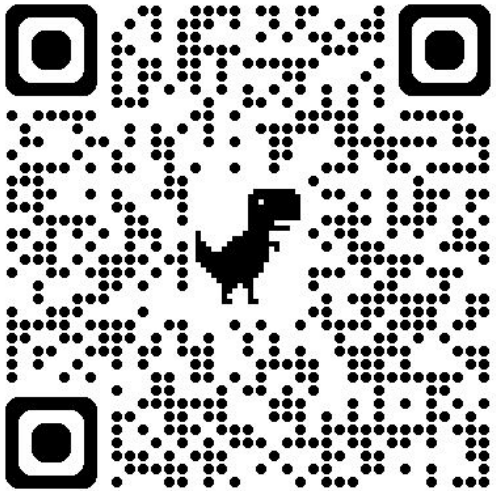
Programming Concepts
L-02

By Vladyslav **BARDIN**

00: Examples

- **Today's Examples:**
  https://tinyurl.com/l02-examples



01: What Are Variables?

- **Definition:**
  A variable is a named container for storing information (like a labeled box).
- **Purpose:**
  Allows us to reference and manipulate data easily.

```
name = "Alice"  # 'name' is the variable, "Alice" is the value (string)
```

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: 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."
```

06: 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.

```
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)
```

07: 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
```

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: Conditional Operators: The if Statement

```
age = 17
if age ⩾ 18:
    print("You are eligible to vote.")  # Not executed
else:
    print("You are not yet eligible to vote.")  # Executed

name = "Bob"
if name == "Alice":
    print("Welcome, Alice!")          # Not executed

is_raining = True
is_sunny = False
temperature = 95
if is_raining and not is_sunny:
    print("Take an umbrella.")       # Executed

if temperature > 90 or is_sunny:
    print("It's a hot day!")          # Executed
```

10: Additional Resources

- Lecture Examples: https://tinyurl.com/l02-examples