CP020001 Computer Programming

Lecture: Basic Python Part I

https://github.com/kaopanboonyuen/CP020001_ComputerProgramming_2024s2

Contact: teerapong.pa@chula.ac.th

Outlines

- Data Types
- Input
- Loop and Condition
- Start, stop, and step arguments in Python's range()
- Break, continue, and else with loops
- Assignment

Data Types

🌟 What Are Data Types?

Data types specify the kind of data a variable holds, helping Python determine how to manipulate that data.

🔢 1. Numeric Data Types

Used for numbers like integers and decimals.

a. Integer (`int`)

- Whole numbers (positive or negative, without decimals).

Example:

python Copy code

```python
x = 42
print(type(x))  # Output: <class 'int'>
```

b. Float (`float`)

- Numbers with decimal points.

Example:

python Copy code

```python
y = 3.14
print(type(y))  # Output: <class 'float'>
```

c. Complex (`complex`)

- Numbers with real and imaginary parts.

Example:

python Copy code

```python
z = 3 + 5j
print(type(z))  # Output: <class 'complex'>
```

✏️ 2. String (`str`)

A sequence of characters enclosed in single, double, or triple quotes.

Example:

python

Copy code

```python
name = "Python"
print(type(name))  # Output: <class 'str'>
```

Common Operations on Strings:

python

Copy code

```python
greeting = "Hello"
name = "World"
print(greeting + " " + name)  # Concatenation: Output: "Hello World"
print(len(greeting))          # Length: Output: 5
print(greeting[0])            # Indexing: Output: "H"
print(greeting[-1])           # Negative Indexing: Output: "o"
print(greeting.upper())       # Uppercase: Output: "HELLO"
```

✅ 3. Boolean (`bool`)

Represents `True` or `False` values (used in conditions).

Example:

python

Copy code

```python
is_python_fun = True
print(type(is_python_fun))  # Output: <class 'bool'>
print(5 > 3)                # Output: True
print(3 > 5)                # Output: False
```

🛍️ 4. List (`list`)

A collection of items that are **ordered**, **mutable**, and **allow duplicates**.

Example:

python

Copy code

```python
fruits = ["🍎", "🍌", "🍇"]
print(type(fruits))          # Output: <class 'list'>
print(fruits[0])             # Access first item: Output: "🍎"
fruits.append("🍓")          # Add an item
print(fruits)                # Output: ['🍎', '🍌', '🍇', '🍓']
```

📋 5. Tuple (`tuple`)

A collection of items that are **ordered**, **immutable**, and **allow duplicates**.

Example:

python

Copy code

```python
coordinates = (10, 20, 30)
print(type(coordinates))      # Output: <class 'tuple'>
print(coordinates[1])         # Access second item: Output: 20
```

🗂️ 6. Dictionary (`dict`)

A collection of key-value pairs that are **unordered**, **mutable**, and **do not allow duplicate keys**.

Example:

python

Copy code

```python
person = {"name": "Alice", "age": 25}
print(type(person))           # Output: <class 'dict'>
print(person["name"])         # Access value by key: Output: "Alice"
person["age"] = 26            # Modify value
print(person)                 # Output: {'name': 'Alice', 'age': 26}
```

🎛️ 7. Set (`set`)

A collection of unique items that are **unordered** and **unindexed**.

Example:

python Copy code

```python
unique_numbers = {1, 2, 3, 3}
print(type(unique_numbers))   # Output: <class 'set'>
print(unique_numbers)         # Output: {1, 2, 3} (duplicates removed)
unique_numbers.add(4)         # Add item
print(unique_numbers)         # Output: {1, 2, 3, 4}
```

🚫 8. NoneType (`None`)

Represents the absence of a value.

Example:

python

Copy code

```python
x = None
print(type(x))                # Output: <class 'NoneType'>
```

⚡ Quick Summary of Data Types:

Type	Description	Example
`int`	Whole numbers	`42`
`float`	Numbers with decimals	`3.14`
`complex`	Numbers with real and imaginary parts	`3 + 5j`
`str`	Text or characters	`"Hello"`
`bool`	Boolean values (`True` or `False`)	`5 > 3`
`list`	Ordered, mutable collection	`["🍎", "🍌", "🍇"]`
`tuple`	Ordered, immutable collection	`(10, 20, 30)`
`dict`	Key-value pairs	`{"name": "Alice", "age": 25}`
`set`	Unordered collection of unique items	`{1, 2, 3}`
`NoneType`	Represents no value	`None`

Input()

🌟 What Is `input()`?

- `input()` is a function used to take input from the keyboard.
- By default, it takes input as a **string**.
- If you need an **integer** or **float**, you must convert the input using `int()` or `float()`.

🖋️ 1. Input as a String

No conversion is needed since `input()` already takes input as a string.

Example:

python

Copy code

```python
name = input("What's your name? ")
print(f"Hello, {name}!")  # Output: Hello, [name entered]
```

🧱 2. Input as an Integer

Convert the input to an integer using `int()`.

Example:

python

Copy code

```python
age = int(input("How old are you? "))
print(f"Next year, you'll be {age + 1} years old!")  # Adds 1 to the entered age
```

🔢 3. Input as a Float

Convert the input to a float using `float()`.

Example:

python

Copy code

```python
height = float(input("Enter your height in meters: "))
print(f"Your height is {height} meters.")  # Displays the entered height as a float
```

🚀 Putting It All Together

Here's a combined example to take different types of inputs:

python

Copy code

```python
# Input as a string
name = input("Enter your name: ")

# Input as an integer
age = int(input("Enter your age: "))

# Input as a float
weight = float(input("Enter your weight in kilograms: "))

# Output all inputs
print(f"Hello, {name}! You are {age} years old and weigh {weight} kg.")
```

📝 Key Notes:

1. **Always validate input**:
    - Use `try-except` for error handling if the user enters invalid data.
2. **Type Conversion**:
    - Use `int()` for integers.
    - Use `float()` for decimals.
    - Keep input as `str` for text or alphanumeric input.

⚡ Challenge for Students:

Write a program that:

1. Takes the user's name (string).
2. Asks for their birth year (integer).
3. Calculates and displays their age in 2024 (integer).
4. Asks for their height in meters (float) and calculates BMI given their weight.

🧱 How the Program Works:

1. **Inputs:**
   - User provides their name (`str`), birth year (`int`), height (`float`), and weight (`float`).
2. **Calculations:**
   - `age = current_year - birth_year` calculates the user's age.
   - `bmi = weight / (height ** 2)` calculates the Body Mass Index (BMI).
3. **Outputs:**
   - The program prints all the collected and calculated data in a neat format.

🌟 Sample Run

Input:

yaml Copy code

```
Enter your name: Alice
Enter your birth year: 2000
Enter your height in meters: 1.65
Enter your weight in kilograms: 55
```

Output:

csharp Copy code

```
--- Results ---
Hello, Alice!
You are 24 years old.
Your height is 1.65 meters, and your weight is 55 kg.
Your BMI is 20.20.
```

Loop and Condition

(This Week) - Loop and Condition

Structure	Purpose	When to Use
For Loop	Repeats for a fixed number of times	When you know how many times to iterate
While Loop	Repeats as long as a condition is true	When the number of iterations depends on a condition
If-Else	Executes code based on a condition	When you need decision-making
Elif	Handles multiple conditions	When more than two outcomes are possible

🔁 For Loop

A `for` loop is used when you know **how many times** you want to iterate or when you want to iterate over items in a sequence (like a list, string, or range).

Syntax:

python Copy code

```python
for variable in sequence:
    # Code to execute repeatedly
```

Example:

python

Copy code

```python
fruits = ["🍎", "🍌", "🍇"]
for fruit in fruits:
    print(f"I love {fruit}")
# Output:
# I love 🍎
# I love 🍌
# I love 🍇
```

- **Explanation**:
  - The variable `fruit` takes each value in the `fruits` list, one at a time.
  - The loop stops after the last item is processed.

🔁 While Loop

A `while` loop is used when you don't know the number of iterations in advance and want to repeat **as long as a condition is true**.

Syntax:

python

Copy code

```
while condition:
    # Code to execute repeatedly
```

Example:

python

Copy code

```python
count = 0
while count < 5:
    print("🍄" * (count + 1))  # Print mushrooms
    count += 1  # Increment the counter
# Output:
# 🍄
# 🍄🍄
# 🍄🍄🍄
# 🍄🍄🍄🍄
# 🍄🍄🍄🍄🍄
```

- **Explanation**:
  - The condition `count < 5` is checked **before** each iteration.
  - The loop stops when the condition becomes false.

🌟 If-Else Conditional

The `if-else` structure allows decision-making in the program based on a **condition**.

Syntax:

python Copy code

```python
if condition:
    # Code to execute if the condition is true
else:
    # Code to execute if the condition is false
```

Example:

python

Copy code

```python
number = 10
if number % 2 == 0:
    print(f"{number} is even 🎉")
else:
    print(f"{number} is odd 🔢")
# Output: 10 is even 🎉
```

🌟 Elif (Else-If)

The `elif` statement allows **multiple conditions** to be checked one after another.

Syntax:

python

Copy code

```python
if condition1:
    # Code to execute if condition1 is true
elif condition2:
    # Code to execute if condition2 is true
else:
    # Code to execute if none of the above is true
```

Example:

python

Copy code

```python
age = 18
if age < 13:
    print("You are a child 👶")
elif age < 20:
    print("You are a teenager 🧕")
else:
    print("You are an adult 👨‍💼")
# Output: You are a teenager 🧕
```

- **Explanation**:
  - The conditions are checked **in order**.
  - Once a true condition is found, the rest are skipped.

💡 Additional Concepts:

1. Range in `for` Loops

The `range()` function generates numbers, which are often used in loops.

Syntax: `range(start, stop, step)`

Example:

python

Copy code

```python
for i in range(1, 10, 2):  # Start at 1, go up to 9, step by 2
    print(i, end=" ")
# Output: 1 3 5 7 9
```

2. Combining `if` with Loops

You can use `if` conditions **inside loops** to filter or control logic.

Example:

python

Copy code

```python
for i in range(1, 10):
    if i % 2 == 0:
        print(f"{i} is even 🎯")
    else:
        print(f"{i} is odd 💡")
# Output:
# 1 is odd 💡
# 2 is even 🎯
# 3 is odd 💡
# ...
```

3. Infinite Loop with `While`

An infinite loop happens when the condition in `while` **never becomes false**.

Example:

python

Copy code

```python
while True:  # This will run forever!
    answer = input("Type 'exit' to stop: ")
    if answer == "exit":
        print("Goodbye! 👋")
        break  # Exit the loop
```

start, stop, and step arguments in Python range()

Explanation of `range(start, stop, step)`

- `start` **(optional)**: The starting value of the sequence (default is `0`).
- `stop` **(required)**: The sequence ends **before** this value.
- `step` **(optional)**: The difference between consecutive values (default is `1`). Can be negative for reverse counting.

1. Basic Counting with Default Step

python

Copy code

```python
# Example 1: Default start (0) and step (1)
for i in range(5):  # start=0, stop=5, step=1
    print(i, end=" ")  # Output: 0 1 2 3 4
```

2. Specify `start` and `stop`

python

Copy code

```python
# Example 2: Specifying start and stop
for i in range(2, 6):  # start=2, stop=6, step=1
    print(i, end=" ")  # Output: 2 3 4 5
```

3. Counting Backwards with Negative Step

python

Copy code

```python
# Example 3: Negative step for reverse counting
for i in range(5, 0, -1):  # start=5, stop=0, step=-1
    print(i, end=" ")  # Output: 5 4 3 2 1
```

4. Skipping Numbers with Positive Step

python

Copy code

```python
# Example 4: Step > 1
for i in range(0, 10, 2):  # start=0, stop=10, step=2
    print(i, end=" ")  # Output: 0 2 4 6 8
```

5. Counting Odd Numbers

python

Copy code

```python
# Example 5: Generating odd numbers
for i in range(1, 10, 2):  # start=1, stop=10, step=2
    print(i, end=" ")  # Output: 1 3 5 7 9
```

6. Counting in Reverse by Skipping

python

Copy code

```python
# Example 6: Reverse counting with skipping
for i in range(10, 0, -2):  # start=10, stop=0, step=-2
    print(i, end=" ")  # Output: 10 8 6 4 2
```

7. Custom Range with Negative Start

python

Copy code

```python
# Example 7: Counting with a negative start
for i in range(-5, 6):  # start=-5, stop=6, step=1
    print(i, end=" ")  # Output: -5 -4 -3 -2 -1 0 1 2 3 4 5
```

8. Counting by Fractions (Approximation)

python

Copy code

```python
# Example 8: Simulating fractions using integers
for i in range(0, 10, 3):  # step=3
    print(i / 10, end=" ")  # Output: 0.0 0.3 0.6 0.9
```

9. Using `range` with a List

python

Copy code

```python
# Example 9: Access elements by index
names = ["Alice", "Bob", "Charlie"]
for i in range(len(names)):  # range(0, 3)
    print(f"Index {i}: {names[i]}")
# Output:
# Index 0: Alice
# Index 1: Bob
# Index 2: Charlie
```

10. Nested Loops with Ranges

python

Copy code

```python
# Example 10: Creating a multiplication table
for i in range(1, 4):  # Outer loop (1 to 3)
    for j in range(1, 4):  # Inner loop (1 to 3)
        print(f"{i} x {j} = {i * j}", end=" | ")
    print()  # New line after each row
# Output:
# 1 x 1 = 1 | 1 x 2 = 2 | 1 x 3 = 3 |
# 2 x 1 = 2 | 2 x 2 = 4 | 2 x 3 = 6 |
# 3 x 1 = 3 | 3 x 2 = 6 | 3 x 3 = 9 |
```

Explanation of `range(start, stop, step)`

- `start` **(optional)**: The starting value of the sequence (default is `0`).
- `stop` **(required)**: The sequence ends **before** this value.
- `step` **(optional)**: The difference between consecutive values (default is `1`). Can be negative for reverse counting.

break, continue, and else with loops

Explanation of Concepts

1. `break` : Stops the loop immediately and exits.
2. `continue` : Skips the current iteration and moves to the next.
3. `else` : Executes when the loop completes without a `break` .

1. Using `break` to Exit a Loop Early

python

Copy code

```python
# Example 1: Break when finding a target
for i in range(1, 10):  # Loop through numbers 1 to 9
    print(f"Checking {i}...")
    if i == 5:  # Break the loop when i equals 5
        print("Found 5! Breaking the loop.")
        break
# Output:
# Checking 1...
# Checking 2...
# Checking 3...
# Checking 4...
# Checking 5...
# Found 5! Breaking the loop.
```

2. Using `continue` to Skip an Iteration

python

Copy code

```python
# Example 2: Skip even numbers
for i in range(1, 10):  # Loop through numbers 1 to 9
    if i % 2 == 0:  # If the number is even, skip it
        continue
    print(i, end=" ")  # Print odd numbers
# Output: 1 3 5 7 9
```

3. Combining `break` and `else` in Loops

python

Copy code

```python
# Example 3: Check if a number is prime
n = int(input("Enter a number: "))  # Input a number
for i in range(2, n):  # Loop from 2 to n-1
    if n % i == 0:  # If divisible by i, not a prime number
        print(f"{n} is not a prime number. Found a divisor: {i}")
        break
else:  # This runs if the loop finishes without breaking
    print(f"{n} is a prime number!")
# Try inputs like 7 (prime) or 10 (not prime)
```

4. Skipping with `continue` in a Nested Loop

python

Copy code

```python
# Example 4: Print a grid, but skip certain cells
for row in range(1, 4):  # Outer loop for rows
    for col in range(1, 4):  # Inner loop for columns
        if row == col:  # Skip cells where row equals col
            continue
        print(f"Cell ({row}, {col})")
# Output:
# Cell (1, 2)
# Cell (1, 3)
# Cell (2, 1)
# Cell (2, 3)
# Cell (3, 1)
# Cell (3, 2)
```

5. Breaking Out of a Nested Loop

python

Copy code

```python
# Example 5: Break out of a nested loop
for i in range(1, 4):  # Outer loop
    for j in range(1, 4):  # Inner loop
        print(f"Checking ({i}, {j})...")
        if i == 2 and j == 2:  # Break both loops when (2, 2)
            print("Breaking out of both loops!")
            break
    if i == 2:  # Ensure the outer loop also stops
        break
# Output:
# Checking (1, 1)...
# Checking (1, 2)...
# Checking (1, 3)...
# Checking (2, 1)...
# Checking (2, 2)...
# Breaking out of both loops!
```

6. Using `break` to Simulate a Simple Game

python

Copy code

```python
# Example 6: Guess the number game
secret = 7  # Secret number
print("Guess the secret number (1-10):")
while True:  # Infinite loop
    guess = int(input("Your guess: "))
    if guess == secret:  # If guessed correctly, break
        print("🎉 Correct! You win!")
        break
    elif guess < secret:  # Hint if guess is too low
        print("Too low! Try again.")
    else:  # Hint if guess is too high
        print("Too high! Try again.")
```

7. `continue` in a Fun Pattern

python

Copy code

```python
# Example 7: Skip printing 🍄 on certain rows
for i in range(1, 6):  # Rows 1 to 5
    if i == 3:  # Skip row 3
        continue
    print("🍄" * i)  # Print mushrooms
# Output:
# 🍄
# 🍄🍄
# 🍄🍄🍄🍄
# 🍄🍄🍄🍄🍄
```

8. Using `else` with a Loop for a Treasure Hunt

python

Copy code

```python
# Example 8: Treasure hunt
treasure_map = ["💧", "💧", "💧", "💰", "💧"]
for i, spot in enumerate(treasure_map):  # Enumerate the list
    if spot == "💰":  # If treasure is found
        print(f"Treasure found at position {i}!")
        break
else:  # If the loop completes without finding treasure
    print("No treasure found!")
# Output: Treasure found at position 3!
```

Explanation of Concepts

1. `break` : Stops the loop immediately and exits.
2. `continue` : Skips the current iteration and moves to the next.
3. `else` : Executes when the loop completes without a `break` .

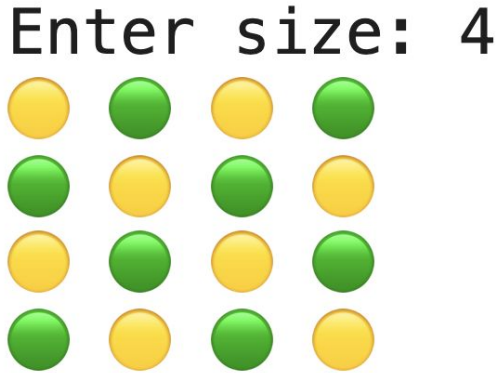
3. Generate an Upside-Down Checkerboard 🟢🟡

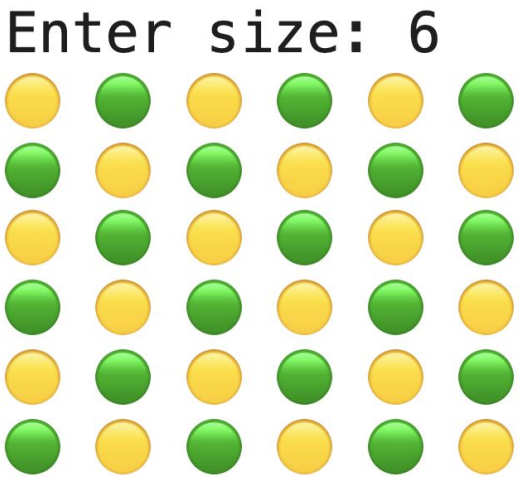
Write a program to print a **checkerboard pattern** of size `n x n`, but with a twist: it should start from the bottom row and work upward.

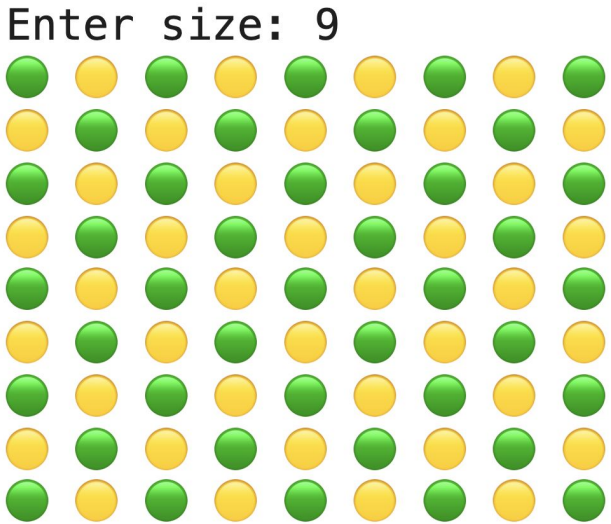
Example Input/Output:

Input: `Enter size: 4`

Output:







Assignment for this week

Enter size of the Christmas tree: 9

(10 Points) Christmas Tree Assignment 🎄⭐

Objective:

The goal of this assignment is to:

1. Practice using **loops** to repeat actions.
2. Use **conditionals** to create a pattern for the Christmas tree.
3. Work with **user input** to customize the size of the tree.
4. Have fun by designing a cool Christmas tree in Python using emojis.
