Pangasinan State University
Region's Premier University of Choice

LESSON 5:

INTERMEDIATE ELEMENTS OF PYTHON 2

Modules

A module is a file that contains Python code, such as functions, classes, and variables. Modules serve to organize code and promote code reusability by breaking down large programs into smaller, more manageable components. They allow you to logically group related code together and separate concerns in your projects.

Types of Modules

1. Built-in modules

- ✓ math
- ✓ datetime
- ✓ random
- ✓ time
- ✓ collections
- ✓ itertools

math module

Provides mathematical functions such as square roots, trigonometric functions, logarithms, and constants like pi and e.

Example:

```
import math
print(math.sqrt(16))  # Output: 4.0
print(math.pi)        # Output: 3.141592653589793
```

Task # 1

Calculate the area of a circle, asking for an input for its radius.

Formula: 2 x pi x r

datetime module

The datetime module allows for date and time manipulation, formatting, and arithmetic.

Example:

```
from datetime import datetime, timedelta
now = datetime.now()
print(now)
print(now + timedelta(days=5))
```

datetime module

Format Code	Meaning	Example
%Y	Year with century (4 digits)	2024
%y	Year without century (2 digits)	24
%m	Month as zero-padded decimal	01 to 12
%B	Full month name	November
%b	Abbreviated month name	Nov
%d	Day of the month (zero-padded)	01 to 31
%A	Full weekday name	Tuesday
%a	Abbreviated weekday name	Tue
%w	Weekday as decimal (0=Sunday, 6=Saturday)	0 to 6
%j	Day of the year (zero-padded)	001 to 366
%U	Week number of the year (Sunday as first day of the week, zero-padded)	00 to 53
%W	Week number of the year (Monday as first day of the week, zero-padded)	00 to 53

datetime module

Format Code	Meaning	Example
%H	Hour (24-hour clock, zero-padded)	00 to 23
%I	Hour (12-hour clock, zero-padded)	01 to 12
%p	AM or PM	AM, PM
%M	Minute (zero-padded)	00 to 59
%S	Second (zero-padded)	00 to 59
%f	Microsecond (zero-padded to 6 digits)	000000 to 999999

Task # 2

1. Ask a date inputs from the user with the following format (yyyy-mm-dd). Print the date with the following format:
Example:

```
Enter date: 2024-11-12
November 12, 2024
```

2. Get the current datetime.
Expected Output:

```
Tue 2024-11-12 10:43AM
```

random module

The random module generates pseudo-random numbers and selections from sequences.

Example:

```
import random
print(random.randint(1, 10))
choices = ['rock', 'paper', 'scissors']
print(random.choice(choices))
```

Task # 3

Let’s create a Random Word Guessing Game.

time module

The time module deals with time-related functions, including delays and measuring execution time.

Example:

```
import time
print(time.time())  # Current timestamp
time.sleep(2)       # Pauses for 2 seconds
```

Task # 4

Measure the time it takes to execute a specific function.

collections module

The collections module includes specialized data types like Counter, deque, defaultdict, etc.

Example:

```
from collections import Counter
data = ['apple', 'banana', 'apple']
print(Counter(data))
```

Task # 5

- ✓ Create a function to ask for user inputs and then add it to an empty list terminated by `exit`.
- ✓ Create a function to count the occurrences of each word using Counter.

itertools module

The itertools module provides tools for creating iterators for efficient looping.

Example:

```
from itertools import permutations
data = [1, 2, 3]
print(list(permutations(data, 2)))
```

Task # 6

- ✓ Create a function that ask for 3 elements and store it in a list.
- ✓ Generate All Possible Permutations and Combinations of the elements.

Types of Modules

2. User-defined modules - User-defined modules in Python are custom modules created by developers to organize their code into separate files and promote code reusability

Sets

Sets are a built-in data type used to store collections of unique elements. A set is an unordered collection of elements, meaning that the elements have no specific order, and they are defined using curly braces {} or the set() constructor.

Creating a set

```
mySet = {1, 2, 3, 4, 5}
myStringSet = {"apple", "banana", "orange"}
empty_set = set()
```

Note: You cannot create an empty set using {} because it will create an empty dictionary

Looping a set

You can use a for loop to iterate over the elements in the set. Since sets are unordered, the order of iteration is not guaranteed.

Adding items to a set

1. `add()`

- allows you to add one element at a time to the set. If the element you are trying to add is already in the set, the add() method will have no effect, as sets do not allow duplicate elements.

Adding items to a set

`2. union() or |`

- Allows you to add tuples to tuples, so if you want to add one item, (or many), create a new tuple with the item(s), and add it to the existing tuple.

Adding items to a set

`3. update()`

`-` It is used to modify a set by adding elements from another iterable.

Changing set items

You cannot change individual items in a set directly. Sets are mutable, but they do not support item assignment because they are unordered collections of unique elements. If you want to change the elements in a set, you must remove the existing elements and then add new ones.

Removing set items

1. remove()
This method removes a specific element from the set. If the element is not present in the set, it will raise a KeyError.

Removing set items

2. discard()

This method removes a specific element from the set if it exists. If the element is not present, it does nothing and does not raise an error.

Removing set items

3. pop()

This method removes and returns an arbitrary element from the set. As sets are unordered, the choice of which element is removed is arbitrary (for strings).

Removing set items

4. clear()

Remove all elements.

Other methods: difference()

The difference() method returns a set that contains the difference between two sets. The returned set contains items that exist only in the first set, and not in both sets. As a shortcut, you can use the - operator instead.

Syntax:

```
set.difference(set1, set2 ... etc.)
```

Other methods: intersection ()

The intersection() method returns a set that contains the similarity between two or more sets. The returned set contains only items that exist in both sets, or in all sets if the comparison is done with more than two sets. As a shortcut, you can use the & operator instead

Syntax:

```
set.intersection(set1, set2 ... etc.)
```

Other methods: symmetric_difference ()

The symmetric_difference() method returns a set that contains all items from both set, but not the items that are present in both sets. The returned set contains a mix of items that are not present in both sets. As a shortcut, you can use the ^ operator instead.

Syntax:

```
set.intersection(set1, set2 ... etc.)
```