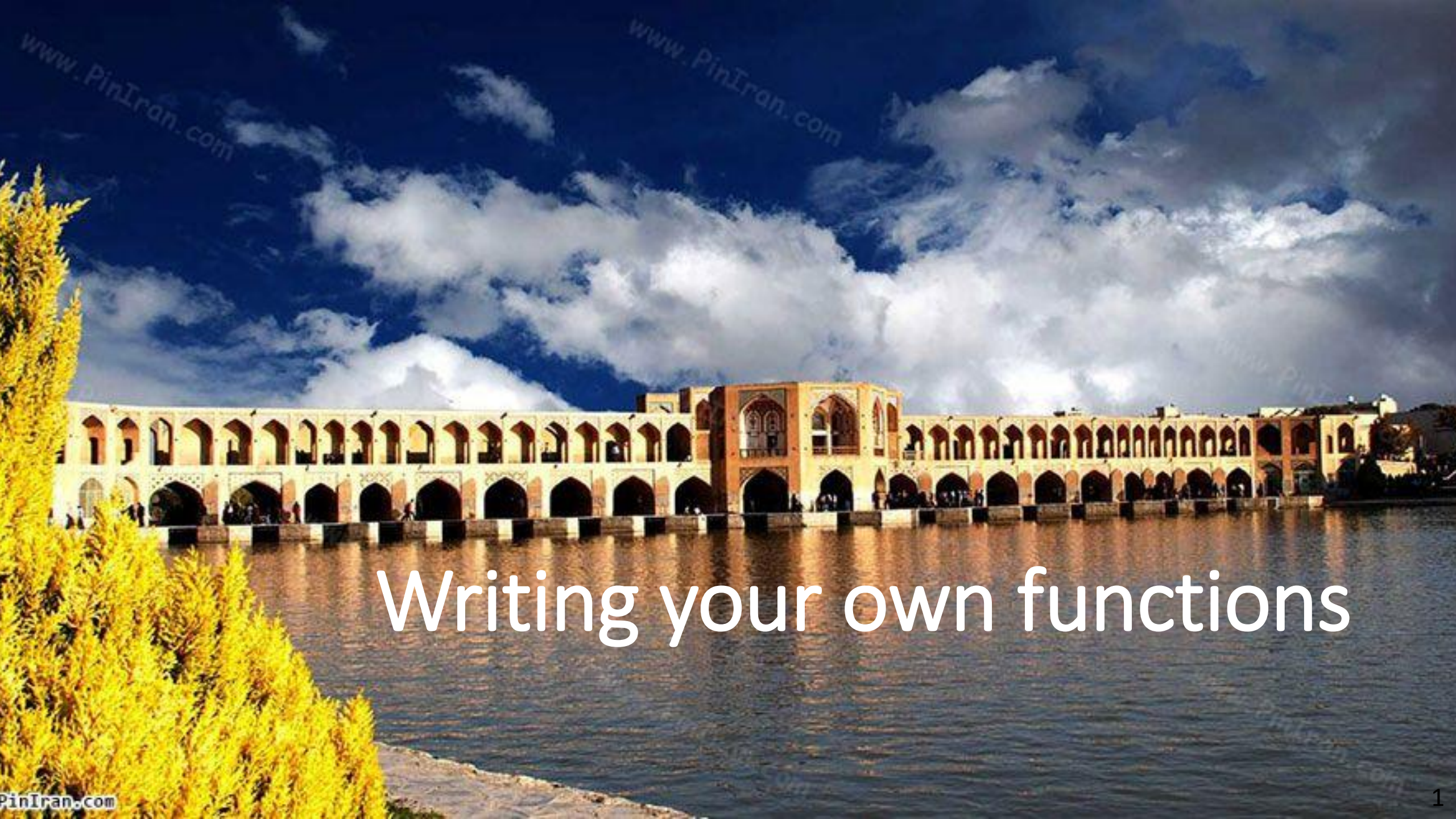


# Writing your own functions


User-defined
functions

# You'll learn:

- Define functions without parameters
- Define functions with one parameter
- Define functions that return a value
- Later: multiple arguments, multiple return values

# Built-in functions

- str()

```
x = str(5)


print(x)
```

```
'5'
```

```
print(type(x))
```

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

# Defining a function

```
def square():     # <- Function header
    new_value = 4 ** 2     # <- Function body
    print(new_value)
square()
```

```
16
```

# Function parameters

```
def square(value):
    new_value = value ** 2
    print(new_value)

square(4)
```

parameter

```
16
```

```
square(5)
```

Arguments

```
25
```

# Return values from functions

- Return a value from a function using return

```
def square(value):
    new_value = value ** 2
    return new_value
num = square(4)

print(num)
```

```
16
```

# Docstrings

- Docstrings describe what your function does
- Serve as documentation for your function
- Placed in the immediate line afer the function header
- In between triple double quotes """

```
def square(value):
    """Return the square of a value."""
    new_value = value ** 2
    return new_value
```


Multiple Parameters
and Return Values

# Multiple function parameters

- Accept more than 1 parameter:

```
def raise_to_power(value1, value2):
    """Raise value1 to the power of value2."""
    new_value = value1 ** value2
    return new_value
```

- Call function: # of arguments = # of parameters

```
result = raise_to_power(2, 3)

print(result)
```

```
8
```

# A quick jump into tuples

- Make functions return multiple values: Tuples!
- Tuples:
  - ✓ Like a list - can contain multiple values
  - ✓ Immutable - can’t modify values!
  - ✓ Constructed using parentheses ()

```
even_nums = (2, 4, 6)

print(type(even_nums))
```

```
<class 'tuple'>
```

# Unpacking tuples

- Unpack a tuple into several variables:

```
even_nums = (2, 4, 6)

a, b, c = even_nums
```

```
print(a)
```

```
2
```

```
print(b)
```

```
4
```

```
print(c)
```

```
6
```

# Accessing tuple elements

- Access tuple elements like you do with lists:

```
even_nums = (2, 4, 6)

print(even_nums[1])
```

```
4
```

```
second_num = even_nums[1]

print(second_num)
```

```
4
```

- Uses zero-indexing

# Returning multiple values

```
def raise_both(value1, value2):
    """Raise value1 to the power of value2
    and vice versa."""

    new_value1 = value1 ** value2
    new_value2 = value2 ** value1

    new_tuple = (new_value1, new_value2)

    return new_tuple
```

```
result = raise_both(2, 3)

print(result)
```

```
(8, 9)
```


Let's Continue!
PinIran.com