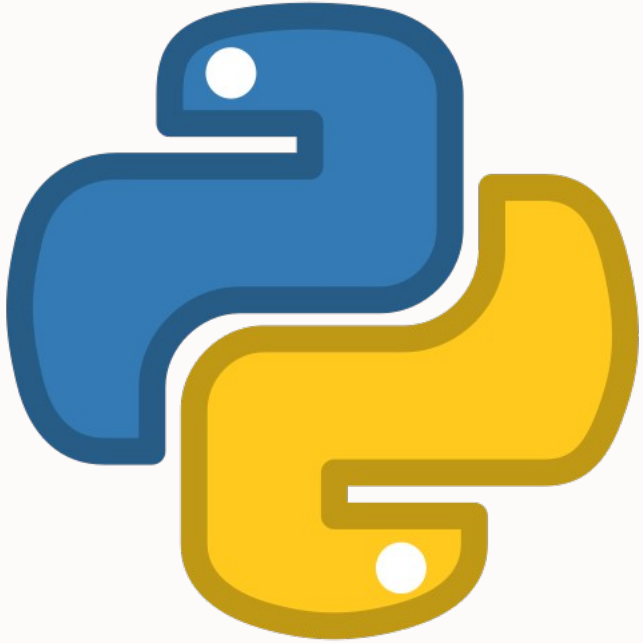
Welcome to the

Python Camp

Day 02



Log into the virtual machine

Username: student
Password: StudentDLH2024

Content of the course

- **Variables and data types**
- **Logical and arithmetic operations**
- Conditionals
- Loops
- Lists
- Dictionaries
- Sorting

Recap

Python syntax rules

Python syntax rules

- Case-sensitive
- Indentation
- Line continuation (/)
- Comments # or “”” or ‘’’

Variables

Variables

Rules ?

Variables

- Start with letter or _underscore
- Cannot start with a number
- Unique word in the code
- Only Alpha-numeric chars (a-z or A-Z or 0-9)
- No Python keywords

Data Type

Text type: **str**
Numeric types: **int, float, complex**
Sequence types: **list, tuple, range**
Mapping types: **dict**
Set types: **set, frozenset**
Boolean type: **bool**
Binary types: **bytes, bytesarray, memoryview**
None type: **NoneType**

Data Type

```
my_string = "Hello
world !”
my_int = 5
my_float = 20.5
```

Function

An independent block of code that can be called from anywhere

```
function_name(argument1, argument2, …)
```

Name of the algorithm.

Data on how to execute the algorithm now. Commas separate the function parameters.

Casting

Converting from a data type to another

```
number =
int(n)
```

Number

Other data type

Operations

Numbers

Addition	Subtraction	Multiplication	Division	Exponentiation
+	-	*	/	**

The integer part of the division	The remainder of the division
//	%

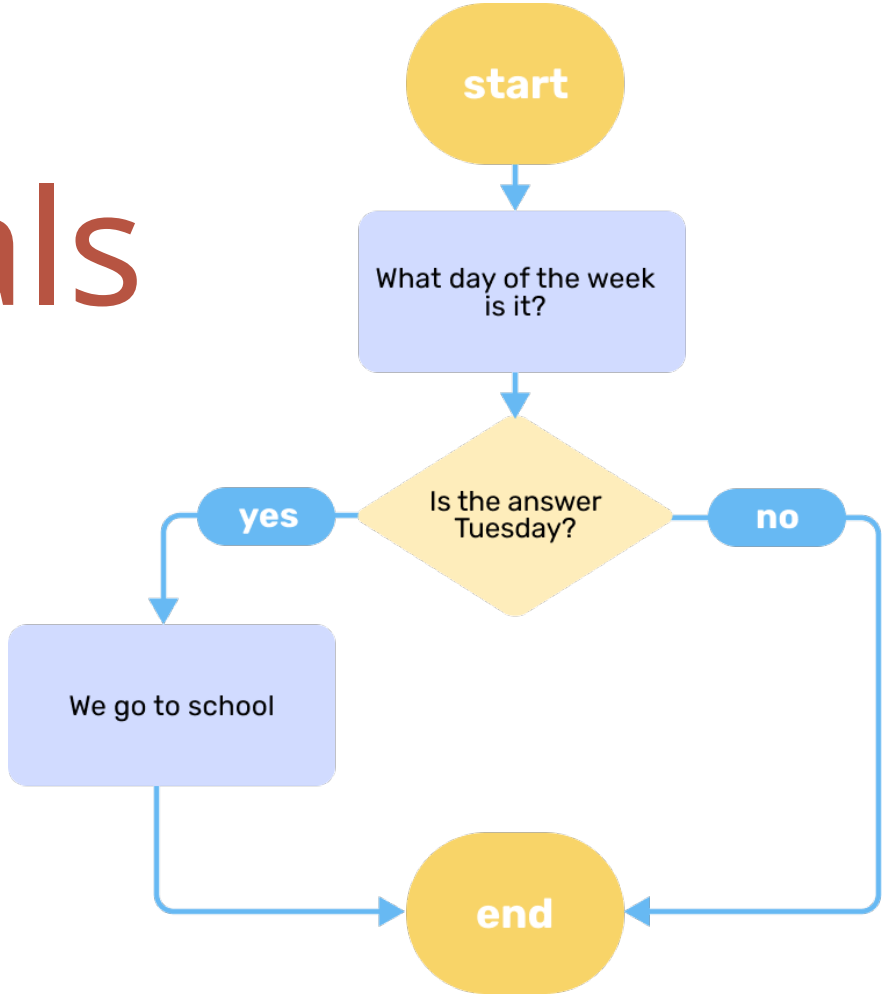
String

Concatenation (combining)	Multiplication (duplication)
+	*

Conditionals

Conditionals

```
day = input("What day of the week is it")
if day == "Tuesday":
    print("We go to school")
```



Conditionals

```
a = int(input("Enter a number: "))
if a > 0:
    print("Positive number")
elif a < 0:
    print("Negative number")
else:
    print("Number equal to zero")
```

Conditionals

The condition to check.

Conditional statement.

```
Command 1
if <condition>:
    Command 2
    Command 3
Command 4
```

Comparison operators

<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
==	Equality
!=	Inequality

Now YOUR TURN !

Let’s do exercises number 1

Operators priorities

Operator	Description
(**)	Exponentiation
(*)(/)(%)(//)	Multiplication, division, remainder of division, integer part of division
(+)(-)	Addition, subtraction
(<)(<=)(>)(>=)	Comparison operators
(==)(!=)	Equality operators
(=)	Assignment operator
(not)	Logical operator "NOT"
(and)	Logical operator "AND"
(or)	Logical operator "OR"

Example

```
if (x_1 == 5 or y_1 == 10) and (x_2 == "green" or y_2 ==
"red"):
```

Example

1 3 2 7 4 6 5

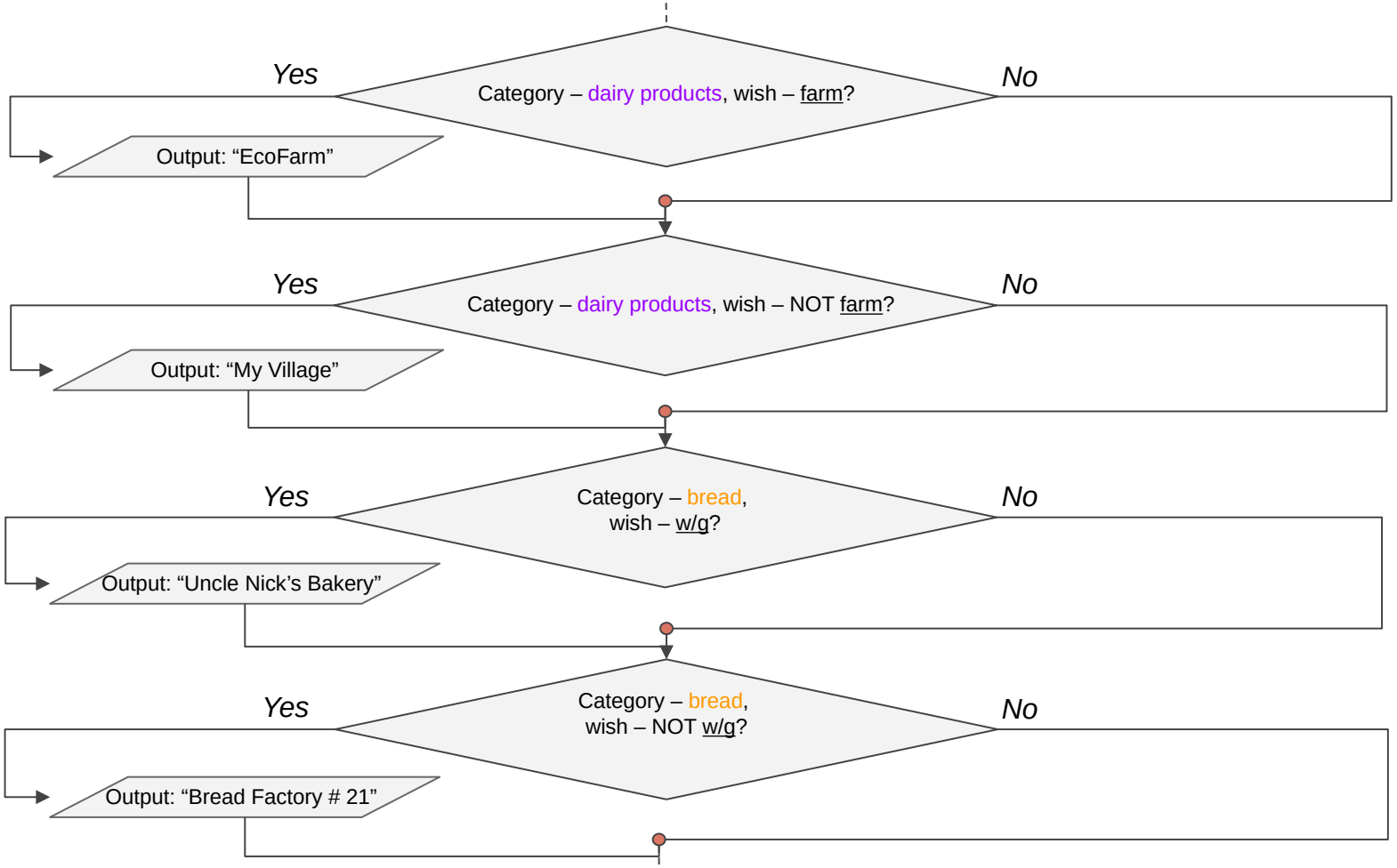
```
if (x_1 == 5 or y_1 == 10) and (x_2 == "green" or y_2 ==
"red"):
```

Now YOUR TURN !

Let’s do exercises number 2

Nested Conditionals

The program flowchart might look like this:



Nested Conditionals

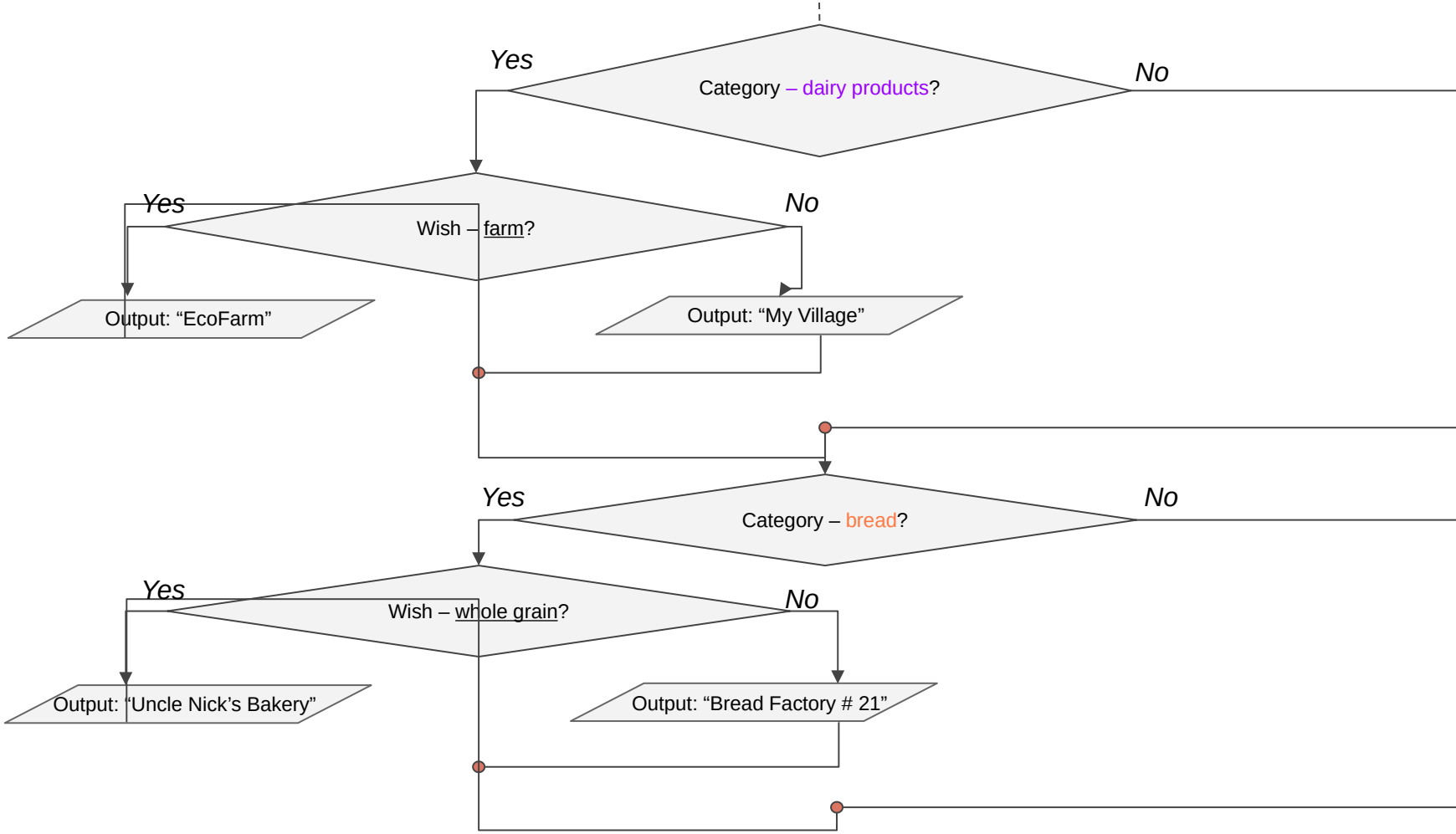
```
category = input('Product category:')
wish = input('Wish:')
if category == 'dairy products' and wish == 'farm':
    print('EcoFarm')
if category == 'dairy products' and wish != 'farm':
    print('My Village')
if category == 'bread' and wish == 'whole grain':
  print('Uncle Nick’s Bakery')
if category == 'bread' and wish != 'whole grain':
  print('Bread Factory # 21')
```

The category check is repeated.

Can we **do the check once**, not twice?

If so, how will the flowchart change?

The program flowchart might look as follows:



Nested Conditionals

```
category = input('Product category:')
wish = input('Wish:')
if category == 'dairy products':
   if wish == 'farm':
       print('EcoFarm')
   else:
       print('My Village')
if category == 'bread':
   if wish == 'whole grain':
       print('Uncle Nick’s Bakery')
   else:
       print('Bread Factory # 21')
```

Nesting design

There are no new design rules. You must adhere very carefully to the rules you already know.

```
if Expression_1 is true :
    if Expression_2 is true :
        Execute action 1
    else:
        Execute action 2
else:
    Execute action 3
```

Now YOUR TURN !

Let’s do exercises number 3

For loop

Example for for_loop

```
for i in 1,2,3:
    print(i, end=" ")

print()

for i in range(5):
    print(i, end=" ")
print()
for i in range(1,5):
    print(i, end=" ")
print()
for i in range(1,10,2):
    print(i, end=" ")

print()
```

for Loop with Python range()

In Python, the range() function returns a sequence of numbers. For example,

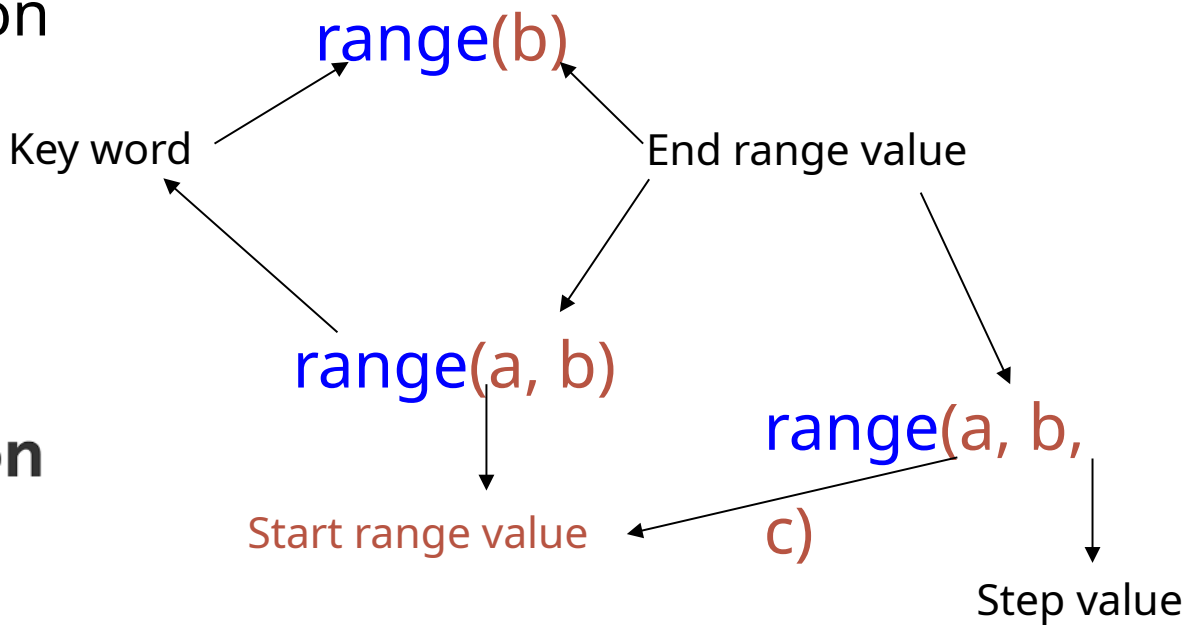
values = range(Here, range(4) returns a sequence of **0**, **1**, **2** ,and **3**.

Since the range() function returns a sequence of numbers, we can iterate over it using a for loop. For example,

```
# iterate from i = 0 to i
= 3
for i in range(4):
    print(i)
```

For Loops

Range function



Tutor Python

For Loops

Multiple values in a for loop must be specified using a sequential data structure (for example, a list or string).

```
for i in 1, 2, 3:
  print(i)
```

The variable i alternately takes the values 1, 2, 3.

```
for i in "one", "two",
"three":
  print(i)
```

The variable i alternately takes the values “one”, “two”, “three”.

For Loops

```
# Example 1: Using range(stop)
# Generates numbers from 0 up to (but not including)
the stop value
print("Example 1:")
for num in range(5):
    print(num) # Output: 0, 1, 2, 3, 4
```

For Loops

```
# Example 2: Using range(start, stop)
# Generates numbers from start up to (but not
including) the stop value
print("Example 2:")
for num in range(2, 6):
    print(num) # Output: 2, 3, 4, 5
```

For Loops

```
# Example 3: Using range(start, stop, step)
# Generates numbers from start up to (but not
including) the stop value with the specified step
print("Example 3:")
for num in range(1, 10, 2):
    print(num) # Output: 1, 3, 5, 7, 9
```

thon for Loop

Python, we use a for loop to iterate over sequences such as lists, strings, dictionaries, etc.

```
languages = ['Swift', 'Python', 'Go']

# access elements of the list one by one
for lang in languages:
    print(lang)
```

```
Swift
Python
Go
```

for loop Syntax

```
r val in sequence: # body of the loop
```

The for loop iterates over the elements of sequence in order. In each iteration, the body of the loop is executed.
The loop ends after the last item in the sequence is reached.

While Loops

```
while <repetition condition>:
    Command 1
    Command 2
    ......................
    Command n

continuation of the program
```

Loop

Indent

Command block

Python while Loop

In Python, we use a while loop to repeat a block of code until a certain condition is met. For example,

```
number = 1

while number <= 3:
    print(number)
    number = number
+ 1
```

```
1
2
3
```

Now YOUR TURN !

Let’s do exercises number 4