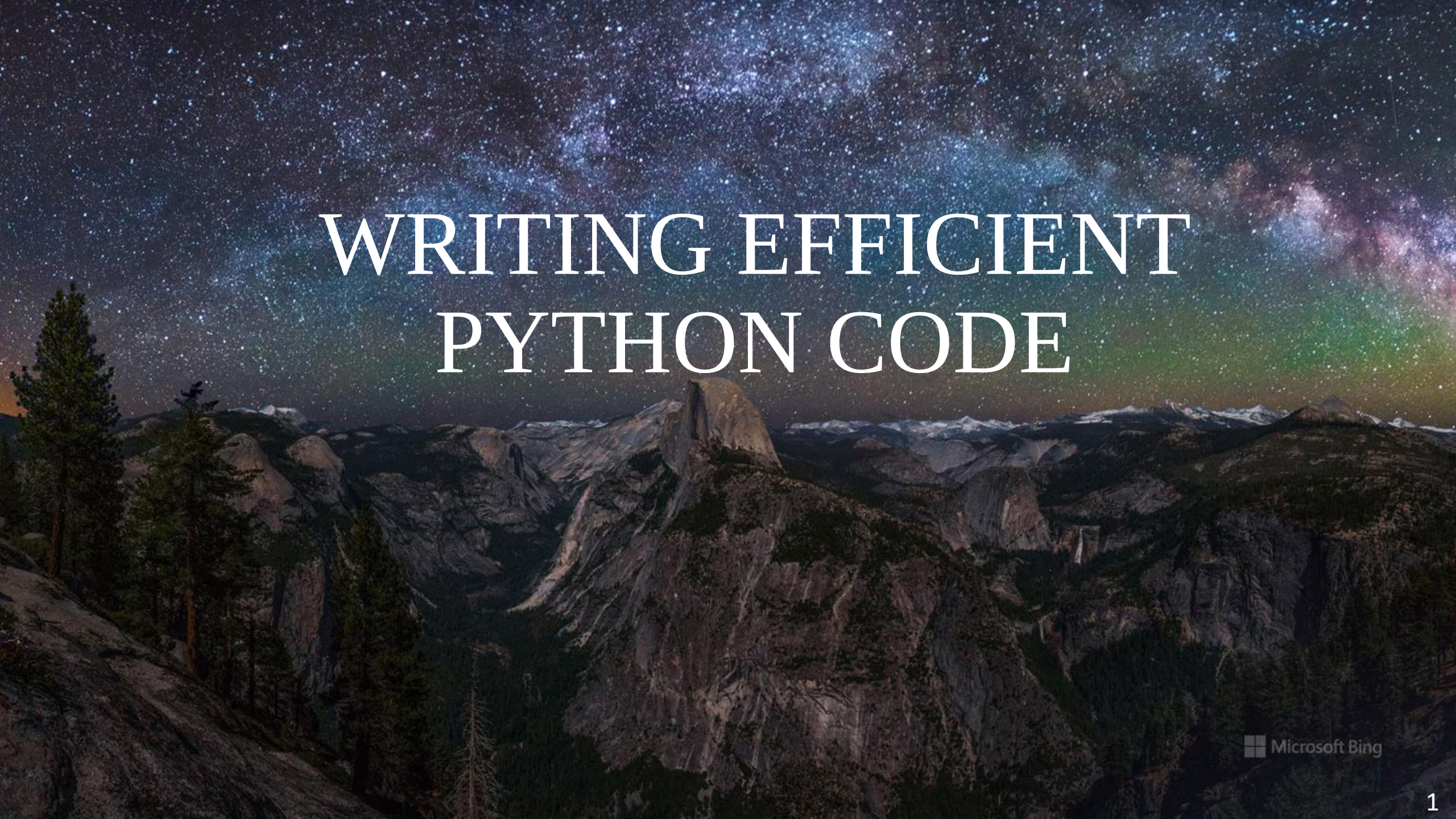


WRITING EFFICIENT PYTHON CODE

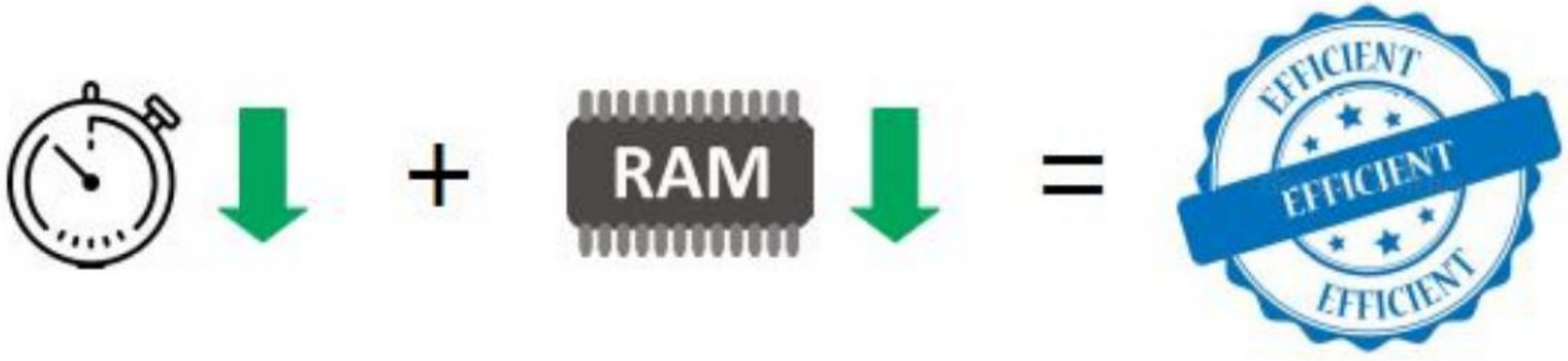
Course overview

- Your code should be a tool used to gain insights
  - ✓ Not something that leaves you waiting for results
- In this course, you will learn:
  - ✓ How to write clean, fast, and effcient Python code
  - ✓ How to profile your code for bottlenecks
  - ✓ How to eliminate bottlenecks and bad design paferns

Defining efficient

- Writing effcient Python code

  - ✓ Minimal completion time (fast runtime)
  - ✓ Minimal resource consumption (small memory footprint)



Defining Pythonic

- Writing effcient Python code
  - ✓ Focus on readability
  - ✓ Using Python's constructs as intended (i.e., Pythonic)

ut=[0, 1, 2, 3, 4]

```
# Non-Pythonic
doubled_numbers=[]

for i in range(5):
    doubled_numbers.append(i*2)

print(doubled_numbers)
```

```
# Pythonic
doubled_numbers = [x * 2 for x in range(5)]
print(doubled_numbers)
```

```
1 doubled_numbers
```

[0, 2, 4, 6, 8, 10]

Building with built-ins

The Python Standard Library

- **Python 3.6 Standard Library**
  - Part of every standard Python installation
- Built-in types
  - `list`, `tuple`, `set`, `dict`, and others
- Built-in functions
  - `print()`, `len()`, `range()`, `round()`, `enumerate()`, `map()`, `zip()`, and others
- Built-in modules
  - `os`, `sys`, `itertools`, `collections`, `math`, and others

Built-in function: range()

- Explicitly typing a list of numbers

```
nums = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

- Using range() to create the same list

```
# range(start,stop)
nums = range(0,11)

nums_list = list(nums)
print(nums_list)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

```
# range(stop)
nums = range(11)

nums_list = list(nums)
print(nums_list)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Built-in function: range()

- Using range() with a step value

start

end

step

```
even_nums = range(2, 11, 2)

even_nums_list = list(even_nums)
print(even_nums_list)
```

```
[2, 4, 6, 8, 10]
```

Built-in function: enumerate()

- Creates an indexed list of objects

```
letters = ['a', 'b', 'c', 'd' ]

indexed_letters = enumerate(letters)

indexed_letters_list = list(indexed_letters)
print(indexed_letters_list)
```

```
[(0, 'a'), (1, 'b'), (2, 'c'), (3, 'd')]
```

index item

Built-in function: enumerate()

- Can specify a start value

```
letters = ['a', 'b', 'c', 'd' ]

indexed_letters2 = enumerate(letters, start=5)

indexed_letters2_list = list(indexed_letters2)
print(indexed_letters2_list)
```

```
[(5, 'a'), (6, 'b'), (7, 'c'), (8, 'd')]
```

Built-in function: map()

- Applies a function over an object

```
nums = [1.5, 2.3, 3.4, 4.6, 5.0]

rnd_nums = map(round, nums)

print(list(rnd_nums))
```

```
[2, 2, 3, 5, 5]
```

Built-in function: map()

- map() with lambda (anonymous function)

```
nums = [1, 2, 3, 4, 5]

sqrd_nums = map(lambda x: x ** 2, nums)

print(list(sqrd_nums))
```

```
[1, 4, 9, 16, 25]
```

The power of NumPy arrays

NumPy array overview

- Alternative to Python lists

```
nums_list = list(range(5))
```

```
[0, 1, 2, 3, 4]
```

```
import numpy as np

nums_np = np.array(range(5))
```

```
array([0, 1, 2, 3, 4])
```

NumPy array homogeneity

```
# NumPy array homogeneity
nums_np_ints = np.array([1, 2, 3])
```

```
array([1, 2, 3])
```

```
nums_np_ints.dtype
```

```
dtype('int64')
```

```
nums_np_floats = np.array([1, 2.5, 3])
```

dot

```
array([1. , 2.5, 3. ])
```

```
nums_np_floats.dtype
```

```
dtype('float64')
```

NumPy array broadcasting

- Python lists don't support broadcasting

```
nums = [-2, -1, 0, 1, 2]
nums ** 2
```

```
TypeError: unsupported operand type(s) for ** or pow(): 'list' and 'int'
```

List approach

```python
nums = [-2, -1, 0, 1, 2]
```

```python
# For loop (inefficient option)
sqrd_nums = []
for num in nums:
    sqrd_nums.append(num ** 2)
print(sqrd_nums)
```

```
[4, 1, 0, 1, 4]
```

```python
# List comprehension (better option but not best)
sqrd_nums  = [num ** 2 for num in nums]

print(sqrd_nums)
```

```
[4, 1, 0, 1, 4]
```

NumPy array broadcasting

- NumPy array broadcasting for the win!

```
nums_np = np.array([-2, -1, 0, 1, 2])
nums_np ** 2
```

```
array([4, 1, 0, 1, 4])
```

NumPy Indexing

Basic 1-D indexing (lists)

```
nums = [-2, -1, 0, 1, 2]
nums[2]
```

```
0
```

```
nums[-1]
```

```
2
```

```
nums[1:4]
```

```
[-1, 0, 1]
```

Basic 1-D indexing (arrays)

```
nums_np = np.array(nums)
nums_np[2]
```

```
0
```

```
nums_np[-1]
```

```
2
```

```
nums_np[1:4]
```

```
array([-1, 0, 1])
```

NumPy Indexing

```
# 2-D list
nums2 = [ [1, 2, 3],
          [4, 5, 6] ]
```

```
# 2-D array

nums2_np = np.array(nums2)
```

- Basic 2-D indexing (lists)

```
nums2[0][1]
```

List index

Component of the list index

```
2
```

```
[row[0] for row in nums2]
```

```
[1, 4]
```

- Basic 2-D indexing (arrays)

```
nums2_np[0,1]
```

Row index

Column index

```
2
```

```
nums2_np[:,0]
```

```
array([1, 4])
```

NumPy array boolean indexing

```
nums = [-2, -1, 0, 1, 2]
nums_np =  np.array(nums)
```

- Boolean indexing

```
nums_np > 0
```

```
array([False, False, False,  True,  True])
```

```
nums_np[nums_np > 0]
```

```
array([1, 2])
```

No boolean indexing for lists

```python
nums = [-2, -1, 0, 1, 2]
# For loop (inefficient option)
pos = []
for num in nums:
    if num > 0:
        pos.append(num)
print(pos)
```

```
[1, 2]
```

```python
# List comprehension (better option but not best)
pos = [num for num in nums if num > 0]
print(pos)
```

```
[1, 2]
```


Let's Continue!
Microsoft Bing