Programming Concepts
L-05

By Vladyslav **BARDIN**

01: Agenda

- Data types & Conditional Statements
- Loops
- Collections
- Packages in Python
- Wordle

02: What Are Data Types?

➢ **Definition:**

Data types define the kind of information a variable can hold. Determines what operations you can perform on data

➢ **Python's Core Data Types:**

- ✓ Numeric: `int`, `float`
- ✓ Text: `str`
- ✓ Boolean: `bool`

```
age = 30  # 'age' is an integer variable
price = 19.99  # 'price' is a floating-point variable
```

03: Numeric Types: `int`

➢ Integers

- ✓ Whole numbers (no decimal points).

➢ Common Operations:

- ✓ Arithmetic: +, -, *, /, // (floor division), % (modulo)
- ✓ Comparisons: ==, !=, >, <, >=, <=

```
age = 30
count = -5

result = age // 2  # Floor division (result: 15)
remainder = count % 3  # Modulo (result: 1)
```

04: Numeric Types: `float`

➢ Floating-Point Numbers

- ✓ Numbers with decimal points.

➢ Common Operations:

- ✓ Arithmetic: +, -, *, /, // (floor division), % (modulo)
- ✓ Comparisons: ==, !=, >, <, >=, <=

```
pi = 3.14159
temperature = 98.6

# Floating-Point Operations
rounded_pi = round(pi, 2)  # Rounds pi to 3.14
distance = abs(-10.5)      # Absolute value (result: 10.5)
```

05 Boolean Types: `str`

➢ Boolean

- ✓ Represents logical values – True or False.

➢ Common Operations:

- ✓ Logical Operators:
  - → and: True if both operands are True.
  - → or: True if at least one operand is True.
  - → not: Reverses the truth value (True -> False, False ->True).
- ✓ Comparison Operators:
  - → Used to compare values and produce Boolean results.

```
count = stripped_text.count("is")
print(count)                    # 2

# Checking prefixes and suffixes
print(stripped_text.startswith("This"))  # True
print(stripped_text.endswith("words."))  # True
```

06: String Types: `str`

➢ String

- ✓ Represents sequences of characters (text).

➢ Common Operations:

`lower()`
Converts all characters to lowercase.
`upper()`
Converts all characters to uppercase.
`capitalize()`
Capitalizes the first letter of the string.
`title()`
Capitalizes the first letter of each word.
`strip()`
Removes leading and trailing whitespace (spaces, tabs, newlines).

```
text = "   This is a sample TEXT string with SOME  Repeated words.   "

# Case manipulation
print(text.lower())          # "   this is a sample text string with some  repeated words.
  "
print(text.upper())          # "   THIS IS A SAMPLE TEXT STRING WITH SOME  REPEATED WORDS.
  "
print(text.capitalize())     # "   This is a sample text string with some  repeated
words.   "
print(text.title())          # "   This Is A Sample Text String With Some  Repeated
Words.   "

# Whitespace and replacement
stripped_text = text.strip()
print(stripped_text)         # "This is a sample TEXT string with SOME  Repeated words."
```

07: String Types: `str`

➢ Common Operations:

`replace(old, new, [count])`
Replaces occurrences of old with new. Optionally, count limits the number of replacements.

`split(sep=None, maxsplit=-1)`
Splits the string into a list of substrings based on the separator sep. maxsplit limits the number of splits.

`join(iterable)`
Joins elements of an iterable (like a list) into a string, using the string itself as the separator.

`find(sub, start=0, end=len(string))`
Returns the lowest index where the substring sub is found. Returns -1 if not found.

```python
replaced_text = stripped_text.replace(" ", "_").replace("SOME", "some")
print(replaced_text)       # "This_is_a_sample_TEXT_string_with_some__Repeated_words."

# Splitting and joining
words = stripped_text.split()
print(words)                # ['This', 'is', 'a', 'sample', 'TEXT', 'string', 'with',
'SOME', 'Repeated', 'words.']
joined_text = "-".join(words)
print(joined_text)          # "This-is-a-sample-TEXT-string-with-SOME-Repeated-words."

# Searching and counting
index = stripped_text.find("sample")
print(index)
```

08: Conditional Operators: The if Statement

➢ Conditional Operators

- ✓ Control the flow of your program based on conditions.

➢ Conditions:

- ✓ Expressions that evaluate to either True or False (Boolean values).
- ✓ Comparison Operators:
  - → == (equal to),
  - → != (not equal to),
  - → > (greater than),
  - → < (less than),
  - → >= (greater than or equal to),
  - → <= (less than or equal to)
- ✓ Logical Operators:
  - → and: True if both operands are True.
  - → or: True if at least one operand is True.
  - → not: Reverses the truth value (True -> False, False ->True).

09: Loops

- **Iteration:**
  The process of going through each item in a collection, one at a time.
- **Why Loops Matter:**
  Loops allow you to automate repetitive tasks, perform calculations on each element, and dynamically interact with your data.
- **Types of Loops in Python:**
  - ✓ for loops: Ideal for iterating over sequences (lists, tuples, strings, etc.)
  - ✓ while loops: Used when the number of iterations is unknown beforehand.

10: Loops

➢ Use **enumerate()** when you need both the index and the value of each item.

```
fruits = ["apple", "banana", "cherry"]
for index, fruit in enumerate(fruits):
    print(f"{index}: {fruit}")
    # Output: 0: apple
    #         1: banana
    #         2: cherry
```

11: Lists

- **Lists:**
  Ordered, mutable sequences – our workhorse for general-purpose collections.
  Your grocery shopping list, where order matters and you can add/remove items.
- **Ordered:**
  Items have a specific position, like numbers in a line.
- **Mutable:**
  You can change, add, or remove items after the list is created.
- **Sequence:**
  A type of collection where items can be accessed by their index (position).

12: Lists

Method	Description	Example
append()	Adds an item to the end of the list	fruits.append("grape")
insert()	Inserts an item at a specific index	fruits.insert(2, "mango")
pop()	Removes and returns the last item	last_fruit = fruits.pop()
remove()	Removes the first occurrence of a value	fruits.remove("orange")
Slicing	Extracts a portion of the list	first_two = fruits[:2] all_but_first = fruits[1:]

13: Tuples

- **Tuples:**
  Ordered, immutable sequences – ideal for fixed data that shouldn't change.
  A person's name and birthdate, which should stay fixed.
- **Ordered:**
  Like lists, tuples maintain the order of their items.
- **Immutable:**
  Once created, you cannot change the values within a tuple.
- **Sequence:**
  Also accessed by index, just like lists.

14: Tuples

Method	Description	Example
count()	Returns the number of times a value appears	my_tuple.count(5)
index()	Returns the index of the first occurrence of a value	my_tuple.index('a')

15: Sets

- **Sets:**
  Unordered, unique collections – perfect for eliminating duplicates and performing set operations.
  A collection of unique ingredients in a recipe.
- **Unordered:**
  Items have no specific position; you cannot access them by index.
- **Unique:**
  Each value can appear only once in a set.
- **Mutable:**
  You can add or remove items, but not modify existing ones.

16: Sets

Method	Description	Example
add()	Adds an element to the set	unique_numbers.add(8)
remove()	Removes a specific element (raises an error if not found)	unique_numbers.remove(3)
discard()	Removes a specific element (no error if not found)	unique_numbers.discard(10)

17: Sets

Method	Description	Example
union()	Combines elements from two or more sets	set1.union(set2)
intersection()	Returns a new set with elements common to both sets	set1.intersection(set2)
difference()	Returns a new set with elements in the first set but not the second	set1.difference(set2)

18: Dictionary

- **Dictionary:**
  Unordered, mutable collections of key-value pairs – our go-to for fast lookups and associations.
- **Unordered:**
  Items do not have a specific position..
- **Mutable:**
  You can change, add, or remove key-value pairs after the dictionary is created.
- **Key-Value Pairs:**
  Each key is unique and maps to a value, enabling efficient data retrieval based on keys.

19: Dictionary

Method	Description	Example
dict.clear()	Removes all items from the dictionary.	student_grades.clear()
dict.copy()	Returns a shallow copy of the dictionary.	grades_copy = student_grades.copy()
dict.fromkeys()	Creates a new dictionary with keys from a sequence and values set to a specified value.	new_dict = dict.fromkeys(['Alice', 'Bob'], 0)
dict.get()	Returns the value for a specified key if the key is in the dictionary.	student_grades.get('Alice', 'No record')
dict.items()	Returns a view object that displays a list of dictionary's key-value tuple pairs.	for item in student_grades.items(): print(item)
dict.keys()	Returns a view object that displays a list of all the keys.	keys = student_grades.keys()
dict.pop()	Removes the specified key and returns the corresponding value.	grades = student_grades.pop('Alice')
dict.popitem()	Removes and returns the last inserted key-value pair.	last_item = student_grades.popitem()
dict.setdefault()	Returns the value of a key if it is in the dictionary. If not, inserts the key with a specified value.	grade = student_grades.setdefault('Eve', [80, 85, 90])
dict.update()	Updates the dictionary with elements from another dictionary object or from an iterable of key-value pairs.	student_grades.update({'Eve': [80, 85, 90]})
dict.values()	Returns a view object that displays a list of all the values.	values = student_grades.values()

20: List Comprehensions

```python
# Traditional for loop
squares = []

for num in range(1, 6):
    squares.append(num ** 2)

# List comprehension equivalent
squares = [num ** 2 for num in range(1, 6)]
```

21: Additional Resources

- Python Loops Tutorial: for Loops, while Loops and Nested Loops by Programming with Mosh: https://www.codingem.com/nested-loops-in-python/
- Python Tutorial for Beginners 5: Lists, Tuples, and Sets by Corey Schafer: https://www.youtube.com/watch?v=W8KRzm-HUcc
- Lists and Tuples in Python on Real Python: https://realpython.com/python-lists-tuples/
- Python Sets on W3Schools: https://www.w3schools.com/python/python_sets.asp
- https://www.w3schools.com/python/python_dictionaries.asp
- https://www.geeksforgeeks.org/python-dictionary/