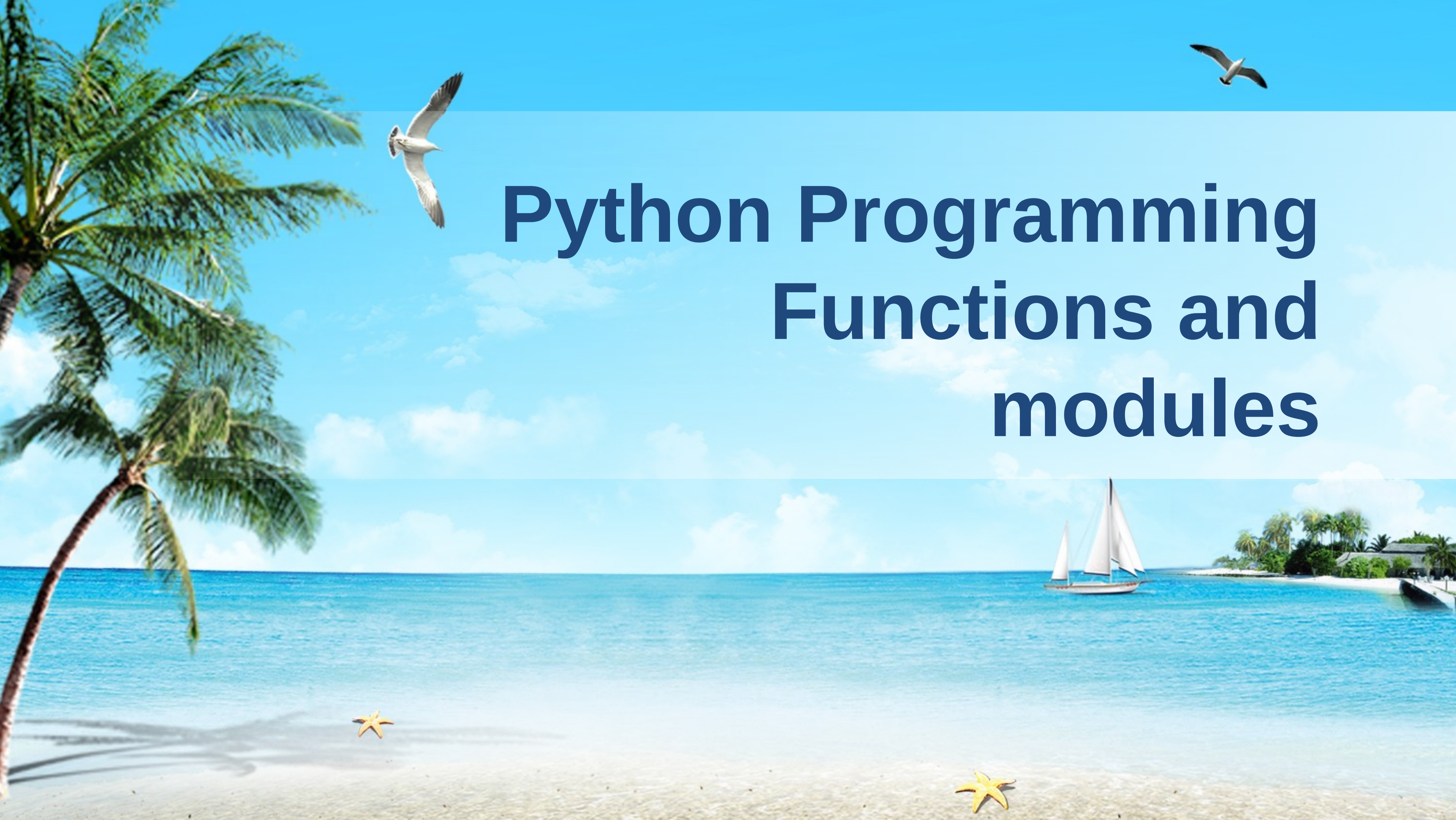

Python Programming Functions and modules

Functions and modules

- Defining and calling functions.
- Parameters and arguments.
- Default values for parameters.
- Variable scope.
- Return values.
- Creating and importing modules.

Defining functions

- syntax:

```
def function_name():
    # content of function is indented
    do_something()
# This is no longer part of the function
do_something_else()
```

- demo 7 : functions/Demos/hello_you.ps

Defining Functions

- demo 8 : functions/Demos/hello_you_expanded.ps
- output:

```
What is your name? Nat
===
Hello, Nat!
===
How about a Monty Python poem?
===
Much to his Mum and Dad's dismay
Horace ate himself one day.
He didn't stop to say his grace,
He just sat down and ate his face.
===
Goodbye!
```

- not all Python programs are modules
- modules that aren't programs wouldn't have a main() function

Local variables

- Demo 9: functions/Demos/local_var.py
- output:

```
NameError: name 'x' is not def
```

- A good IDE will let you know in the editor (like VS Code)

Global variables

- Demo 10: functions/Demos/global_var.py

```
from set_x(): 1

from get_x(): 0
```

- To Modify global variables within functions use global keyword:
- Demo 11: functions/Demos/global_var_in_function.py
- To prevent/deal with global variables:
  - use different naming e.g. underscores: _x
  - use parameters
- local / global variables have different scope

Function parameters

- recall:

```
def insert_separator():
    print("==="
```

- what if we want different separators?
- use more functions or:

```
def function_name(param1, param2, param3):
    do_something(param1, param2, param3)
```

- Demo 12: functions/Demos/hello_you_with_params.py
- Arguments vs Parameters?
  - Arguments are the actual values, while parameters are the variables

Using Parameter Names in Function Calls

- you can specify name when passing in arguments:

```
def divide(numerator, denominator):
    return numerator / denominator

divide(10, 2)
divide(numerator=10, denominator=2)
divide(denominator=2, numerator=10)
divide(10, denominator=2)
```

- then you can use arbitrary order
- you can also use combinations

Exercise 6

- A Function with Parameters
  - 15-25 min

Default values

- assign default values:

```
def function_name(param=default):
    do_something(param)
```

- or:

```
def insert_separator(s="="):
    print(s, s, s, sep="")
```

- Demo: /Demos/hello_you_with_defaults.py

Exercise 7

- Parameters with Default Values
  - 15–25 min

Returning values

- functions can return values
- Demo 13: functions/Demos/add_nums_with_return.py

Import Modules

- what if you want to make your functions available to other modules (code re-use)?
- modules can import other modules with the **import** keyword
- Demo 14: functions/Demos/import_example.py
- import with or without prefix
- then use keyword **from**

```
from module_name import function1, function
```

- Demo 15: functions/Demos/import_example2.py

Import with aliases

- you can also import with alias

```
import add_nums_with_return as anwr

total = anwr.add_nums(1, 2, 3, 4, 5)
```

- to prevent naming conflicts

```
from foo import do_this
from bar import do_this as do_tha
```

main() function

- a module can check to see if it is being imported by checking the **__name__** variable

```
if __name__ == '__main__':
    main()
```

- they only run their main() function if __name__ equals __main__
- so if a module is being imported it doesn't run their own main()
- demo

Module search path

- locating imported modules:
  - current directory
  - library of standard modules
  - the paths in sys.path

```
import sys

print(sys.path)
```

pyc files

- Files with a .pyc extension are compiled Python files. They are automatically created in a __pycache__ folder the first time a file is imported
- loading code from a . pyc file is faster than parsing and translating a . py file, so the presence of precompiled . pyc files improves the start-up time of Python scripts.
- "pyc" files are not compatible across Python major releases.
- They will automatically be created/updated each time you import a module that is new or has been changed.

Methods vs. Functions

- built-in functions:
  - print()
  - input()
  - len()
- your own functions:
  - insert_separator()
  - divide()
- methods:
  - 'Hello World'.upper()
- methods are part of an Object