, while writing a division function, you're confident the divisor shouldn't be zero, you assert divisor is not equal to zero.
- Assertions are simply boolean expressions that checks if the conditions return true or not. If it is true, the program does nothing and move to the next line of code. However, if it's false, the program stops and throws an error.
- It is also a debugging tool as it brings the program on halt as soon as any error is occurred and shows on which point of the program error has occurred.

Assert FlowChart



Python assert Statement

Python has built-in assert statement to use assertion condition in the program. assert statement has a condition or expression which is supposed to be always true. If the condition is false assert halts the program and gives an AssertionError.

- assert <condition>
- assert <condition>,<error message>

Example 1: Using assert without Error Message

```
def avg(marks):
    assert len(marks) != 0
    return sum(marks)/len(marks)

mark1 = []
print("Average of mark1:",avg(mark1))
```

Example 2: Using assert with error message

```
def avg(marks):
    assert len(marks) != 0,"List is empty."
    return sum(marks)/len(marks)

mark2 = [55,88,78,90,79]
print("Average of mark2:",avg(mark2))

mark1 = []
print("Average of mark1:",avg(mark1))
```

Key Points to Remember

- Assertions are the condition or boolean expression which are always supposed to be true in the code.
- assert statement takes an expression and optional message.
- assert statement is used to check types, values of argument and the output of the function.
- assert statement is used as debugging tool as it halts the program at the point where an error occurs.

Thank you

Miss You...