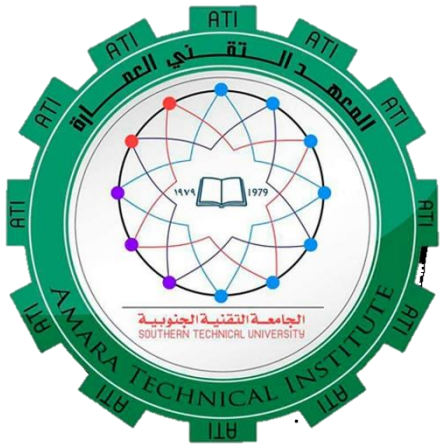


وزارة التعليم العالي والبحث العلمي
الجامعة التقنية الجنوبية
المعهد التقني / العمارة
قسم تقنيات الشبكات وبرامجيات الحاسوب
المرحلة الاولى

الحقيبة التدريسية لمادة

PYTHON LANGUAGE

M.S.c. Haider . J . Swadi
Assistant Teacher
Master of Information Technology

دليل البرنامج

Python is a high-level, versatile, open-source, interpreted programming language that supports multiple programming paradigms. These features have made it one of the most popular and sought-after languages of the modern era.

Python is a truly general-purpose language, meaning it is not designed for a single problem but can be adapted to solve a very wide range of programming challenges. This versatility, coupled with its simplicity and active community, makes it an indispensable tool in any programmer's arsenal today.

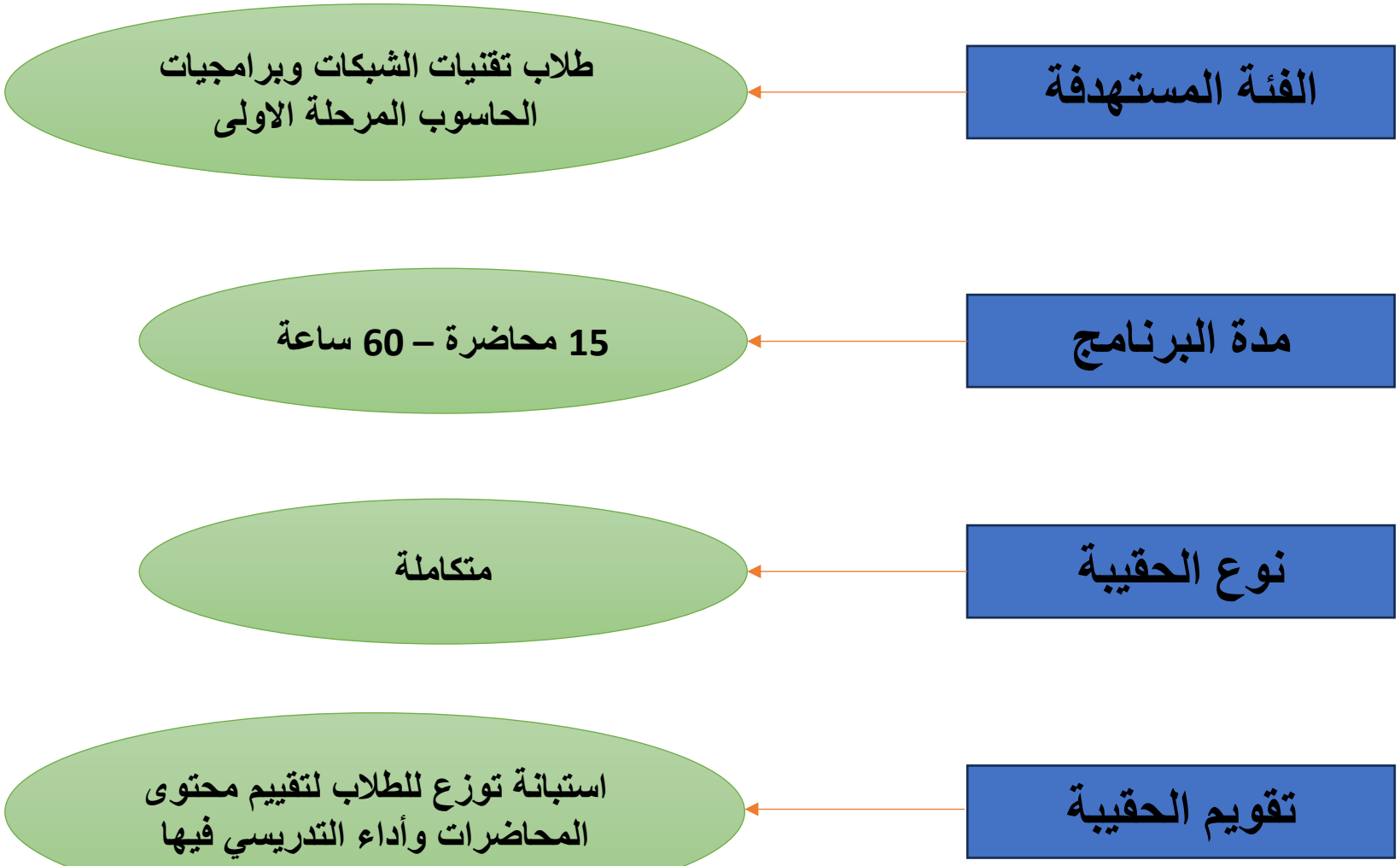
محتويات الحقيبة التدريسية

دليل البرنامج

اسم البرنامج ← PYTHON

اهداف البرنامج ← **الهدف العام** ←

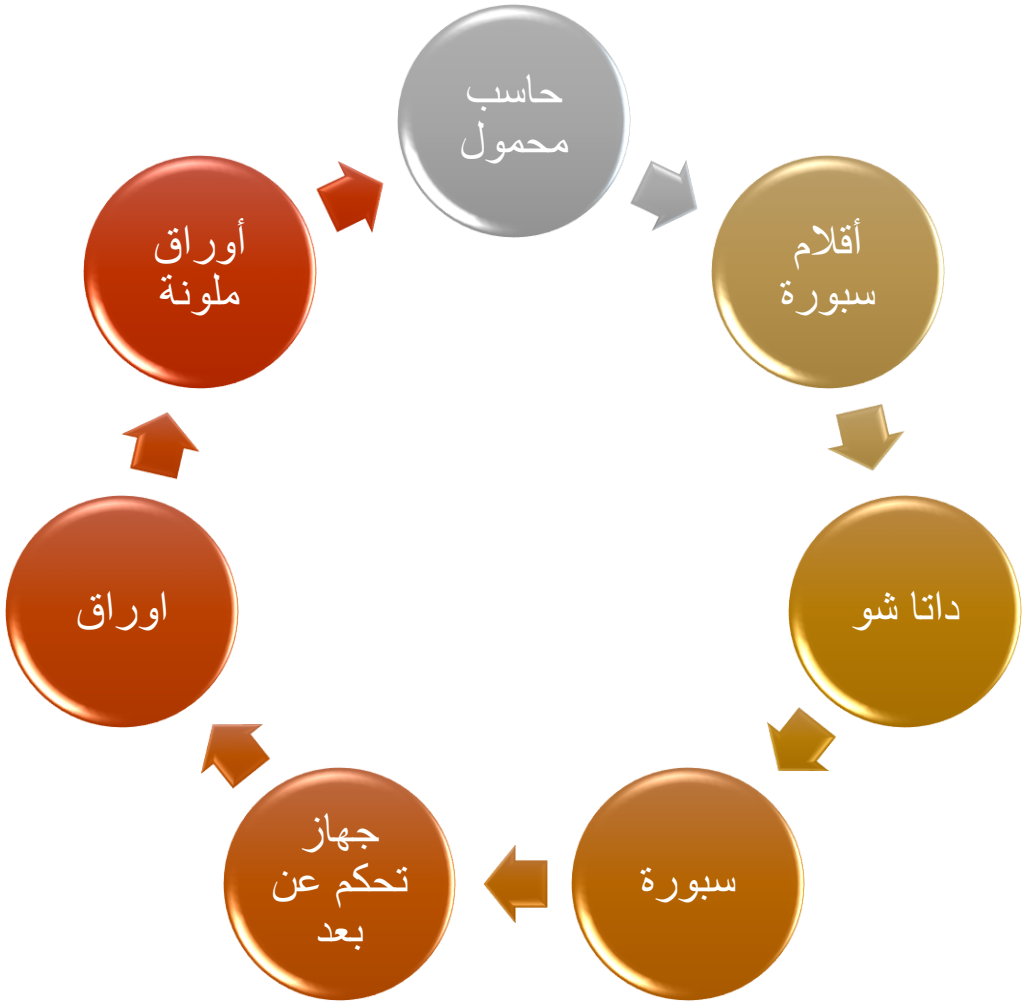
1.فهم أساسيات الإدخال والإخراج، وهياكل التحكم، والوظائف، والتسلسلات، والقوائم.
2. تصميم منطق البرامج ثم تنفيذها باستخدام بايثون.
3. فهم مفاهيم البرمجة ومهارات حل المشكلات، دون افتراض أي خبرة برمجية سابقة.


الفئة المستهدفة
طلاب تقنيات الشبكات وبرامجيات الحاسوب المرحلة الاولى
مدة البرنامج
15 محاضرة – 60 ساعة
نوع الحقيبة
متكاملة
تقويم الحقيبة
استبانة توزع للطلاب لتقييم محتوى المحاضرات وأداء التدريسي فيها

الأدوات والوسائل

برامج التواصل

المنصة التعليمية



الأنشطة والأساليب التعليمية المستخدمة

إرشادات للطلاب

- التحضير المسبق للمحاضرات.
- المشاركة الفعالة في المحاضرات.
- تدوين الملاحظات بشكل منظم.
- المراجعة المنتظمة للمواد.
- حل البرامج والمسائل.
- العمل الجماعي مع الزملاء.
- استخدام مصادر إضافية.
- ربط المفاهيم النظرية بالتطبيقات العملية.
- متابعة التطورات الحديثة في مجال تراسل البيانات.

الاسبوع الاول
Introduction to programming

الهدف العام : سنتناول في هذه المحاضرة النقاط التالية:

١. مقدمة عن برنامج البايثون.

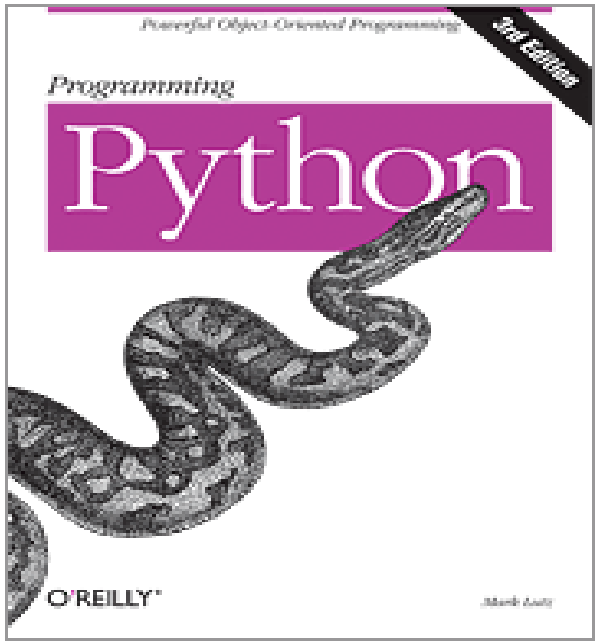
٢. لماذا نستخدم البايثون واختياره .

٣. ما هو الفرق بين المفسر والمترجم في بايثون

٤. الادخال والإخراج والمعالجة في لغه البايثون مع إعطاء امثله عمليه .

م	الجلسة الاولى	استراحة	الجلسة الثانية
مواضيعها	Introduction to programming	10 دقيقة	Input, Processing, and output
	Why Choose Python?		Examples
	Compilers and Interpreters		Using Python
			مناقشة + Quiz
زمنها	50 دقيقة		50 دقيقة

Introduction to Python Programming



Python is one of the most widely used programming language known for its simplicity and versatility. It's used in web development, data science, AI, automation, cybersecurity, and more. Whether you're a beginner or experienced, Python is an excellent language to learn due to its clear syntax and strong community support.

Python was developed by Guido van Rossum.

Introduction to Programming with Python

What is Programming?

Programming is the process of writing instructions for a computer or any other device to perform a specific task. When you write a program, you define a series of instructions that the device will follow to perform the task you want.

Programs for Humans...

while music is playing:
- Left hand out and up
- Right hand out and up
- Flip Left hand
- Flip Right hand
- Left hand to right shoulder
- Right hand to left shoulder
- Left hand to back of head
- Right hand to back of head
- Left hand to right hip
- Right hand to left hip
- Left hand on left bottom
- Right hand on right bottom
- Wiggle
- Wiggle
- Jump

Programs for Python...

the clown ran after the car and the car ran into the tent and the tent fell down on the clown and the car.

Why Choose Python?

1 **Easy to Learn**

Its syntax is simple and similar to human language, making it accessible for beginners.

2 **many-sided**

Python can be used for web development, artificial intelligence, automation, and more.

3 **Large Community**

Thousands of free libraries and resources are available, providing extensive support.

4 **Cross-Platform**

Python works on Windows, macOS, and Linux, offering flexibility across different operating systems.

Compilers and Interpreters

1.What is compilers:

A compiler translates the entire program into machine code before execution. This means that once the program is compiled, it can be executed without needing the compiler again.

Advantages of a Compiler:

- Faster execution (after compilation).
- Detects all errors before running the program.
- Produces an independent executable file.

Disadvantages:

- Compilation takes time.
- Debugging is harder since all errors appear at once.

Examples of Compiled Languages:

- C
- C++
- Java (compiles to bytecode, then interpreted by the JVM)

Compilers and Interpreters

2.What is Interpreter :

An interpreter reads the code line by line and executes it immediately, instead of converting the entire program at once.

Advantages of an Interpreter:

- Easier debugging (errors appear one at a time).
- More flexible, as code can be tested quickly.

Disadvantages:

- Slower execution because the code is processed at runtime.
- The program must be interpreted every time it runs.

Examples of Interpreted Languages:

- Python
- JavaScript
- Ruby

Python is an interpreted language, making it more interactive and easier to debug.

Compilers vs. Interpreters

Interpreter

Reads the code line by line and executes it immediately, making debugging easier and code more flexible for quick testing.

Examples: Python, JavaScript, Ruby.

Compiler

Translates the entire program into machine code before execution, resulting in faster execution after compilation and detecting all errors before running the program.

Examples: C, C++, Java.

IPO: Input, Processing, Output

1 2 3

Input

Receiving data from the user or a file, such as name and age.

Processing

Performing calculations, logical operations, or transformations on data, like calculating next year’s, age.

Output

Displaying the result on the screen or storing it in a file, such as a greeting message with the calculated age.

Semple examples for input , processing and output in python

```
# Input: Get user information
name = input("Enter your name: ")
age = int(input("Enter your age: "))

# Processing: Calculate next year's age
next_year_age = age + 1

# Output: Display the result
print("Hello,", name + "! Next year, you will be", next_year_age, "years old.")
```

Output:

Enter your name: Haider

Enter your age: 31

Hello, haider! Next year, you will be 32 years old.

Using Python

1. Python in Game Development

Python is used in designing electronic games, including Battlefield 2, Eve Online, Civilization IV, and World of Tanks. Its versatility and ease of use make it a popular choice for game developers.

Battlefield 2

One of the popular games designed using Python.

Eve Online

Another well-known game that utilizes Python.

Using Python

2. Python for 3D Programs

Blender

A 3D creation suite developed using Python.

MAYA

Another program created using Python for 3D design.

Dropbox

A file hosting service created using Python and other languages.

3. Major Companies Using Python

Yahoo

Uses Python in various programs and products.

IBM

Relies on Python for its programs and private work.

YouTube

Employs Python in its infrastructure and services.

Python's flexible Applications

Python is not just for beginners; it's used to create large and well-known programs. Its versatility makes it essential in the programming world and a required skill in the labor market.

1. **Web Development**
   Used in frameworks like Django and Flask.
2. **Data Science**
   Popular for data analysis and machine learning.
3. **Automation**
   Used for scripting and automating tasks.

Thank’s For Listening

الاسبوع الثاني
Python Variable Type

الهدف العام :

تعريف الطلاب بمفهوم المتغيرات في البايثون وكيفية استخدامها، وشرح أنواع البيانات الرقمية وتعريف الطلاب بوضيفة داله الطباعه والادخال وكيفية استخدامهما لعرض وادخال النصوص او البيانات وتعزيز فهم الطلاب للتركيب اللغوي الأساسي في بايثون.

مدة المحاضرة: ساعتان (نضري ساعة وعملي ساعة)

م	الجلسة الاولى	استراحة	الجلسة الثانية
مواضيعها	Python Variables Explained	10 دقيقة	Numeric Variables
	Variables Example		Integers
	Print Function		Flaots+ Complex Numbers
	Uses of the input function		مناقشة Quiz +
زمن الجلسة	50 دقيقة		50 دقيقة

Python Variables Explained

١. Variable: A named piece of memory that can store a value.
   - Usage:
   - Compute an expression's result.
   - store that result into a variable,
   - and use that variable later in the program.

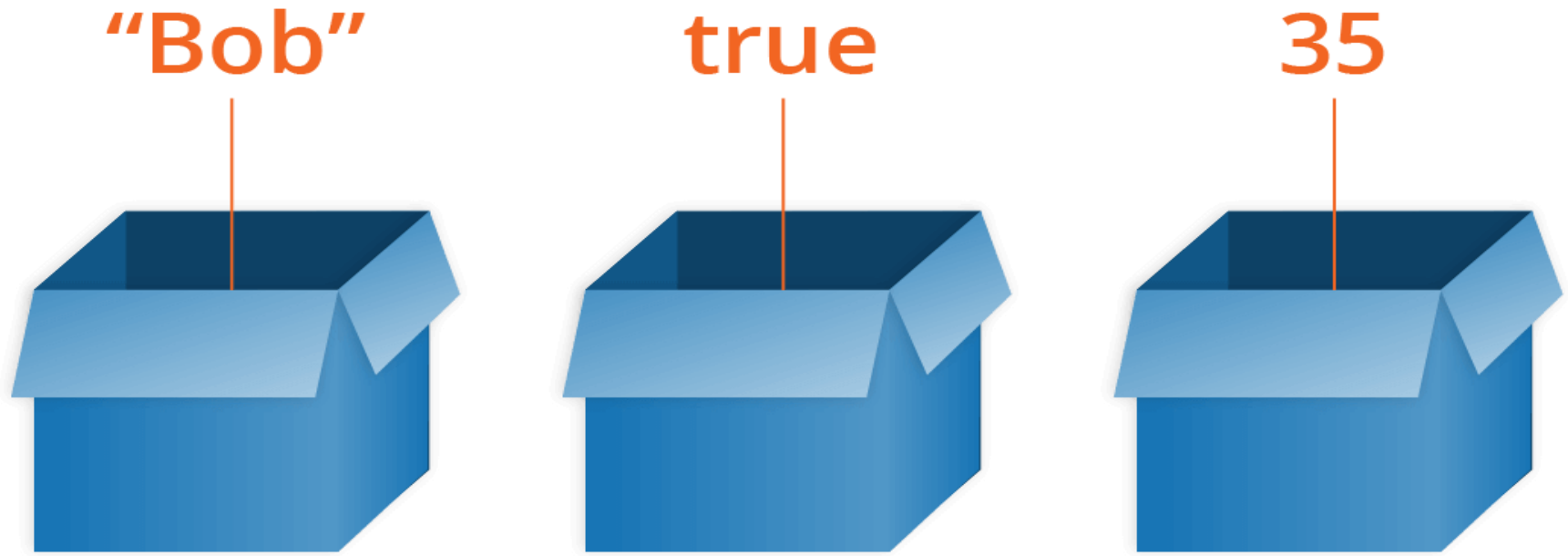
٢. **Assignment statement:** Stores a value into a variable.
   - Syntax:

```
name = value
```

   - A variable that has been given a value can be used in expressions.

Variables

It is a storage space with a distinctive name in which we put data of the same type for later use. The variable name is not repeated, and it is also required that the data in the same variable be of the same type, whether it is Integer or other.



أنواع المتغيرات

المتغير	الرمز	حجم – البت	مثال
بايت	byte	8	
عدد قصير	short	16	n=5
عدد صحيح	int	32	n=300
عدد كبير	long	64	n=698552215522
عدد عشري	float	32	n=5.56
عدد مضاعف	double	64	n=52.2566225
النص	string	–	n='ahmed'
تعبيري	boolean	–	n= true
عام	var	–	n= 5 , n = 'ahmed'
حرف	char	–	n='A'

Variables Example

Examples of declaring variables

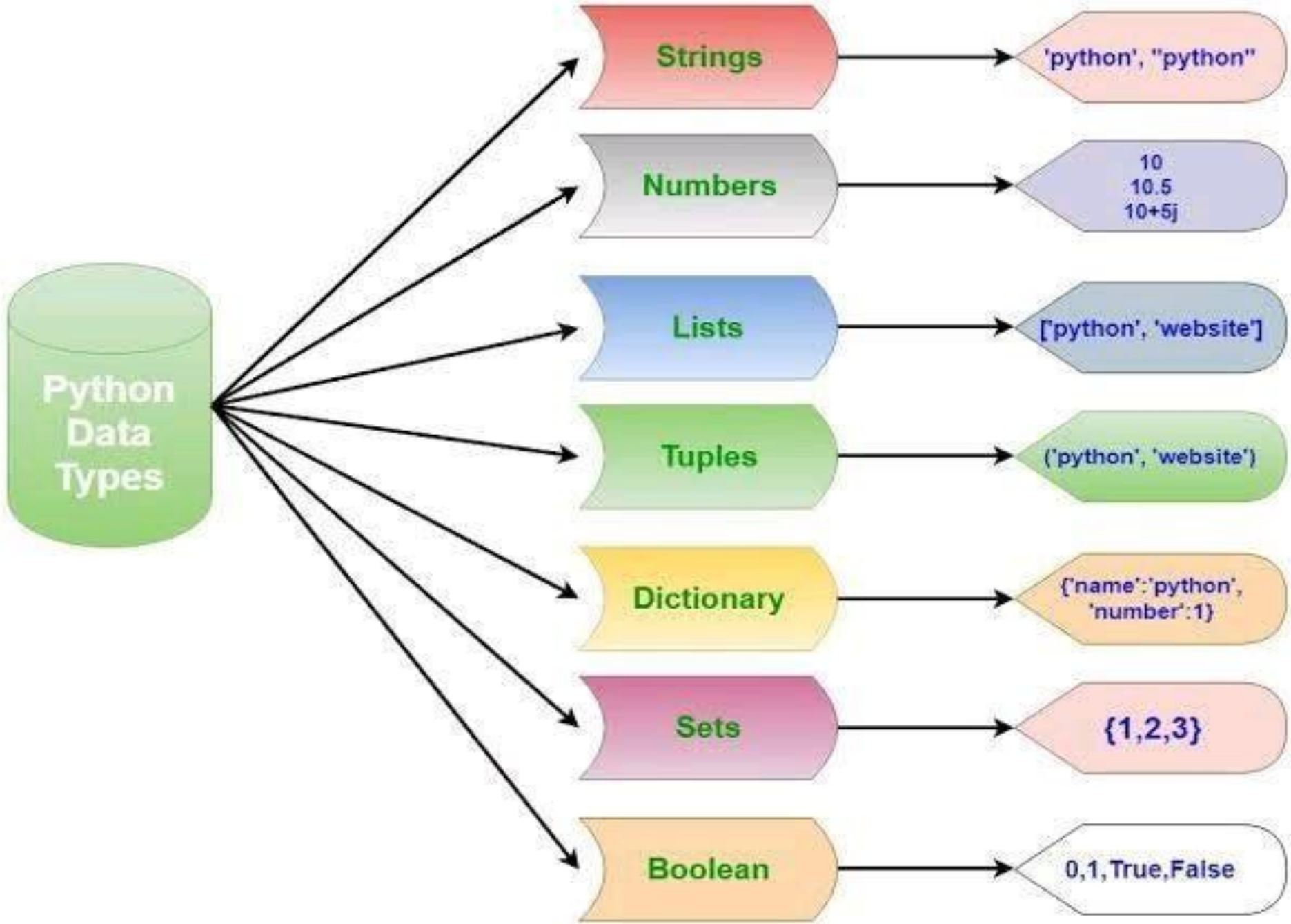
Example 1

Test.py

```
var = 5       # هنا قمنا بتعريف متغير اسمة var وقيمته 5
print(var)    # هنا قمنا بطباعة المتغير
```

We will see the result :

5


Python Data Types
Strings
'python', "python"
Numbers
10
10.5
10+5j
Lists
['python', 'website']
Tuples
('python', 'website')
Dictionary
{'name':'python', 'number':1}
Sets
{1,2,3}
Boolean
0,1,True,False

Print Function

is a function used to display text, variables, or any other object on the :The print() function .output screen

How do we use the print() function?

Name= "احمد"
Age= 30
Print("الاسم: ",name, " العمر :",age) Run
الاسم: احمد العمر: 30 ←

1. print (3)
Run → ٣

Run 1. print (3+7)
→ ١٠

Hello -- Run- 1. print ("Hello World!")
World!

1.print ("I Love Python") run → I Love Python

2. print ()

3. print ("print (I Love Python)") run →
print (I Love Python)

Print Function

- Print Produces text out on the console.
- Syntax:

  print (*"Message"*)

  print *Expression*

  - Prints the given text message or expression value on the console, and moves the cursor down to the next line.

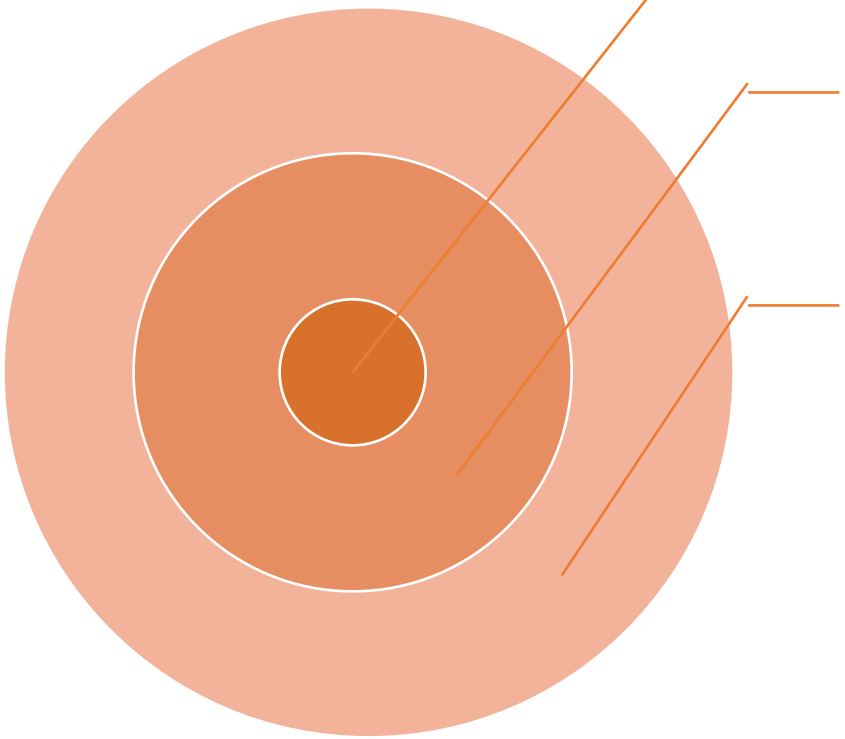
    print *Item1, Item2, ..., ItemN*

  - Prints several messages and/or expressions on the same line.

- Examples:

```
print("hello ,your name")
age=int(input("Enter your age:"))
print("Next year,you will be", age+1,"years old")
```

Uses of the input function



1. It prompts the user to enter data.

2. It suspends code execution until the user enters the required data.

3. The data entered by the user is stored within the program itself. You can try the following code to verify this.

Examples of declaring variables

Example 1

1. name = input ("What is your name? \n")

2. print (name)
3. Print (“Hello”+ name)

? Run →

Numeric Variables

1 **Integers**

Store whole numbers without a decimal point.

2 **Floats**

Store decimal numbers with a decimal point.

3 **Complex Numbers**

Store complex numbers with the imaginary number (j).

Numeric Variables

1 Integers

Integers are numbers we're all know it , like 1, 2, 3, -1, -2, and so on. In Python, we use the int type to represent these numbers.

```
x = 10
y = -5
z = 0

print(type(x))                    # سيظهر class<
'int'>
print(type(y))                    # سيظهر class
<'int'>
print(type(z))                    # سيظهر class
<'int'>
```

Numeric Variables

2 Floats

Decimals are numbers that contain a decimal point, such as 3.14, -2.5, or 0.001. In Python, we use the float type to represent these numbers.

```
a = 3.14
b = -2.5
c = 0.0

print(type(a))                    # سيظهر class
<'float'>
print(type(b))                    # سيظهر class
<'float'>
print(type(c))                    # سيظهر class
<'float'>
```

Numeric Variables

3 Complex Numbers

Numbers that consist of a real part and an imaginary part, and are written in the form a + bj, where a is the real part, b is the imaginary part, and j is the imaginary unit (the square root of -1).

```
d = 2 + 3j
e = -1 - 1j

print(type(d))                  # سيظهر class
<'complex'>
print(type(e))                  # سيظهر class
<'complex'>
```

Thank’s For Listening

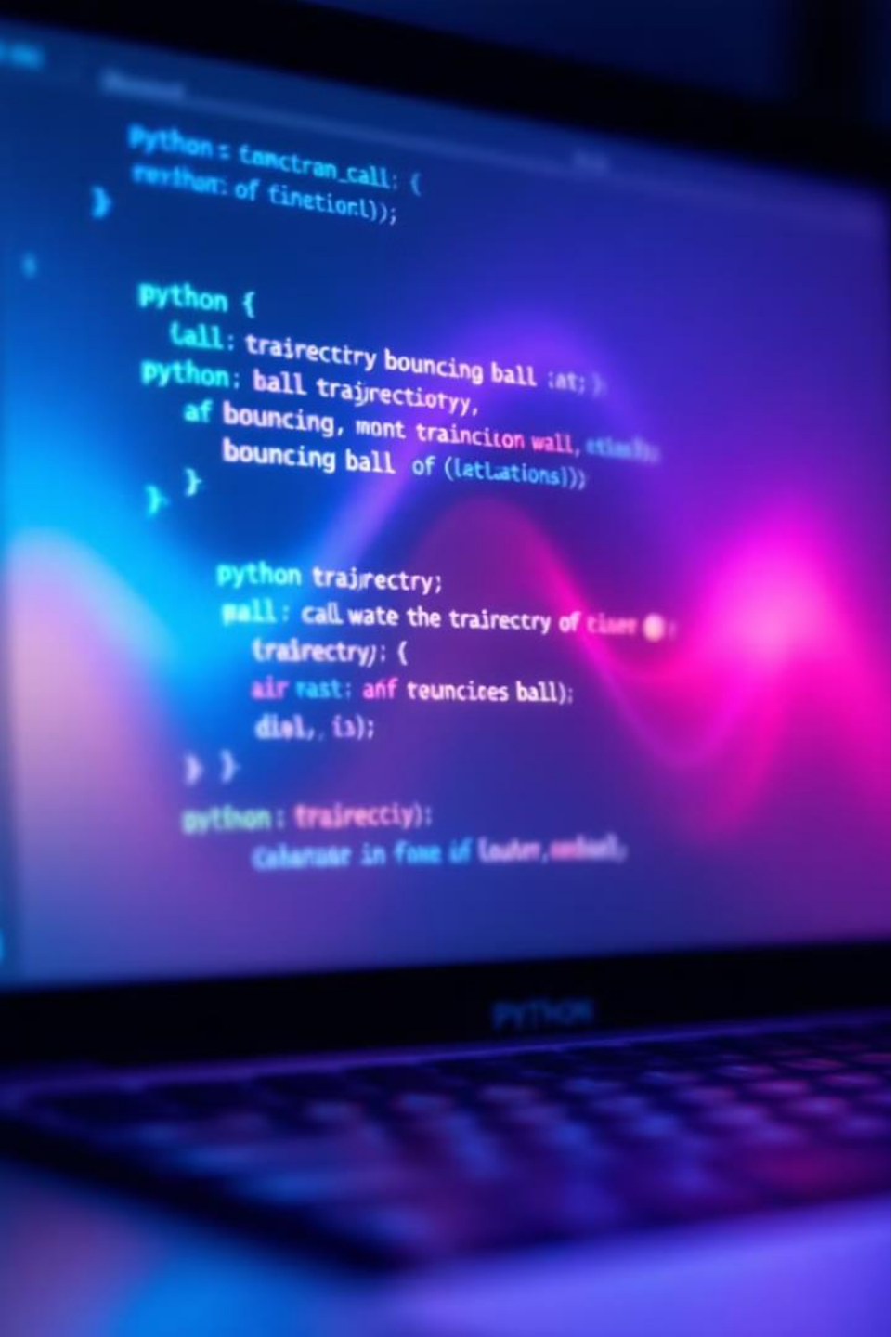
الاسبوع الثالث
Arithmetic Operators in Python

الهدف العام :

تعريف الطلاب بمفهوم المعاملات الحسابية في بايثون ،يتناول ماهية هذه المعاملات، وكيفية عملها، واستخداماتها العملية في البرمجة. كما يناقش أولوية المعاملات ومعاملات التعيين المعززة لمساعدة الطلاب على كتابة أكواد أكثر كفاءة.

مدة المحاضرة: ساعتان (نضري ساعة +عملي ساعة)

م	الجلسة الاولى	استراحة	الجلسة الثانية
مواضيعها	Arithmetic Operators in Python	10 دقيقة	Operator Precedence
	Performing Calculations		The Augmented Assignment Operators
	Floating-Point and Integer Division		Examples
			مناقشة Quiz +
زمن الجلسة	50 دقيقة		50 دقيقة



Arithmetic Operators in Python

Arithmetic operators are symbols with specific meanings used in mathematical operations. In Python, these operators allow us to perform various calculations efficiently. Understanding these operators is fundamental to programming as they form the building blocks of computational logic.

This presentation will explore the different types of arithmetic operators in Python, how they work, and their practical applications in coding. We'll also cover operator precedence and augmented assignment operators to help you write more efficient code.

Table of operations used in arithmetic operations

شرح الكود	مثال	رمزه	إسمة
أعطي a قيمة b	a = b	=	**Assignment**
أضف قيمة b على قيمة a	a + b	+	**Addition**
إطرح قيمة b من قيمة a	a - b	-	**Subtraction**
أضرب قيمة a بقيمة b	a * b	*	**Multiplication**
أقسم قيمة a على قيمة b	a / b	/	**Division**
للحصول على آخر رقم يبقى عندما نقسم قيمة a على قيمة b	a % b	%	**Modulo**

Performing Calculations

Operator= (Assignment Operator)

Symbol: =

Used to assign a value to a variable

Example: a = 5

#Here we define a variable named 'a' and give it the value 5

Example: b = a

#Here we define a variable 'b' and give it the same value as variable 'a'

The assignment operator is the foundation of variable manipulation in Python. It allows us to store values in memory locations that we can reference and modify throughout our program. When we use the assignment operator, we're telling Python to associate a specific value with a variable name.

Addition and Subtraction Operators

Subtraction (-)

Used to subtract one value from another

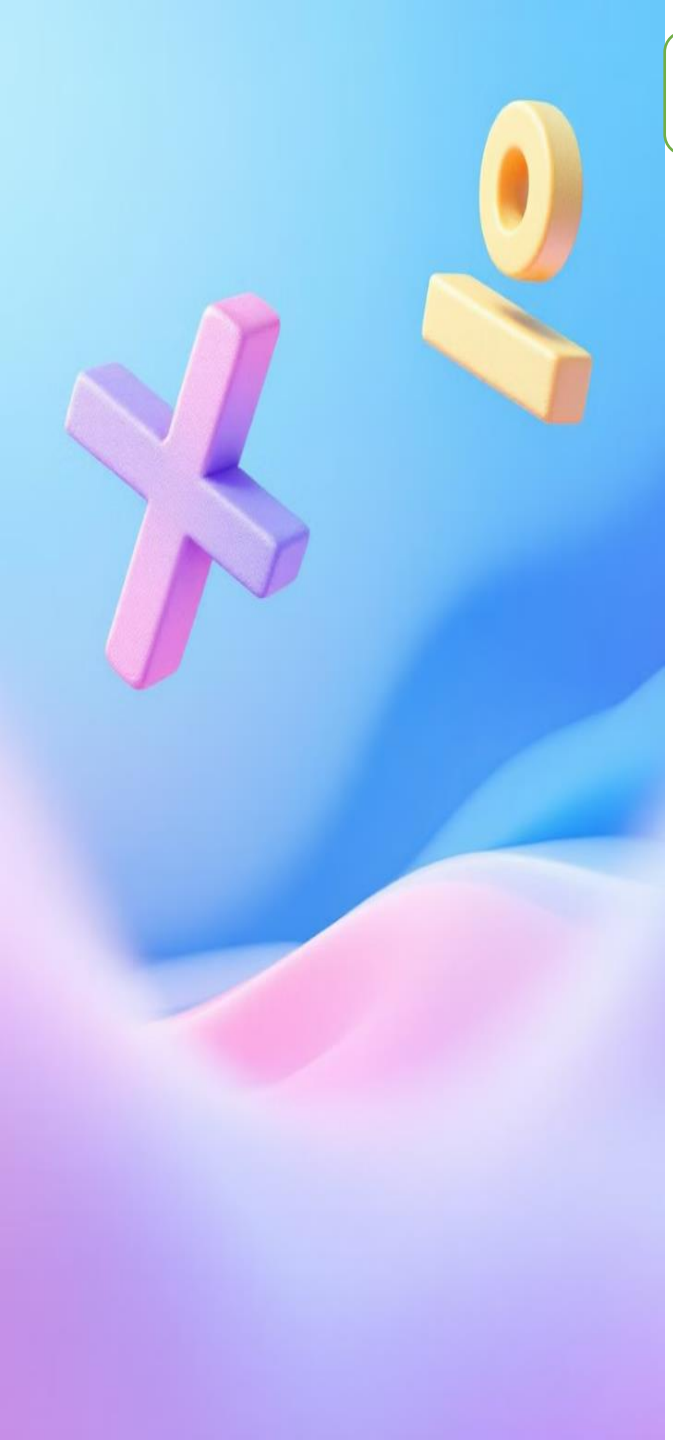
```
a = 3
b = 4
c = a - b  # c = -1
```

Addition (+)

Used to add values together

```
a = 3
b = 4
c = a + b  # c = 7
```

Addition and subtraction operators work just as they do in mathematics. They can be used with integers, floating-point numbers, and even for string concatenation in the case of the addition operator. These operations form the basis of arithmetic calculations in Python.



Performing Calculations

Multiplication and Division Operators

Multiplication (*)

Used to multiply values

```
a = 6
b = 5
c = a * b  # c = 30
```

Division (/)

Used to divide values

```
a = 8
b = 5
c = a / b  # c = 1.6
```

Floor Division (//)

Divides and removes decimal part

```
a = 8
b = 5
c = a // b  # c =1
```

Modulo Operator

Symbol: %

Returns the remainder of division when wanting a whole number result

Example

```
a = 8
b = 5
c = a % b
results in    # c = 3
```

The modulo operator is particularly useful in programming for determining cyclical patterns, checking divisibility, and handling cases where you need to work with remainders. It's commonly used in algorithms that require looping through a fixed range of values repeatedly.

integr= .. □(ivion)

5 / 2

5 /* / 2 = 2

5 / / 2 + 2.5.5

2. Floating - Point and Integer Division

Floating-Point Division (/)

Returns result as a decimal number (float)

- 10 / 3 = 3.3333...
- 6 / 2 = 3.0

Integer Division (//)

Returns result as a whole number (int)

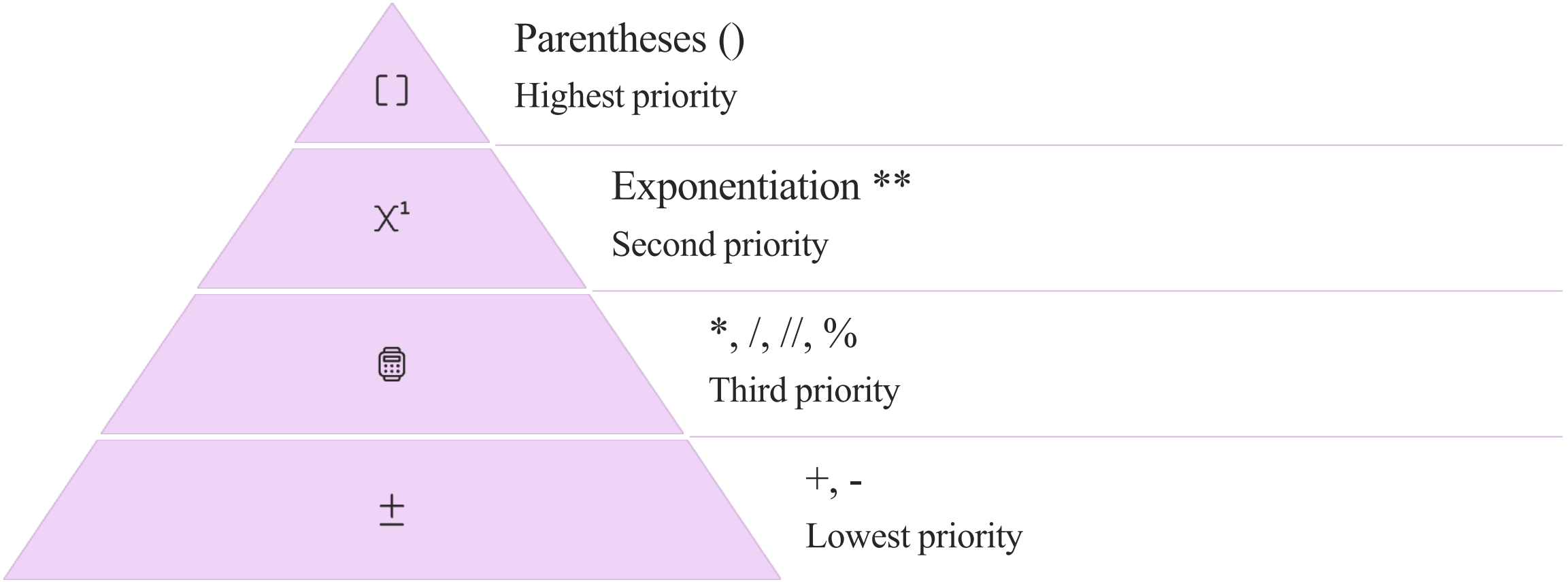
- 10 // 3 = 3
- 6 // 2 = 3

Note

Even with float inputs, integer division truncates the decimal part

- 10.5 // 2 = 5.0

3.Operator Precedence



Understanding operator precedence is crucial for writing correct mathematical expressions in Python. When multiple operators appear in an expression, Python follows these rules to determine the order of operations. For example, in the expression 10 + 5 * 2, multiplication happens first, resulting in 20, not 30.

Precedence Examples

```
result = 10 + 5 * 2          # الأولوية للضرب
print(result)                # 20
result = (10 + 5) * 2        # الأولوية للأقواس
print(result)                # 30
```

```
print(100 + 5 * 3)


# Multiplication has higher precedence than addition, and
needs to be evaluated first.
The calculation above reads 100 + 15 = 115
```

```
print((6 + 3) - (6 + 3))

#Parenthesis have the highest precedence, and
need to be evaluated first.
The calculation above reads 9 - 9 = 0
```

Augmented Assignment Operators

Description	Long Form	Operator
Add and assign	x = x + 5	+=
Subtract and assign	x = x - 3	-=
Multiple and assign	x = x * 4	*=
Divide and assign	x = x / 4	/=
Floor divide and assign	x = x // 2	//=
Modulo and assign	x = x % 3	%=
Exponentiate and assign	x = x ** 2	**=

Practical Examples of Augmented Operators

- Initialize Variable

  Start with x = 10

  - Add and Assign

    x += 5 updates x to 15

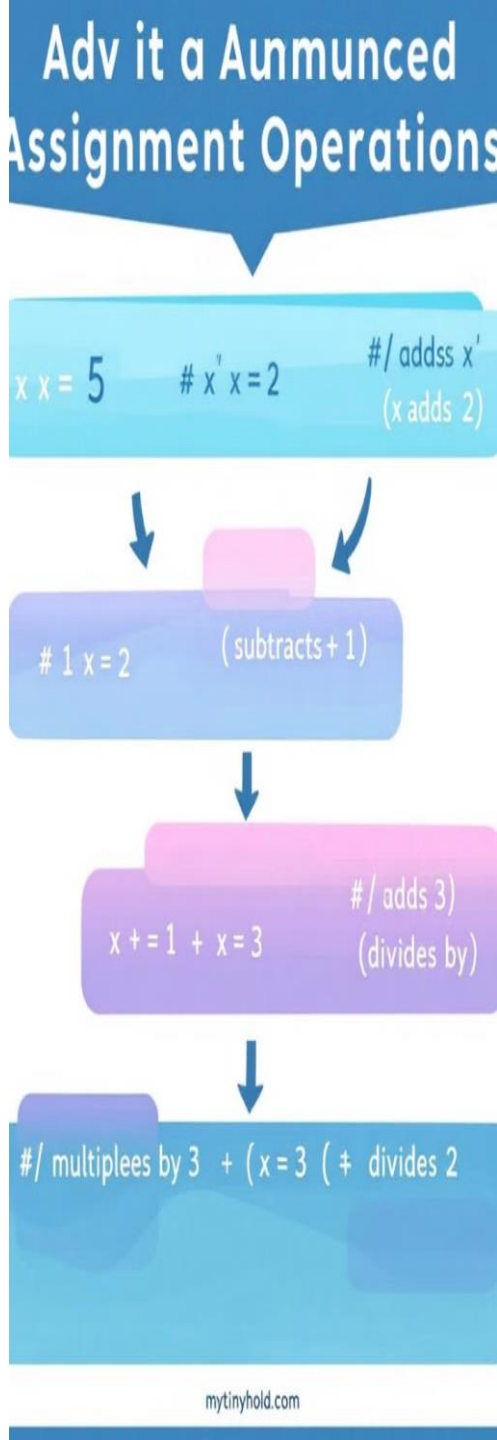
    - Subtract and assign

      x -= 2 updates x to 8

      - Multiple and assign

        x *= 3 updates x to 30

Augmented assignment operators provide a more concise way to update variables based on their current value. They combine an arithmetic operation with assignment in a single step, making code more readable and efficient. These operators are commonly used in loops, counters, and accumulation patterns.



Examples of Augmented Operators

1. Add and Assign (+=)

هنا وضعنا في المتغير a قيمة أكبر من صفر، ثم وضعنا قيمة الـ Unary-Plus لها في المتغير b #

```
a = 10
b=3
b += a              # b = 3+(10) = 13

print( b)
```

هنا وضعنا في المتغير a قيمة أصغر من صفر، ثم وضعنا قيمة الـ Unary-Plus لها في المتغير b #

```
a = -10
b=3
b += a              # b = 3+(-10) = -7

print("b =", b)
```

Examples of Augmented Operators

2. Subtract and assign (-=)

هنا وضعنا في المتغير a قيمة أكبر من صفر، ثم وضعنا قيمة الـ Unary-Minus لها في المتغير b #

```
x = 10
b = 3
b -= x              # b = 3-(10) = -2

print( b)
```

هنا وضعنا في المتغير a قيمة أصغر من صفر، ثم وضعنا قيمة الـ Unary-Minus لها في المتغير b #

```
x = -10
b = 3
b -= X              # b = 3-(-10) = +13

print("b =", b)
```

Examples of Augmented Operators

3. Multiple and assign (*=)

```
x = 10
b=3
b *= x                # b =3* (10) = 30

print("b =", b)
```

Practical Examples of Augmented Operators

Divide and assign (/=)

X /= 3 updates x to 3.33333

Floor divide and assign (//=)

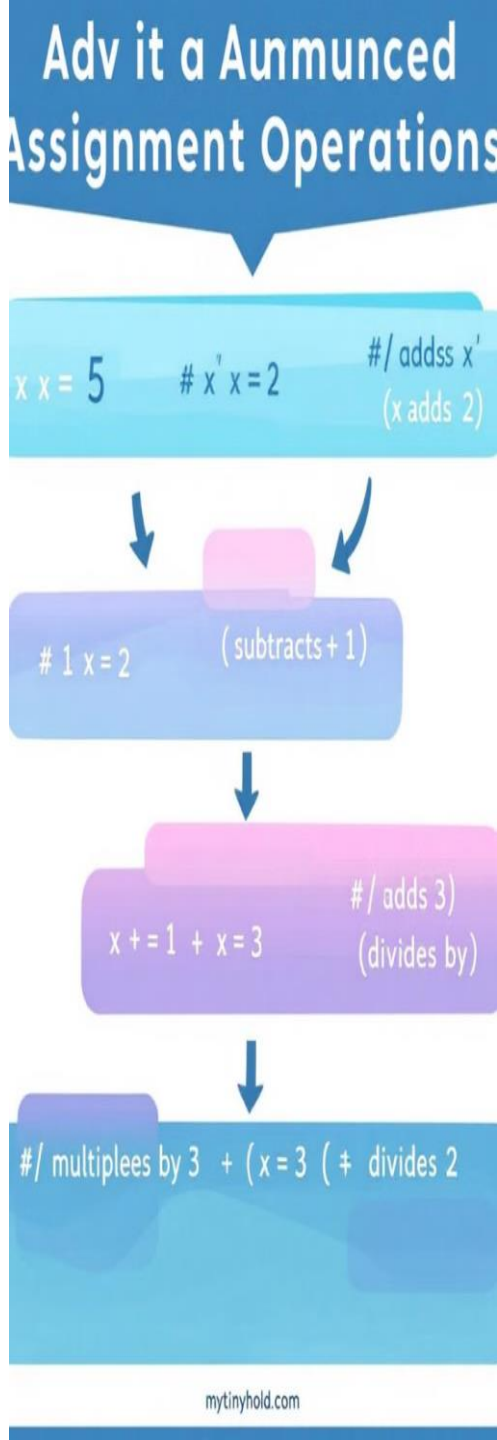
x //= 5 updates x to

Modulo and assign (%=)

x %= 2 updates x to 0

Exponential and assign (=)**

x **= 3 updates x to 1000



Examples of Augmented Operators

4. Divide and assign (/=)

```
x = 5

x /= 3

print(x)
```

----------2----------

```
x = 5
b = 3
b /= x              # b =3/ 5 = 0.6

print("b =", b)
```

Examples of Augmented Operators

5.Floor divide and assign (//=)

```
x = 5

x //= 3                # x = x // 3 = 0

print(x)
```

---------- Ex 2----------

```
x = 5
b = 3
b //= x                # b =3// 5 = 0

print("b =", b)
```

Examples of Augmented Operators

6. Modulo and assign (%=)

```
x = 5

x %= 3                  # x = x % 3= 2

print(x)
```

----------Ex 2----------

```
x = 10
b = 3
b %= x                  # x = x % 3= 10%3=3

print("b =", b)
```

Examples of Augmented Operators

7. Exponential and assign (=)**

```
b = 3
X=5
x **= b                # x = x ** 3= 125

print(x)
```

----------Ex 2----------

```
x = 10
b = 3
x **= b                # x = x ** 3= 1000

print("x =", x)
```

Thank’s For Listening

الأسبوع الرابع
Breaking & Suppressing & Formatting

الهدف العام :

تمكين من فهم تقسيم الجمل الطويلة وتنسيق الكود، التحكم في مخرجات الطباعة ، الأحرف الخاصة وتطبيق أساسيات التعامل مع النصوص والأرقام في لغة بايثون بشكل احترافي وفعال.

مدة المحاضرة: ساعتان (نضري ساعة +عملي ساعة)

م	الجلسة الاولى	استراحة	الجلسة الثانية
مواضيعها	Breaking Long Statements into Multiple Lines	10 دقيقة	Escape Characters
	Examples		Formatting Numbers
	Suppressing the print Function’s Ending Newline		Examples
	Examples		مناقشة Quiz +
زمن الجلسة	50 دقيقة		50 دقيقة

1. Breaking Long Statements into Multiple Lines

Python allows you to break long statements into multiple lines using parentheses () or backslashes \ .

Using Parentheses

```
result = (10 + 20 + 30 +
40 + 50 + 60 +
70 + 80 + 90)
print(result)
```

Using Backslash

```
result = 10 + 20 + 30 + \
40 + 50 + 60 + \
70 + 80 + 90
print(result)
```

Breaking Long Statements into Multiple Lines

1. Using Parentheses ():

It is commonly used to split function calls that contain multiple parameters or complex arithmetic expressions. For example:

```
my_list = (
    "item1",
    "item2",
    "item3"
    )
print("list is : " ,my_list)   run →
```

```
# list is :  ('item1', 'item2', 'item3')
```

Breaking Long Statements into Multiple Lines

2. Using Parentheses {}:

Used to split long dictionaries and collections. Each key-value pair in the dictionary or each element in the collection can be placed on a new line:

```
my_dict = {
    "key_one": "value one",
    "key_two": "value two",
    "key_three": "value three"
}
my_set = {
    "element one" ,
    "element two" ,
    "element three"
}
print(my_dict)
print(my_set)
```

```
# {'key_one': 'value one', 'key_two': 'value two',
'key_three': 'value three'}
{'element three', 'element two', 'element one'}
```

Breaking Long Statements into Multiple Lines

3. Using Backslash \ :

using the backslash (\) character at the end of a line to show that the statement extends to the next line. Here's an example of a Python line break in a string:

EX:

```
long_string = "This is a very long string that \
spans multiple lines \
for readability."
print("long string is : " ,long_string)
```

run →

```
# long string is :  This is a very long string that spans multiple lines for readability.
```

2. Suppressing Print Newlines

In Python, print() by default outputs its content followed by a newline character, \n, which moves the cursor to the next line. but The end parameter in the print() function allows you to control what is printed at the end of the print statement. By default, end is set to \n.

Example

```
print("Hello, ", end="")
print("World!")
```

Practical use cases:

1. Print multiple items on the same line:

end=‘ ‘

end=‘,’

Ex:

```
numbers = [1, 2, 3, 4, 5]
for num in numbers:
    print(num, end=‘ ‘)
```

2. Creating Progress Bars:

end=" can be used to print multiple symbols on the same line to create a progress bar effect.

2. Creating Progress Bars:

رقم التكرار	أمر الطباعة (print('*', end='', flush=True))	المخرجات بعد كل تكرار
1	print('*', end='', flush=True)	*
2	print('*', end='', flush=True)	**
3	print('*', end='', flush=True)	***
4	print('*', end='', flush=True)	****
5	print('*', end='', flush=True)	*****
6	print('*', end='', flush=True)	******
7	print('*', end='', flush=True)	*******
8	print('*', end='', flush=True)	********
9	print('*', end='', flush=True)	*********
10	print('*', end='', flush=True)	**********

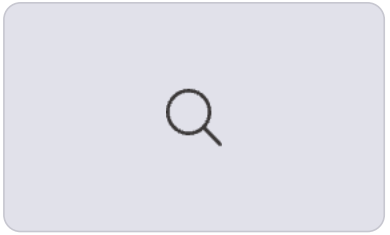
3. Escape Characters

An **escape character** is a character followed by a backslash (\). It tells the Interpreter that this escape character (sequence) has a special meaning. For instance, \n is an escape sequence that represents a newline. When Python encounters this sequence in a string, it understands that it needs to start a new line.

Escape Sequence & Meaning

Sr.No			
1	**\<newline>** Backslash and newline ignored	7	**\f** ASCII Formfeed (FF)
2	**** Backslash (\)	8	**\n** ASCII Linefeed (LF)
3	**\'** Single quote (')	9	**\r** ASCII Carriage Return (CR)
4	**\"** Double quote (")	10	**\t** ASCII Horizontal Tab (TAB)
5	**\a** ASCII Bell (BEL)	11	**\v** ASCII Vertical Tab (VT)
6	**\b** ASCII Backspace (BS)		

Escape Characters

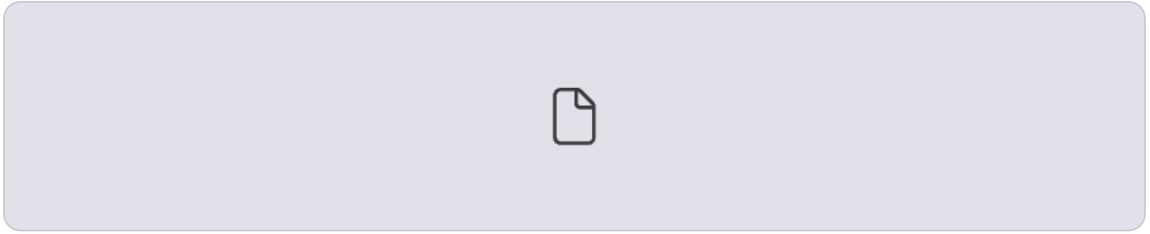


\n - Newline

Inserts a line break

\t - Tab

Inserts a tab space



\\ - Backslash

Inserts a literal backslash

Escape Characters Example

```
n\# newline
s='Hello\nPython’
print (s)
# Horizontal tab \t
s='Hello\tPython’
print (s)
# escape backslash \\
s=s = 'The \\character is called backslash’
print (s)
# escape single quote \’’
“ s=“Hello \’ Python \‘
print (s)
# escape double quote \””
s="Hello \"Python\” "
print (s)
```

```
# escape \b to generate ASCII backspace
s='Hel\blo’
print (s)
# ASCII Bell character
s='Hello\a’
print (s)
# ignore \
s = 'This string will not include \
backslashes or newline characters.’
print (s)
# form feed
s= "hello\fworld"
print (s)
# Octal notation
s="\101"
print(s)
# Hexadecimal notation
s="\x41"
print (s)
```

It will produce the following **output** –

```
This string will not include backslashes or newline characters.
The \character is called backslash
Hello     'Python’
Hello "Python"
Helo
Hello
Hello
Python
Hello Python
hello world
A
A
```

4. Formatting Numbers

Using f-strings (Python 3.6+)

Modern, readable approach for string formatting with embedded expressions.

Using format() method

Flexible string formatting compatible with Python 3.0+.

Using % Operator

Traditional C-style formatting, still supported for backward compatibility.

F-String Number Formatting

1

Define Variable

```
pi = 3.141592653589793
```

2

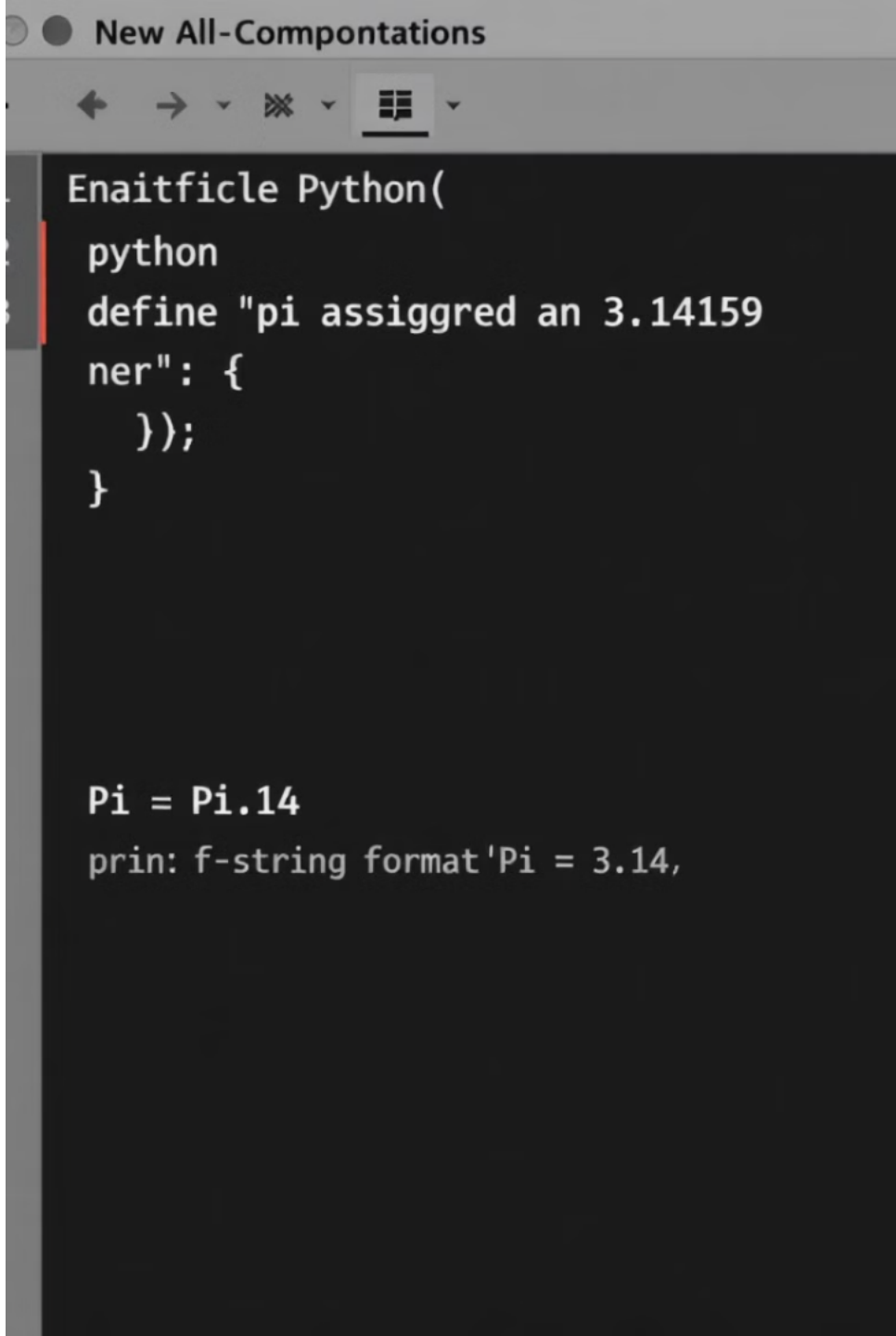
Create f-string (nf.:)

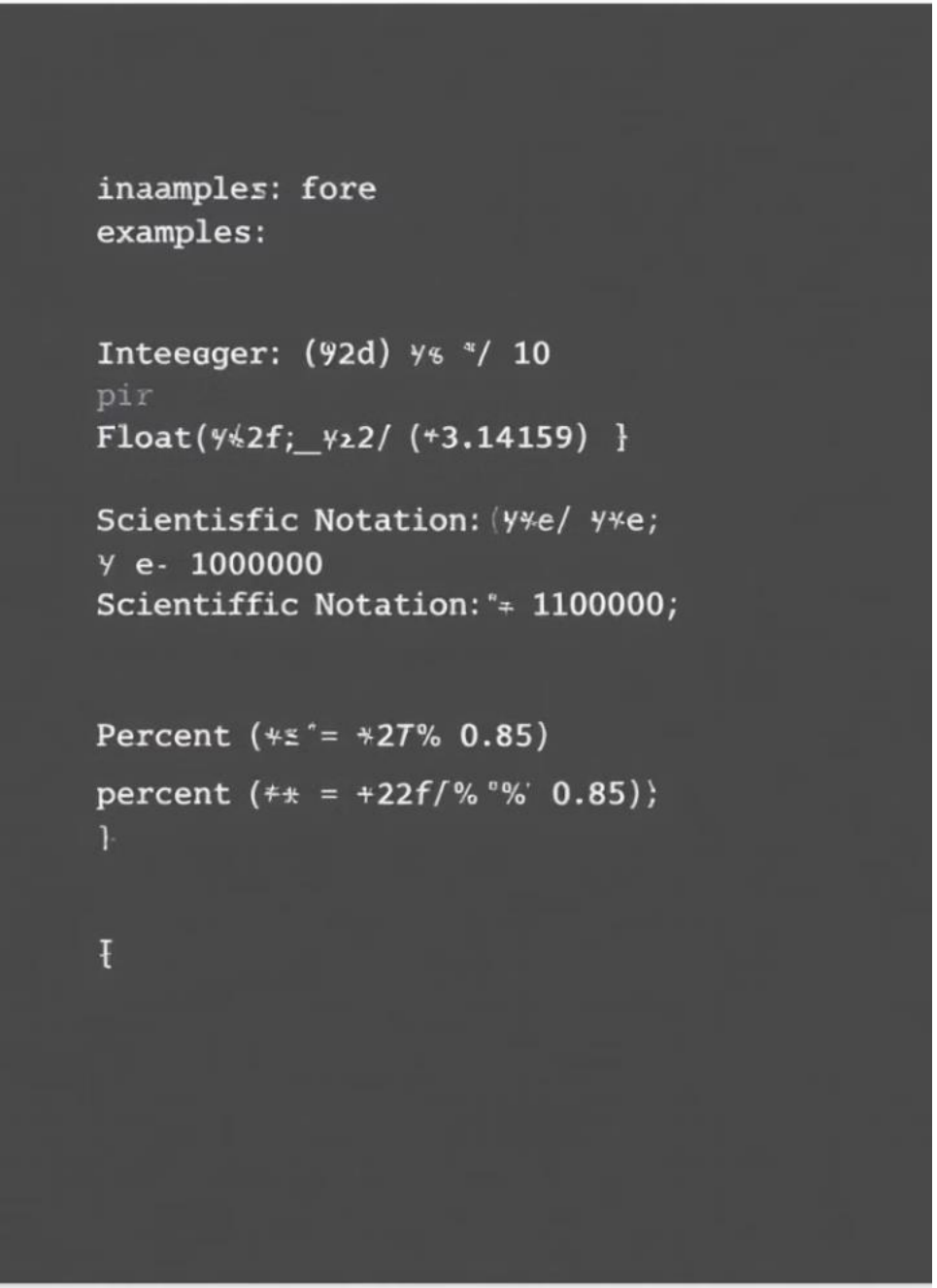
```
"Pi: {:.2f}".format(pi)
```

3

Result

Pi rounded: 3.14





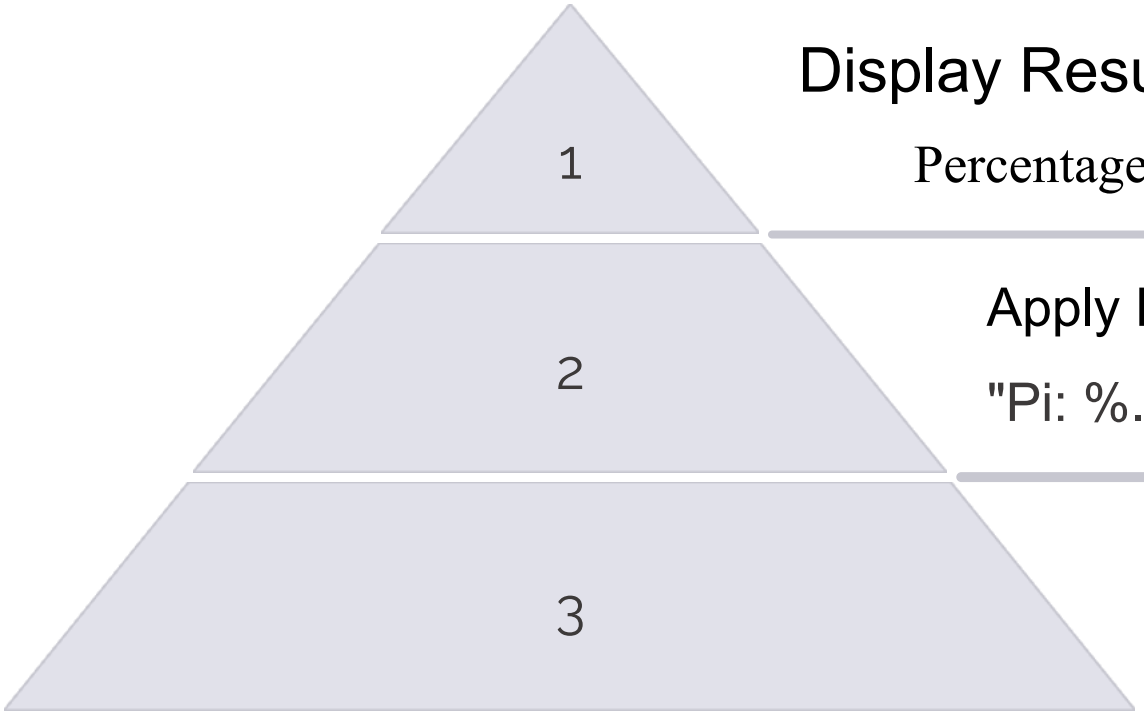
Number Formatting Options

Format	Description	Example
{:.2f}	2 decimal places	3.14
{:,}	Thousands separator	1,000,000
{:.2%}	Percentage with 2 decimals	75.00%
{:e}	Scientific notation	3.14e+00

Format() Method

```
                                        Define Variable
                                pi = 3.141592653589793
                    print( "the value is: {:.2f}".format(pi) )
                        print( f "the value is: {pi:.4f} " )
-------------------------------------------------------------------------
number=10000000          # add separator {:,}
                    print( f "the number is :{number:,}")
```

% Operator Formatting



Display Result

Percentage = 85.0 %

Apply Formatting

"Pi: %.2f" % pi

Define Variable

Percentage = 0.85

Example:

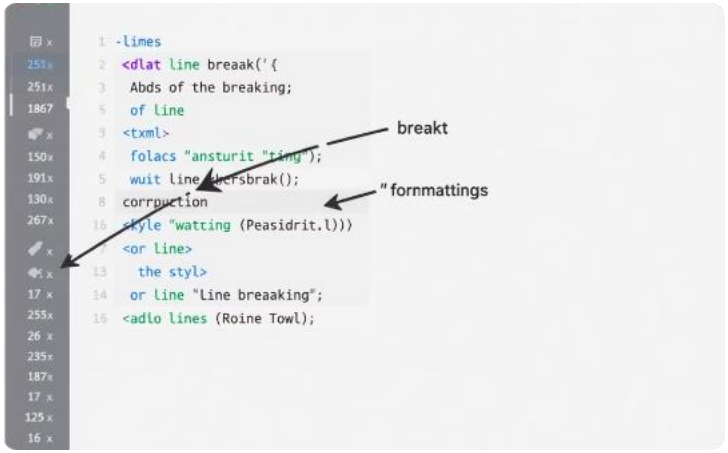
Percentage = 0.85

print("the percentage is : {:.1%}".format(p))

print(f "the percentage is : {p:.2%}")

output: the percentage is : 85.0%

Combining Techniques



Combined Example

```
result = (3.14159 + 2.71828 +
1.61803 + 0.57721)
print("The result is: ", end="")
print(f"{result:.3f}")
```

Output

The result is: 8.055

Best Practices

Combine these techniques for cleaner, more readable code. Follow PEP 8 style guidelines.

Thank ’s For Listening

الاسبوع الخامس
IF statement and logical operators

الهدف العام :

تزويدكم بالمعرفة والمهارات اللازمة لفهم وتطبيق مفاهيم أساسية في البرمجة باستخدام بايثون، وهي كالتالي:
فهم دور التعليقات (Comments) في الكود، إتقان أنواع المتغيرات (Variables)المختلفة و فهم كيفية التعامل مع الأخطاء (Errors) والتعامل معها بفعالية.
مدة المحاضرة: ساعتان (نضري +عملي)

م	الجلسة الاولى	استراحة	الجلسة الثانية
مواضيعها	Logical operators	10 دقيقة	Decision Structure
	The if Statement		Boolean Expressions and Relational Operators
	The if-else Statement		Examples
	Examples		مناقشة Quiz +
زمن الجلسة	50 دقيقة		50 دقيقة

logical operators

In Python, there are three logical operators. The result of a logical operation is a Boolean value, meaning it will be either True or False. Logical operators can be used in various programming contexts, such as with more than one logical expression to produce a final result. The following figure illustrates the logical operators provided by the Python language:

Python Logical Operators		
المعامل	مثال	الشرح
and	*x and y*	ترجع True فقط في حال كانت x قيم y صحيحة.
or	x or y	ترجع False فقط في حال كانت x قيم y خاطئة .
not	not x	تعكس القيمة الصحيحة الى خاطئة والخاطئة الى صحيحة.

1. And

Expression	Value of the Expression
true and false	false
false and true	false
false and false	false
true and true	true

2. OR

Expression	Value of the Expression
true or false	true
false or true	true
false or false	false
true or true	true

3. Not

Expression	Value of the Expression
not true	false
not false	true

Example :

1. And :

```
Print(1==2)      #False
Print(1< 2)      #True
Print (1< 3 and 1==2)      # False
Print (1<3 and 1>4)        # False
Print (1!=3)               #True
```

الشرطان متساويان

الشرطان مختلفان

2. OR :

```
Print(3==9 or 3==8)   #False
Print(3 < 9 or 3>9)   #True
```

3. Not

```
1=[1,2,3]
Print (3 not in 1)

x=[1,2,3,4]
print(5 not in x)
```

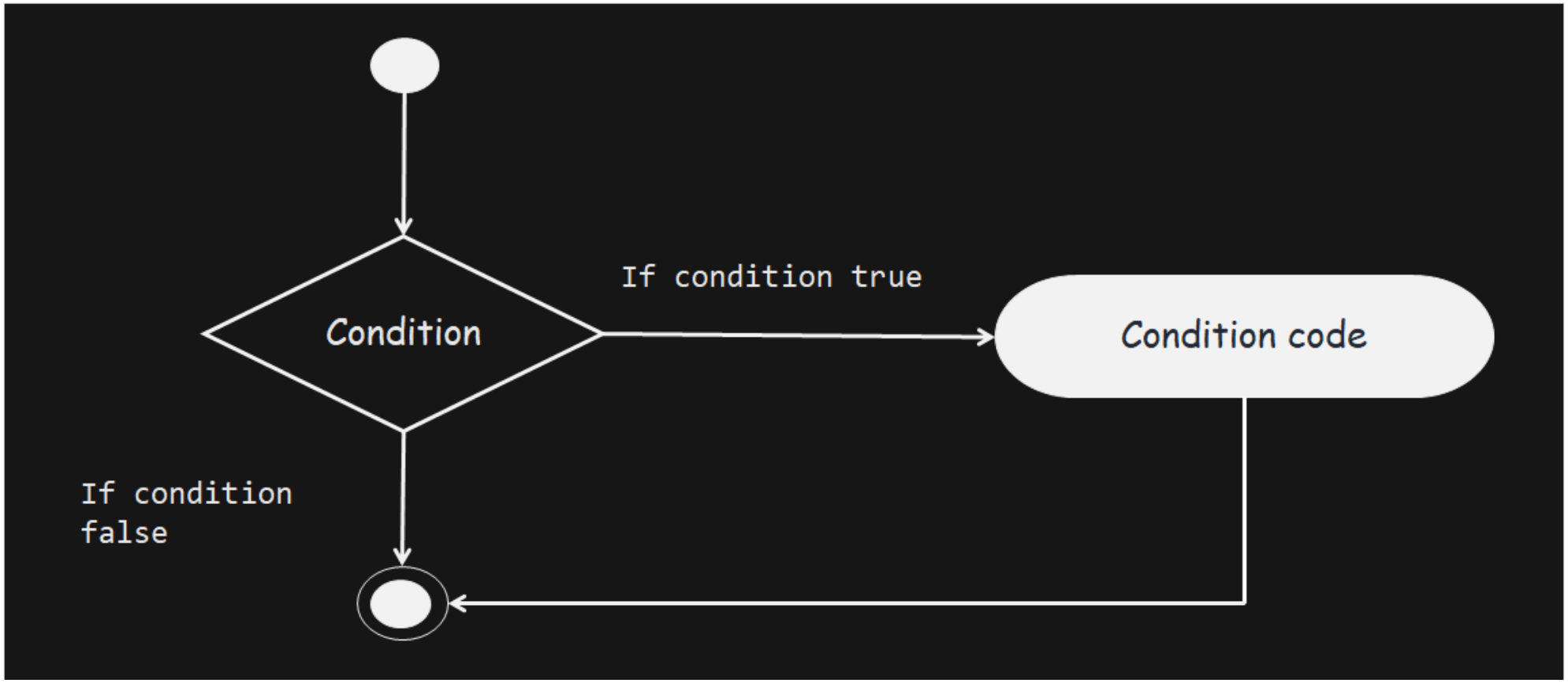
The if Statement

A conditional statement in Python refers to a decision-making statement that predicts the conditions that will occur during program execution and determines the actions to be taken according to certain conditions.

These conditions determine a decision by evaluating multiple expressions that produce either True or False as the result.

The program needs them to determine what action to take and which statements to execute if the result is True or False. Here is the general form of a typical decision-making structure found in programming:

The if Statement



If Statement in python

One of the conditional statements in Python is the if statement.

The "if" statement is written using the if keyword.

The Python programming language provides the following types of if statements:

1- if

2- if…else

1. If Statement

If statement consists of a logical expression followed by one or more statements as shown below:

```
if condition :
        statement(s)
```

If the logical expression evaluates to True, the statement block (code) inside the if statement is executed. If the logical expression evaluates to False, the if block is skipped and program execution continues.

Example :

```
x, y = 10, 100
if x == y:
        print('x = y: x is {} and y is {}'.format(x, y))
if x < y:
        print('x < y: x is {} and y is {}'.format(x, y))
if x > y:
        print('x > y: x is {} and y is {}'.format(x, y))
print('The end')
```

Output:

```
x < y: x is 10 and y is 100

The end
```

Note: In Python, True is assigned to any non-empty values, and False if they are zero or Null.

We can also use a shortened form of the if statement if we only have one statement to execute, we can put it on the same line as the if statement. Example:

Example :

```
x = 10
y= 100
if x < y:
        print("x < y: x is {} and y is {}".format(x, y) )
print("x < y: x is {} and y is {}".format(x, y) )
# you will get an error
```

```
# البرنامج لتحديد حالة الطالب بناءً على الدرجة

grade = 40
if grade >= 50:
    print("الطالب ناجح.")
if grade < 50:
    print("الطالب راسب.")
```

if statement and logical operators

Python allows the use of logical operators with conditional statements to link conditions together to produce the final result of the condition (if statement). Note the following example:

Example :

```
x ,y, z = 100, 10, 0
if x == 100 and y == 10:
        print('x is {} and y is {}'.format(x, y))
if x <= 100 or x < 200:
        print(' 100 <= x < 200 : x is {}'.format(x))
if not z:
        print('z is {} '.format(z))
```

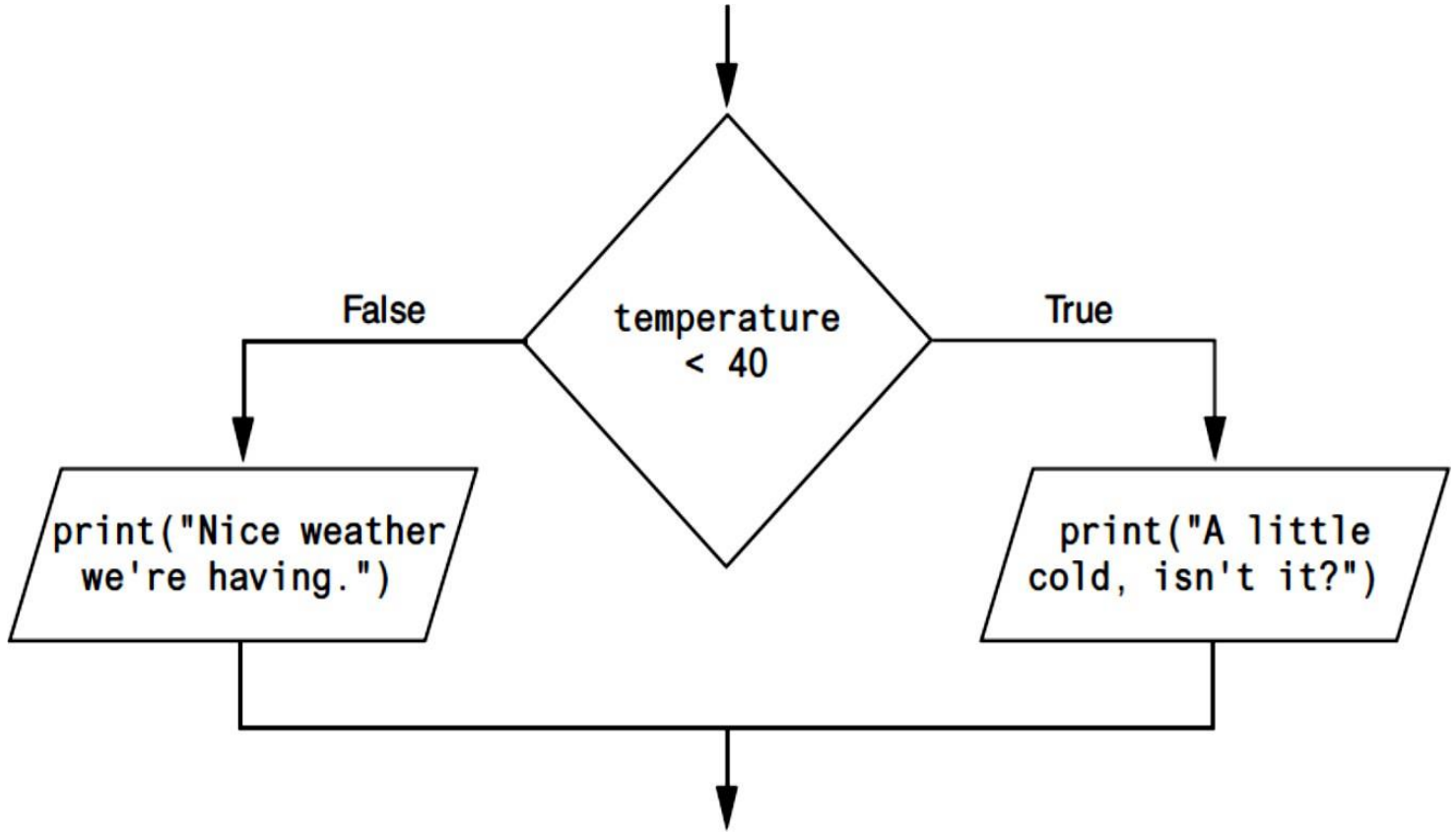
Output:

```
x is: 100 and y is: 10
 100 <= x < 200 : x is 100
z is 0
```

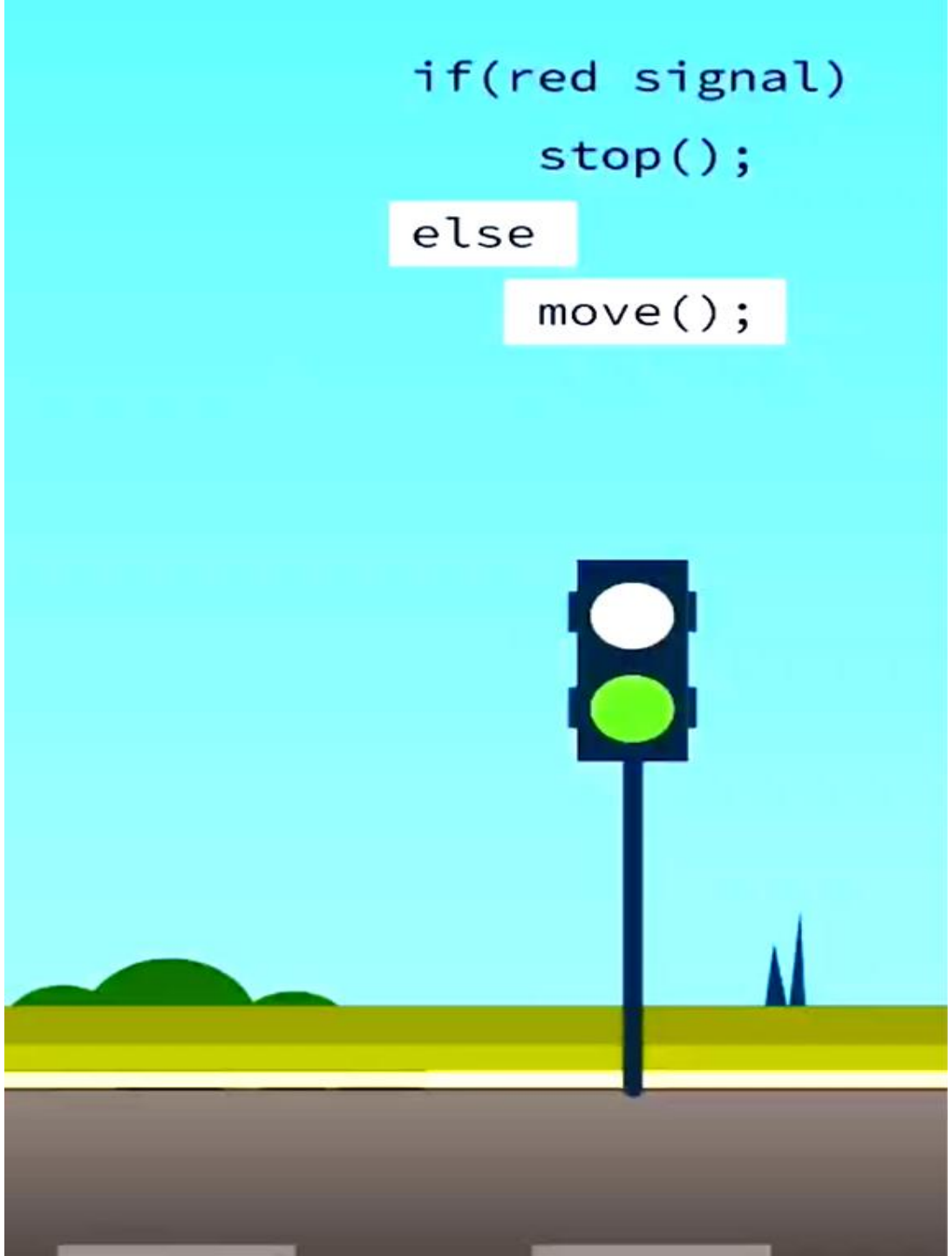
If- Else statement

Now, we will look at the dual alternative decision structure, which has two possible paths of execution—one path is taken if a condition is true, and the other path is taken if the condition is false. Figure below shows a flowchart for a dual alternative decision structure.

```
if condition:
   statement(s)
else:
   statement(s)
```



Simple Video explaining the if- else functions



Example :

```
temperature = 30   # يمكنك تغيير هذه القيمة لتجربة نتائج مختلفة

if temperature < 40:
        print("Nice weather we're having.")
else:
        print("A little cold, isn't it?")
print("End of weather report.")
```

البرنامج لتحديد حالة الطالب بناءً على الدرجة

```
grade = 50
if grade >= 50:
    print( "الطالب ناجح ")
else:
    print( "الطالب راسب")
```

Example :

```
a = 200
b = 33
if b > a:
  print("b is greater than a")
else:
  print("a is greater than b")
```

In this example a is greater than b, so the first condition is not true,
so we go to the else condition and print to screen that "a is greater than b".

Example :

1. Checking if a number is even or odd

```
num = 17                              # يمكنك تغيير هذا الرقم

if num % 2 == 0:
        print(f" العدد{} زوجي " )
else:
        print(f" العدد {} فردي " )
```

Output :

```
العدد ١٧ فردي
```

Example :

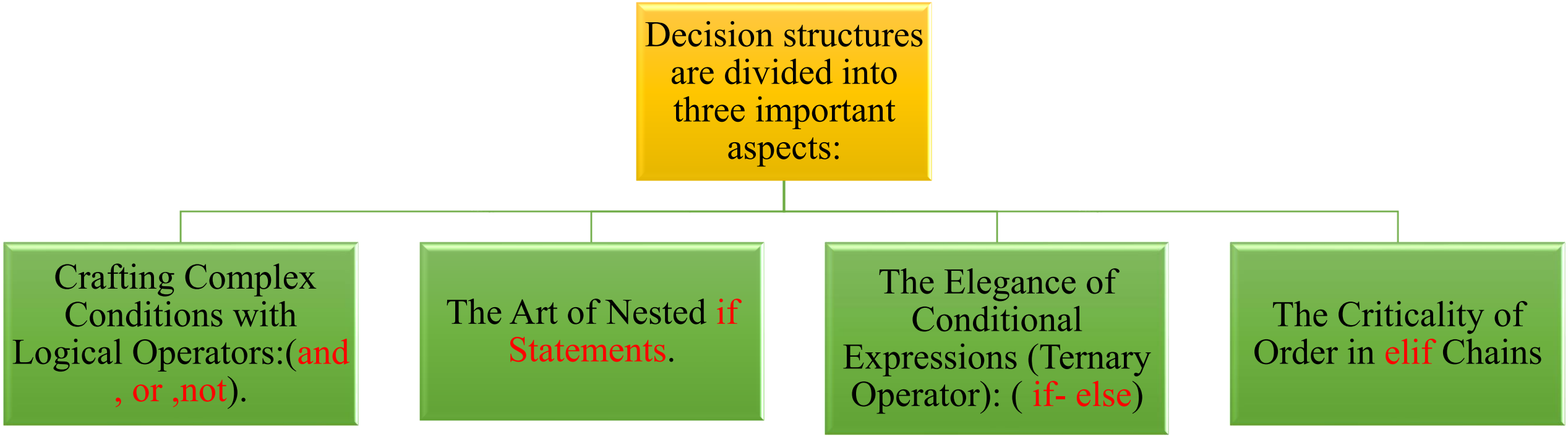
2. Determine whether the user is old or not

```
age = 20                    # يمكنك تغيير هذا العمر

if age >= 18:
    print(" أنت بالغ، يمكنك التصويت ")
else:
    print("أنت قاصر، لا يمكنك التصويت بعد" )
```

Decision Structure

decision structures in Python allow your code to make choices and execute different blocks of instructions based on a variety of conditions. They are the very essence of creating dynamic and responsive programs.



Crafting Complex Conditions with Logical Operators

Python empowers you to combine multiple conditions using logical operators:

1. **AND:** This operator returns True only if all the conditions it connects are True. Imagine you have two conditions: the first one the person is over 18 years old and the second he has a driver's license. In order for him to be allowed to drive, both conditions must be met:

```
age = 28
has_id = True
if age >= 18 and has_id:
        print("Access granted.")
else:
        print("Access denied.")
```

2. OR: This operator returns True if at least one of the conditions it connects is True. Think about offering a discount if a customer is a student or if they have a loyalty card:

```
student = False
loyalty = True
if student or loyalty:
    print( "Discount applied. " )
else:
    print( "No discount available. " )
```

Imagine you are offering a discount on a product if the product price is greater than $100 or if the customer has a discount coupon:

```
price = 120
has_coupon = False
if price > 100 or has_coupon:
    print("تم تطبيق الخصم")
else:
    print("لم يتم تطبيق الخصم")
```

3. NOT: This operator reverses the truth value of a condition. If a condition is True, not makes it False, and vice versa. For example, checking if a list is not empty:

```
my_list = [4, 5, 6]
if not my_list == 0:
    print("The list is not empty.")
                                    # A more Pythonic way:
if my_list:                         # Non-empty lists are considered True in a boolean context
    print("The list is not empty.")
```

Boolean Expressions and Relational Operators

The expressions that are tested by the if statement is called Boolean expressions.

A relational operator determines whether a specific relationship exists between two values. For example, the greater than operator (>) determines whether value is greater than another. The equal to operator (==) determines whether two values are equal. The Table below lists the relational operators that are available in Python.one

Boolean Expressions and Relational Operators

Operator	Meaning
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
==	Equal to
!=	Not equal to

Boolean Expressions and Relational Operators

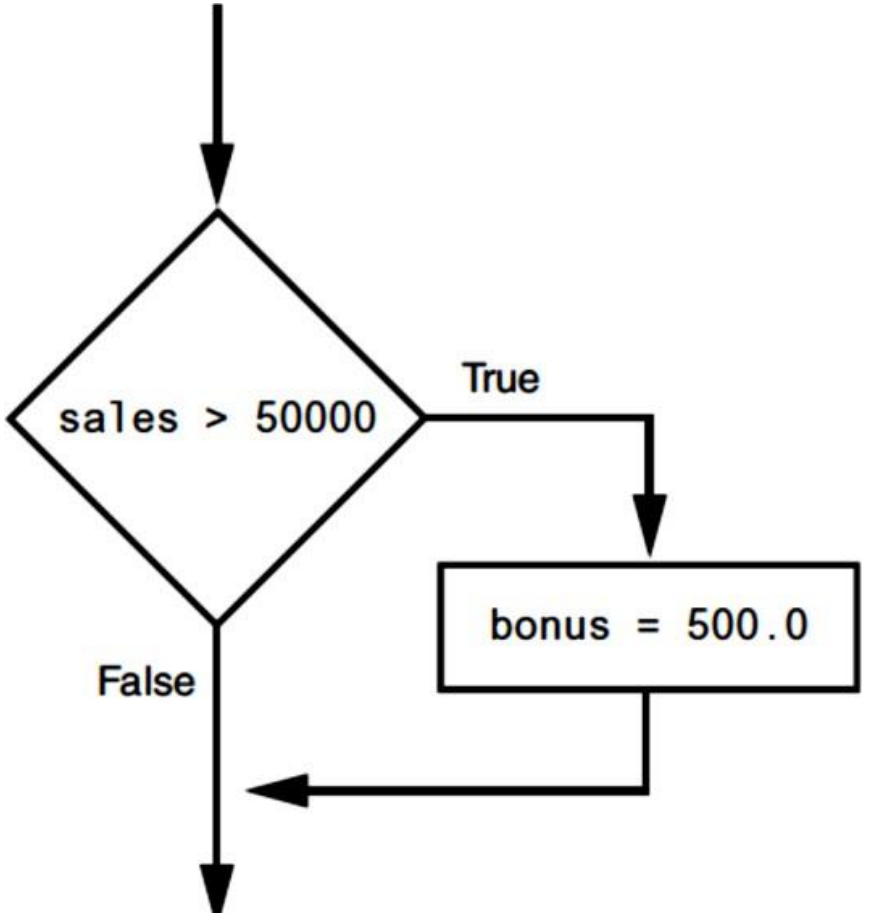
Table below shows examples of several Boolean expressions that compare the variables x and y.

Expression	Meaning
`x > y`	Is x greater than y?
`x < y`	Is x less than y?
`x >= y`	Is x greater than or equal to y?
`x <= y`	Is x less than or equal to y?
`x == y`	Is x equal to y?
`x != y`	Is x not equal to y?

Boolean Expressions and Relational Operators

Let's look at the following example of the if statement:
if sales > 50000: bonus = 500.0
This statement uses the > operator to determine whether sales is greater than 50,000. If the expression sales > 50000 is true, the variable bonus is assigned 500.0.

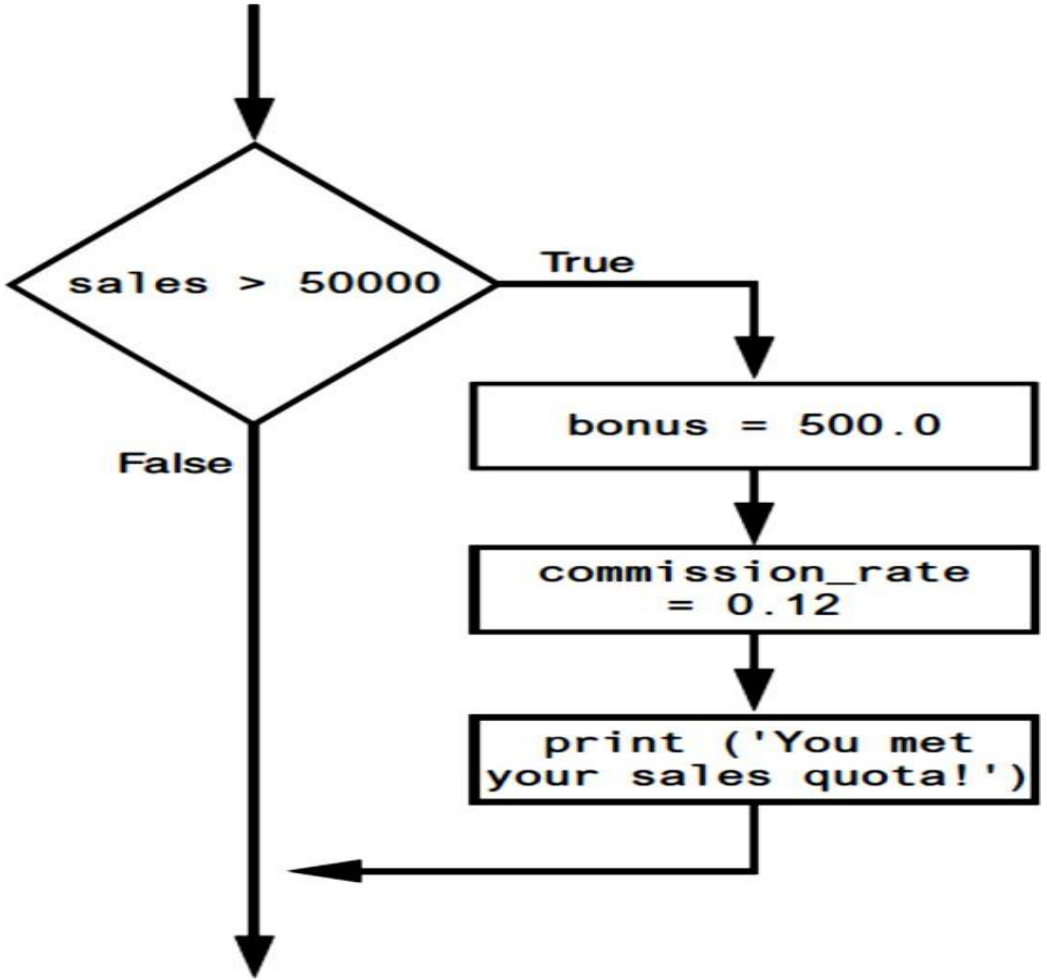
If the expression is false, however, the assignment statement is skipped. Figure shows a flowchart for this section of code.



Boolean Expressions and Relational Operators

The following example conditionally executes a block containing three statements. Figure below shows a flowchart for this section of code:

```
if sales > 50000:
        bonus = 500.0
        commission_rate = 0.12
print('You met your sales quota!')
```



Thank’s For Listening

الاسبوع السادس
Nested Decision Structures &Boolean Variables

الهدف العام :

تمكين من فهم وتطبيق هياكل اتخاذ القرار المتداخلة والمتغيرات المنطقية في بايثون من خلال، فهم هياكل القرار المتداخلة، استخدام عبارات if _elif_else وفهم المتغيرات المنطقية (Boolean Variables)

مدة المحاضرة: ساعتان (نضري +عملي)

م	الجلسة الاولى	استراحة	الجلسة الثانية
مواضيعها	Nested decision structures	10 دقيقة	Nested if-elif Statement
	Nested if Statement		Boolean Variables
	The if-else Statement		Using Boolean Variables
	Examples		مناقشة + Quiz
زمن الجلسة	50 دقيقة		50 دقيقة

Nested Decision Structures

Nested decision structures involve placing if statements inside other if or else clauses. This technique allows for the sequential evaluation of multiple conditions, where the inner condition is checked only if the outer condition is true. Nested structures can include any combination of if, if...else, and elif statements within each other, enabling the construction of complex logic for decision-making based on several criteria.

Nested if statements in Python allow you to have an if statement within another if statement. This is useful for scenarios where you need to evaluate multiple conditions in a hierarchical manner.

Nested if Statement

Example 1

Imagine a program that determines a ticket price based on a person's age and whether they are a member or not.

```
age = 30
member = True
if age > 18:
    if member:
        print("The ticket price is $12.")
    else:
        print("The ticket price is $20.")
else:
    if member:
        print("The ticket price is $8.")
    else:
        print("The ticket price is $10.")
```

The if-elif-else Statement

The if-elif-else statement in Python is a conditional control structure used for making multi-way decisions . This statement allows for checking multiple conditions in sequence and executing a specific block of code when one of the conditions is true .

The execution of an if-elif-else statement begins by evaluating the condition in the if clause. If this condition is true, the corresponding block of code is executed, and the rest of the elif and else conditions are skipped . If the first condition is false, the condition in the first elif clause is evaluated. If this condition is true, its corresponding block of code is executed, and the remaining conditions are skipped . There can be any number of elif clauses, and they are evaluated in order until a true condition is found . If none of the conditions in the if or elif clauses are true, the block of code under the optional else clause is executed . If the else clause is not included and none of the conditions are true, then none of the conditional code blocks are executed, and the program continues execution from the line following the if-elif-else structure .

Example2 The following example illustrates how the if-elif-else statement works

```
number = -5
if number > 0:
    print("The number is positive.")
elif number < 0:
    print("The number is negative.")
else:
    print("The number is zero.")
print("This statement will always be executed.")
```

Example3

This Python code example checks the user's age and prints a different message based on the entered age.

```
age = int(input( “Enter your age: ” ))
if age >= 18:
   print(“you are now signed up!")
elif age < 0:
      print(“You haven’t been born yet!")
elif age >= 100 :
      print(“You are too old to sign up")
else:
      print(“You must be 18+ to sign up")
```

Example4

This Python code example checks the user's age and prints a different message based on the entered age.

```
age = int(input( “Enter your age: ” ))
if age >= 100 :
      print(“You are too old to sign up")
elif age >= 18:
   print(“you are now signed up!")
elif age < 0:
      print(“You haven’t been born yet!")

else:
      print(“You must be 18+ to sign up")
```

Nested if-elif Statement

Example5

Another example illustrates the use of elif within a nested structure :

```
num = 18
if num < 0:
    print("The number is negative.")
elif num > 0:
    if num <= 10:
        print("The number is between 1 and 10.")
    elif num > 10 and num <= 20:
        print("The number is between 11 and 20.")
    else:
        print("The number is greater than 20.")
else:
    print("The number is zero.")
```

Boolean Variables

- Boolean variable: references one of two values, `True` or `False`
  - Represented by `bool` data type
- Commonly used as flags
  - Flag: variable that signals when some condition exists in a program
    - Flag set to `False` → condition does not exist
    - Flag set to `True` → condition exists

Using Boolean Variables in if, elif, and else Statements

Boolean variables can be used directly as conditions in if, elif, and else statements . Since these variables already hold a boolean value (True or False), there is no need for an explicit comparison .

For example, instead of writing if is_raining == True:, you can simply write if is_raining: . Similarly, instead of if is_admin == False:, you can write if not is_admin: . This makes the code more concise and readable.1

Example 6

The following example demonstrates the use of Boolean variables in conditional statements:

```
user_logged_in = True
is_admin = False

if user_logged_in:
    print("User is logged in.")
    if is_admin:
        print("User is an administrator.")
    else:
        print("User is not an administrator.")
else:
    print("Please log in.")
```

Example 7

Example of reducing nesting using elif

```
temperature = 25
humidity = 70

if temperature > 30:
    if humidity > 60:
        print("It's hot and humid.")
    else:
        print("It's hot and dry.")
elif temperature > 20:
    if humidity > 60:
        print("It's moderate and humid.")
    else:
        print("It's moderate and dry.")
else:
    print("It's cold.")
```

Example 8 This code can be rewritten using elif to reduce nesting:

```
temperature = 25
humidity = 70

if temperature > 30 and humidity > 60:
    print("It's hot and humid.")
elif temperature > 30 and humidity <= 60:
    print("It's hot and dry.")
elif temperature > 20 and humidity > 60:
    print("It's moderate and humid.")
elif temperature > 20 and humidity <= 60:
    print("It's moderate and dry.")
else:
    print("It's cold.")
```

Thank ’s For Listening

الاسبوع السابع – الاسبوع الثامن
Repetition Structures

الهدف العام :

تعريف الطلاب بمفهوم هياكل التكرار (Repetition Structures) في البرمجة، شرح مزايا استخدام هياكل التكرار، التركيز على حلقة while كأداة للتحكم في التكرار بناء على شرط معين، توضيح كيفية استخدام جملة else مع حلقة while، شرح كيفية إنشاء حلقات لا نهائية، التركيز على حلقة for كأداة للتكرار على تسلسل من العناصر، شرح استخدام break , countinue داخل حلقه for. شرح وظيفة ودور دالة range() وتلخيص متى يجب استخدام كل من حلقة while , for , range().

مدة المحاضرة: ساعتان (نضري +عملي)

م	الأسبوع السابع	استراحة لكل جلسة	الأسبوع الثامن
مواضيعها	Introduction to Repetition Structures	10 دقيقة	For Loop
	Examples		Using the break and continue statements inside a for loop
	While Loop		Using the range Function with the for Loop
	Examples		مناقشة Quiz +
زمن الجلسة	2 ساعة		2 ساعة

Introduction to Repetition Structures

Let's begin by defining what a repetition structure is. Simply put, it's a programming construct that allows us to execute a specific block of instructions repeatedly until a certain condition is met or a defined number of iterations is completed. Imagine you want to print the numbers from 1 to 10. Instead of writing the print statement ten times, you can use a loop to automate this task.

Repetition structures offer us several significant advantages, including :
Reduced Code Size: Avoiding the need to write the same instructions over and over
Increased Program Efficiency: Facilitating the organized execution of repetitive tasks.
Improved Code Readability and Maintainability: Making the code clearer and more logical through the use of loops.

The while Loop: A Condition-Controlled Loop

The while loop is a powerful tool for executing a block of statements as long as a specified condition remains true. The loop begins by evaluating the condition. If the condition is true, the block of statements inside the loop is executed. After execution, the condition is evaluated again, and this process continues until the condition becomes false, at which point the loop terminates, and the program proceeds to the following loop

General Syntax of a while Loop while condition:

```
while condition:
# Block of statements to be executed as long as the condition is true
  statement1
  statement2
  # ...
```

```
while condition:
    statements
```

Example:

Let's say we want to write a program that prints the numbers from 1 to 5 using a while loop. We can achieve this as follows:

```
counter = 1
while counter <=5:
    print(counter)
    counter += 1
print("Loop finished")
```

```
i = 1
#  في كل دورة من دورات الحلقة سيتم فحص قيمة العداد iو إذا
#كانت أصغر أو تساوي 5 إضافة و من ثم سيتم طباعتها 1عليها
while i <= 5:
    print(i)
    i += 1
```

Validating User Input: While loops are often used to repeatedly prompt the user to enter data until valid input is provided.

```
password = " "
while password!= “haider":
    password = input("Enter your password: ")
    if password!= “haider":
        print("Incorrect password. Try again.")
print("Password accepted!")
```

Example of using the else statement with the while loop in Python

The else statement can come after the while loop. It runs a code section when the while loop's condition is False.

```
i = 1                  # في كل دوره من الدورات سيتم فحص قيمة العداد أي واذا كانت اصغر او تساوي ٥ سيتم
ومن ثم اضافه رقم ١ عليها. واذا لم يتحقق الشرط سوف يتم تنفيذ امر الطباعه الموجود داخل #else. طباعتها
while i < 5:
    print(i)
    i += 1
else:
    print('This block is executed when the condition return False!')
```

How to create an infinite loop

In the following example, we created a loop that continues executing the code placed in it without stopping.

```
while 1 == 1:
    print('I am stuck!’)
```

Output:
I am stuck!
I am stuck!
I am stuck!
I am stuck!
.
.
.

How to create an infinite loop

In this example, we wrote while True: instead of writing while 1 == 1:and this will also make the loop continue executing the code placed within it indefinitely.

```
while True:
    print( 'I am stuck!' )
```

Output:

```
I am stuck!
I am stuck!
I am stuck!
I am stuck!
.
.
.
```

Important Notes Regarding the while Loop:

- Variables used in the condition must be initialized before entering the loop .
- It's essential to ensure that the condition will eventually become false within the loop (or through an external process affecting the condition) to avoid infinite loops.

The for Loop: A Count-Controlled Loop

The for loop differs from the while loop in how it controls the iteration. The for loop is primarily used to iterate over a sequence of items (such as a list, a string, or a range of numbers) and execute a block of statements for each item in that sequence.

General Syntax of a for Loop:

for variable in sequence:

```
    # Block of statements to be executed for each item in the sequence
    statement1
    statement2
    # ...
```

```
for element in sequence:
    statements
```

In this syntax, variable is a variable that takes on the value of each item in the sequence during each iteration of the loop. sequence is the collection of items we want to iterate over.

Example:

Let's assume we have a list of fruits and we want to print each fruit individually

```
fruits = ["apple", "banana", "orange"]
for fruit in fruits:
    print(fruit)
print("All fruits displayed")
```

In this example, the loop iterates three times, and in each iteration, the variable fruit takes the value of one of the elements in the fruits list.

Practical examples:

1. Repetition on a list of numbers:

```
numbers = [1, 2, 3, 4, 5]
for num in numbers:
    print(num * 2)
```

2. Iteration on characters of a string:

```
message = "Hello"
for char in message:
    print(char.upper())
#  print(char)
```

3. use a for loop over a collection

```
days = ["Mon","Tue","Wed","Thu","Fri","Sat","Sun"]
for d in days:
        print (d)
```

Practical examples:

3. use a for loop over a collection

```
colors = ["Red","Blue","Green","Yellow"]
for x in colors:
    if x == "Blue":
        print(f"My favorite color is:{x}")
    else:
        print (x)
```

Using the break and continue statements inside a for loop:

The break and continue statements work similarly inside a for loop as they do inside a while loop. The break statement terminates the loop early, and the continue statement skips the current iteration and moves to the next.

Practical examples:

1. break:

```
for number in range(10):
    if number == 5:
        break
    print(number)
```

2. Continue:

```
for number in range(5):
    if number == 2:
        continue
    print(number)
```

Using the range Function with the for Loop

The range() function is an incredibly useful tool when using a for loop to execute a specific number of iterations. The range() function creates a sequence of numbers. It can be used in three main ways:

- ✓ **range(stop):** Creates a sequence of numbers starting from 0 and ending at stop - 1.
- ✓ **range(start, stop):** Creates a sequence of numbers starting from start and ending at stop - 1.
- ✓ **range(start, stop, step):** Creates a sequence of numbers starting from start and ending at stop - 1, with the increment between each number specified by step.

Examples of Using range() with a for Loop :

Printing numbers from 0 to 4:

```
for i in range(5):                    # Generates numbers 0, 1, 2, 3, 4
    print(f"The number is: {i}")
```

Printing numbers from 2 to 7:

```
for i in range(2, 7):                 # Generates numbers 2, 3, 4, 5, 6
    print(f"The number is: {i}")
```

Print even numbers from 0 to 10:

```
for i in range(0, 10, 2):             # Generates numbers 0, 2, 4, 6, 8
    print(f"The even number is: {i}")
```

In these examples, the range() function generates a sequence of numbers, and in each iteration of the for loop, the variable (i, j, k) takes the next value in that sequence.

Examples of Using range() with a for Loop :

Print Countdown from 5 to 1:

```
for l in range(5, 0, -1):
    print(l)
```

```
# for هنا قمنا بإنشاء سلسلة من الأعداد الموجودة من ١ إلى ٥ و مررنا عليها بواسطة الحلقة
 في كل دورة في الحلقة سيتم جلب عدد من هذه السلسلة و تخزينه في المتغير  #n و من ثم طباعته
for n in range(1, 6):
    print(n)

#  بعد أن توقفت الحلقة قمنا بعرض قيمة المتغير  nالذي تم تعريفه أساساً فيها
print('n contains:', n )
```

Using the else statement with a For Loop

Python supports an else statement associated with a for loop. If an else statement is used with a for loop, the else statement will be executed when the loop exhausts the iteration list. Note the following example:

```
for x in range(5,10):
        print(x)
else:
        print('x is not in the range from 5 to 10')
```

SUMMARY

- while loop: Use it when you need to repeat a block of code as long as a certain condition is true. Remember to ensure the condition will eventually become false to avoid infinite loops.
- for loop: Use it when you need to iterate over a sequence of items (like lists, strings, or the output of range()) and perform an action for each item.
- range() function: Use it within for loops to generate sequences of numbers, allowing you to control the number of iterations.

الاسبوع التاسع – الاسبوع العاشر
Functions in Python

الهدف العام :

تعريف الطلاب بالدوال في بايثون ودورها في تنظيم الكود، شرح كيفية إنشاء الدوال، توضيح أهمية المسافة البادئة (indentation) في بناء الدوال والتمييز بين الدوال التي لا تُرجع قيمة والدوال التي تُرجع قيمة.
تقديم مفهوم المتغيرات المحلية (local variables)ومجالها داخل الدالة.
تقديم مفهوم المتغيرات العامة (global variables) إمكانية الوصول إليها من داخل وخارج الدالة وشرح كيفية استدعاء الدوال لتنفيذها.

مدة المحاضرة: ساعتان (نضري +عملي)

م	الأسبوع التاسع	استراحة لكل جلسة	الأسبوع العاشر
مواضيعها	Introduction to Functions	10 دقيقة	Local Variables
	Void and Value-Returning Functions		Global variables and global constants
	Indentation in Python		Examples
	Examples		مناقشة Quiz +
زمن الجلسة	2 ساعة		2 ساعة

Introduction to Functions

A function is a block of code that performs a specific task.

Suppose we need to create a program to make a circle and color it. We can create two functions to solve this problem:

1. function to create a circle.
2. function to color the shape.

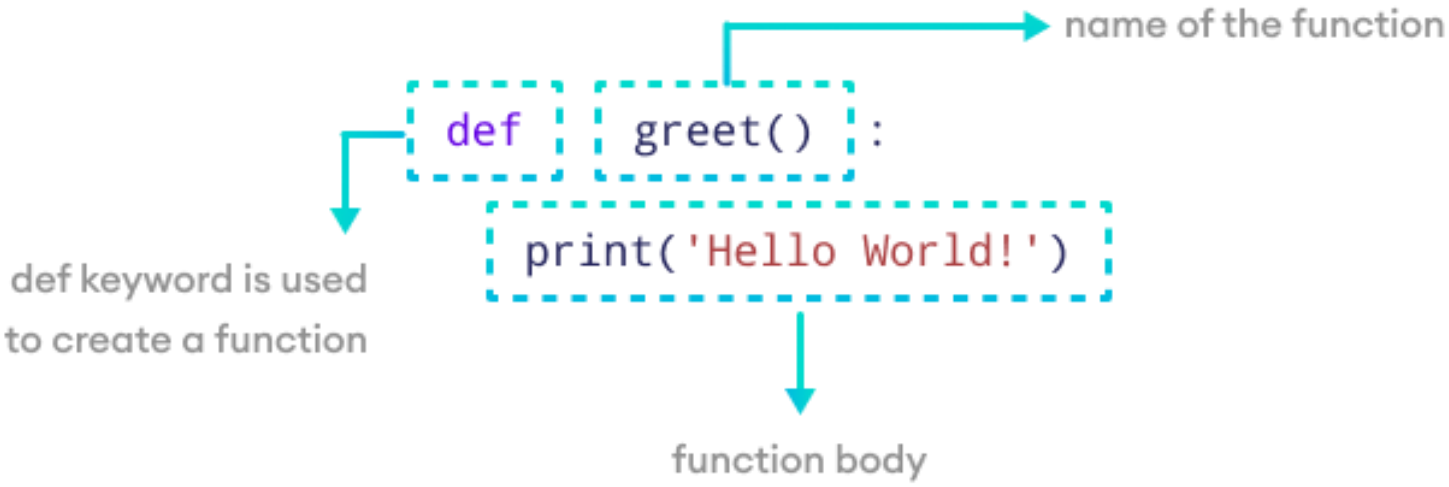
Dividing a complex problem into smaller chunks makes our program easy to understand and reuse.

Create a Function

Let's create our first function.

```
def greet():
        print('Hello World!')
```

Here, we have created a simple function named greet() that prints **Hello World!**



Note: When writing a function, pay attention to indentation, which are the spaces at the start of a code line.

In the above code, the print() statement is indented to show it's part of the function body, distinguishing the function's definition from its body.

In the following example, we defined a function called my_function
with only one print command. Then, we called it.

```
# هنا قمنا بتعريف دالة إسمها my_function
def my_function():
    print('My first function is called')


# هنا قمنا باستدعاء الدالة my_functionحتى يتنفذ الأمر الموضوع فيها #
my_function()
```

Calling a Function

Let's create our first function.

In the above example, we have declared a function named greet().

```
def greet():
        print('Hello World!')
```

If we run the above code, we won't get an output.

It's because creating a function doesn't mean we are executing the code inside it. It means the code is there for us to use if we want to.

To use this function, we need to **call the function**.

Function Call

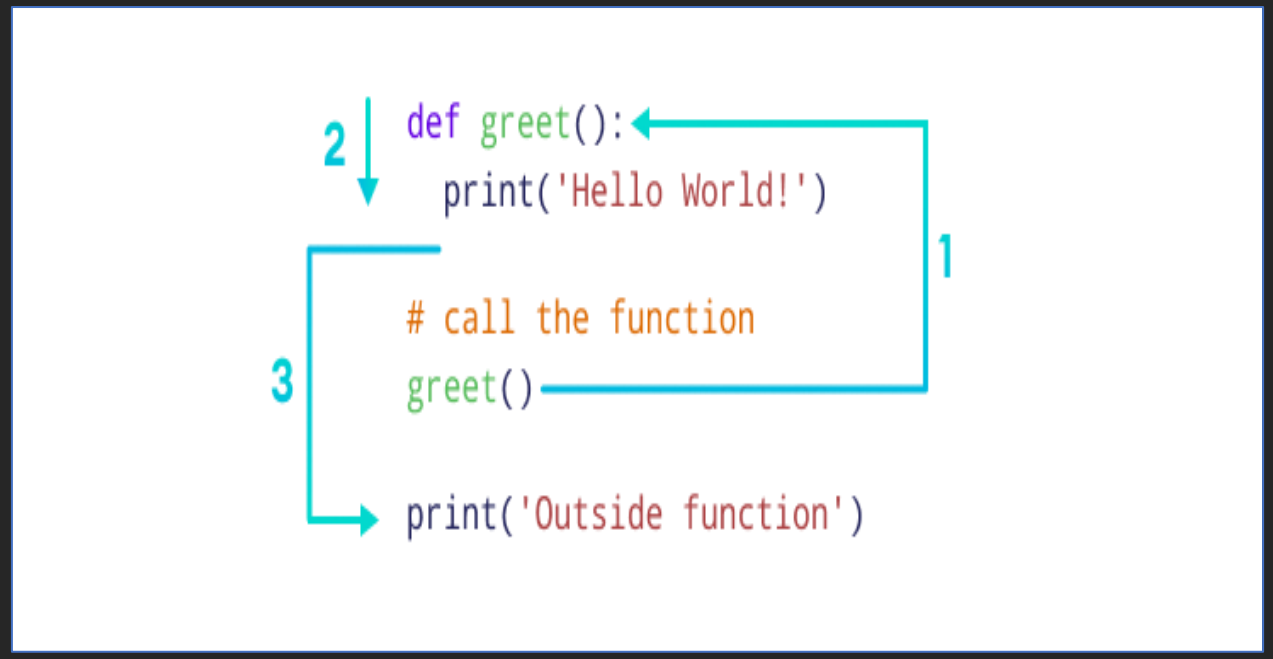
```
greet()
```

Example: Python Function Call

```
def greet():
        print('Hello World!’)
 # call the function
greet()
print('Outside function’)
```

Output

```
Hello World!
Outside function
```



Working of Python Function

Example**: print document inside function and calling the function

```
def hello():
    """This function greeting the user"""
    print("hi ali")


print(hello.__doc__)
hello()
```

Void Functions and Value-Returning Functions in Python

In Python, functions are defined using the def keyword, followed by the function's name, a set of parentheses that may contain parameters, and a colon. The subsequent indented block of code constitutes the function's body, containing the statements to be executed.

These functions are called by using their name followed by parentheses( ), potentially including arguments that correspond to the defined parameters.

It's important to realize that in Python, every function returns a value, regardless of its apparent behavior. This return can be explicit, specified through a return statement, or implicit.

This consistent return mechanism distinguishes Python from some other programming languages where the absence of a return type, often denoted by keywords like "void," signifies a function that does not return any value.

Void Functions and Value-Returning Functions in Python

In Python, the term "void function" is commonly used to describe a function whose primary objective is to execute a series of actions or produce side effects rather than to explicitly compute and return a value for immediate use. These functions might include a return statement, but it will either be used without any accompanying value or omitted entirely. A defining characteristic of these functions in Python is that even when a return statement is absent, or when it appears without an expression, the function will implicitly return a special value known as None. This behavior underscores the principle that all Python functions yield a return value, even if it signifies the absence of a specific result. The None keyword in Python represents this absence of a value or a null value. It is a built-in constant and a singleton object of the type (NoneType). Understanding None is therefore essential to comprehending the behavior of functions that operate like void functions in Python.

A simple function designed to print a message:

```
def print_message(message):
    print(message)
```

Output:

Message

None.

This demonstrates that even though the print_message function doesn't explicitly return anything, it implicitly returns None. Another example involves a function that uses a return statement without a value to control the flow of execution:

```
def check_value(number):
    if number < 0:
        print("Number is negative")
        return
    print("Number is non-negative")
???????
```

If check_value(-5) is called, it will print "Number is negative" and then exit due to the return statement. If we were to call check_value(5) and then inspect its return value, we would find it to be None. Check_value(number or -5)

A simple function designed to print a message:

In the following example, we defined a function called get_sum. When we call it, we pass it two numbers and it returns the sum of them.

```
#هنا قمنا بتعريف دالة إسمها get_sum عند إستدعائها نمرر لها عددين فتقوم بإرجاع ناتج جمعهما
def get_sum(a, b):
    """This method returns the sum of the numbers that you passed in a and b"""
    return a + b;


#هنا قمنا بتخزين ناتج جمع العددين ٣ و ٥ الذي سترجعه الدالة get_sum() في المتغير X
x = get_sum(3, 5)

#  هنا قمنا بعرض قيمة المتغير x و التي تساوي ٨
print(x)
```

#The default value that is set for a variable is called Default Value or Default Argument.

In the following example, we define a function named print_language.
This function has a single variable named language and has the text 'English' as its default value.

```
هنا قمنا بتعريف دالة اسمها print_languageعند استدعائها يمكنك تمرير قيمة#
لها مكان البارامتر language ويمكنك عدم تمرير قيمة لأنه أصلاً يملك قيمة #
def print_language(language='English'):
    print('Your language is:', language)


هنا قمنا باستدعاء الدالة ()print_languageبدون تمرير قيمة #
قيمة مكان المتغيرات languageو بالتالي ستظل قيمته  'English# '
print_language()


هنا قمنا باستدعاء الدالة ()print_languageمع تمرير القيمة #
'Arabic’للمتغيرات ‘ language‘و بالتالي ستصبح قيمته  'Arabic#‘
print_language('Arabic')
```

Indentation in Python

Indentation: More Than Just Formatting:

It's important to note that indentation in Python isn't just a means of improving readability; it's an essential part of the language's syntax. Indentation is used to delineate blocks of code, such as function bodies, loops, and conditional statements.

How Python Uses Indentation to Delineate Code Blocks:

This can be illustrated with examples of how the indentation level determines which instructions belong in a particular block of code. Example:

Example

```
def my_function(x):
    if x > 5:
        print("x is greater than 5")                    # Part of the if block
        y = x * 2                                       # Still part of the if block
    else:
        print("x is 5 or less")                          # Part of the else block
    print("This line is outside the if/else block")     # Not part of the if/else block

my_function(10)
```

Practical Examples Showing Correct and Incorrect Indentation:

Provide several examples of code with correct indentation and equivalent examples with incorrect indentation, showing the resulting errors or unexpected behavior. Example (Correct):

```
def print_numbers(n):
   for i in range(n):
      print(i)
```

Example

Example (Incorrect - IndentationError):

```
def print_numbers(n):
for i in range(n):
print(i)                                    # Error: expected an indented block
```

Example (Incorrect - Logical Error):

```
def print_numbers(n):
    for i in range(n):
    print(i)
print("Loop finished")  # Intended to be outside the loop, but might be incorrectly indented
```

Example: A function that takes a list of names and prints a personalized greeting for each name.

```
def greet_all(names):
    for name in names:
        greet(name)                    # Calling the void function 'greet' defined earlier

def greet(person_name):
    if person_name:
        print(f"Hello, {person_name}!")
    else:
        print("Greeting to nobody!")
student_names =["haider","krar","noor","muhammed"]
greet_all(student_names)
```

#الداله الأولى تأخذ قائمة من الأسماء وتقوم باستدعاء الدالة الثانية لكل اسم في تلك القائمة.

Python Local Variables

When we declare variables inside a function, these variables will have a local scope (within the function). We cannot access them outside the function.

These types of variables are called local variables. For example,

```
def greet():
                                    # local variable ملاحضة فقط
    message = 'Hello’
    print('Local', message)

greet()
                                    # try to access message variable
                                    # outside greet() function
print(message)
```

Output:

```
Local Hello
NameError: name 'message' is not defined
```

Here, the message variable is local to the greet() function, so it can only be accessed within the function.

Global variables and global constants

In Python, a variable declared outside of the function or in global scope is known as a global variable. This means that a global variable can be accessed inside or outside of the function.

Let's see an example of how a global variable is created in Python.

```
# declare global variable
message = 'Hello'
def greet():
                        # declare local variable
    print('Local', message)

greet()
print('Global', message)
```

```
Local Hello
Global Hello
```

```
Local Hello
Global Hello
```

This time we can access the massage variable from outside of the greet() function. This is because we have created the massage variable as the global variable.

Summary

There are two main types of functions in Python: void functions and functions that return a value. Empty functions perform actions or have side effects and do not return an explicit value, but rather return None implicitly.

Functions that return a value perform arithmetic or logical operations and return a specific result using the return statement. The characteristics and uses of each type are explained with practical examples, and the key differences between them are highlighted in a brief table.

الاسبوع الحادي عشر – الأسبوع الثاني عشر
Value-Returning Functions

الهدف العام :

تعريف الطلاب بالدوال التي تُرجع قيمة، واستخدام وحدات المكتبة القياسية من خلال عبارة الاستيراد، والقدرة على توليد أرقام عشوائية باستخدام وحدة random، والاستفادة من الدوال الرياضية في وحدة math، كلها أدوات أساسية في بايثون.

مدة المحاضرة: ساعتان (نضري +عملي)

م	الأسبوع الحادي عشر	استراحة لكل جلسة	الأسبوع الثاني عشر
مواضيعها	Introduction to Value-Returning Functions	10 دقيقة	Generating Random Numbers
	Standard Library Functions and the import Statement		Writing Your Own Value-Returning Functions
	The math Module		Returning Strings and Boolean Values
	Examples		Returning Multiple Values
زمن الجلسة	2 ساعة		2 ساعة

Introduction to Value-Returning Functions

Definition of functions: Programming blocks that perform a specific task and return a value.

The difference between void functions and return-value functions:

- Void functions: Perform a task without returning a value (e.g., print()).
- Return-value functions: Return a value that can be stored or used (e.g., len(), input()).

Ex :

```
result = len("Python")                # 6 يُرجع
age = int(input("أدخل عمرك:"))        #يُرجع قيمة رقمية
print(result)
print(age)
```

Ex1 : Example of a function to add two numbers

```
#                                   داله لجمع رقمين
def add(x, y):
    return x + y
sum_numbers = add(10, 5)
print(sum_numbers)          #15
```

Ex2:Example of a function to check if a number is positive

```
def is_positive(number):
    if number > 0:
        return True
    else:
        return False
result = is_positive(-3)
print(result)
```

Ex1 : Example of a function to add two numbers or calculate a square

```
#                                    داله لجمع رقمين او لحساب مربع
def calculate(num1, num2=None):
 """تقوم هذه الدالة إما بجمع عددين أو حساب مربع عدد واحد."""
     if num2 is not None:
        return num1 + num2
     else:
        return num1 ** 2
                                                  أمثلة على استخدام الدالة #
sum_result = calculate(5, 3)
print( f” The sum of the numbers 5 and 3 is:{sum_result}”)
square_result = calculate(7)
print(f” The square of the number 7 is:{square_result}”)
```

EX: Write a Python function called calculate_square that takes an integer as input and returns the square of that number.

```
def calculate_square(number):
    """تأخذ هذه الدالة عددًا صحيحًا كمدخل وتُرجع مربعه."""
    square = number ** 2
    return square


#مثال للاستخدام
input_number = 5
result = calculate_square(input_number)
print(f"مربع العدد{input_number} هو:{result}")
```

Standard Library Functions and the import Statement

Standard Library: A set of functions built into Python (such as math and random).

- How to import functions:

- Import the entire library:

Ex: Generates a random number between 1 and 10

```
import random

print(random.randint(1, 10))
```

To further expand our capabilities, we learned how to import entire modules that contain a collection of useful functions. For instance, to use advanced mathematical functions, we can import the math module.

The math Module

Python math module is a built-in module that provides various functions and constants to perform the mathematical operations more accurately.

Additional Code Example Explanation from the math module:

In these examples, we explored some other functions available in the math module such as math.radians() for converting angles from degrees to radians, math.cos() for calculating the cosine, math.log() for calculating the natural logarithm, math.exp() for calculating the exponential function, and the mathematical constant math.pi.

```
import math
angle_in_degrees = 30
angle_in_radians = math.radians(angle_in_degrees)
sin_value = math.sin(angle_in_radians)
cos_value = math.cos(angle_in_radians)
tan_value = math.tan(angle_in_radians)
print("For angle", angle_in_degrees, "degrees:")
print("Sin:", sin_value)
 print("Cos:", cos_value
print("Tan:", tan_value)
```

```
print("The value of π (pi) is:", math.pi)
```

Import a specific function: (math)

```
import math
number = 16
square_root = math.sqrt(number)
print(f"The square root of {number} is: {square_root}")
power = math.pow(2, 3)
print(f"2 raised to the power of 3 is: {power}")
#٢٥ لحساب الجذر التربيعي للعدد
result_sqrt = math.sqrt(25)
Print(f” squer the number 25 is:{result_sqrt}”)
```

Generating Random Numbers

In many applications, we need to introduce an element of randomness.

The random module in Python provides us with powerful tools to generate random numbers of various types.

- The most important functions in the random library:

randint(a, b): Returns an integer between a and b.

random(): Returns a random number between 0 and 1.

choice(list): Returns a random element from a list.

Comprehensive Example:

Example:

```
import random

                # Generate a random integer between 1 and 10 (inclusive)
random_integer = random.randint(1, 10)
print(f"عدد صحيح عشوائي بين ١ و ١٠: {random_integer}")
                # Generate a random decimal number between 0.0 and 1.0 (excluding 1.0)
random_float = random.random()
print(f "Random float between 0.0 and 1.0: {random_float}")
                # Choose a random item from a list
my_list = ["apple", "banana", "orange"]
random_item = random.choice(my_list)
print(f "Random item from list: {random_item}“)
```

Writing Your Own Value-Returning Functions

We've learned how to use built-in functions, but the real power of programming lies in our ability to write our own functions to perform specific tasks and return results.

Let's write a simple function that calculates the square of a given number:

```
def calculate_square(number):
    """This function calculates the square of the input number."""
    square = number ** 2
    return square

result = calculate_square(5)
print(f"The square of 5 is: {result}")
```

Returning Strings and Boolean Values

The values that functions can return are not limited to numbers. Functions can also return strings and Boolean values (True or False).

EX:

Let's write a function that checks if a given number is even or odd and returns a Boolean value:

```
def is_even(number):
    """This function determines if a number is even and returns True or False."""
    if number % 2 == 0:
        return True
    else:
        return False


print(f"Is 10 even? {is_even(10)}")
print(f"Is 7 even? {is_even(7)}")
```

Let's write another function that creates a greeting message based on a user's name:

```
def create_greeting(name):
    """This function creates a greeting message with the user's name."""
    greeting = f"Hello, {name}!"
    return greeting

message = create_greeting("Ali")
print(message)
```

What is the expected output when running this code?

```
def create_custom_message(username, event_type):
        return f "A new event has been logged for user {username}: {event_type}."

message = create_custom_message("Khaled", "Login")
print(message) # Output: A new event has been logged for user Khaled: Logged in.
```

Returning Multiple Values

Python offers great flexibility as a function can return more than one value simultaneously. These values are typically grouped together in the form of a "tuple".

Code Example Explanation:

Let's write a function that calculates the area and perimeter of a rectangle and returns both values:

```
def calculate_rectangle_properties(length, width):
    """This function calculates the area and perimeter of a rectangle and returns them."""
    area = length * width
    perimeter = 2 * (length + width)
    return area, perimeter


rectangle_area, rectangle_perimeter = calculate_rectangle_properties(5, 10)
print(f"Rectangle area: {rectangle_area}")
print(f"Rectangle perimeter: {rectangle_perimeter}")
```

Example:

A function that returns a comma-separated value (name and age) in a return statement.

```
def get_name_and_age():
  name = ""علي
  age = 30
  return name, age

person_info = get_name_and_age()
print(person_info)                    # سيتم طباعة: ('علي'، ٣٠)
print(type(person_info))              # سيتم طباعة: <class 'tuple'>
```

Example:

```
def data_analysis(data_list):
    min_val = min(data_list)
    max_val = max(data_list)
    average = sum(data_list) / len(data_list)
    return min_val, max_val, average

my_data = [1, 5, 2, 8, 3, 4]
min_result, max_result, average_result = data_analysis(my_data)
print(f"Min: {min_result}, Max: {max_result}, Average: {average_result}")
```

Summary

Functions that return a value, using standard library modules through the import statement, the ability to generate random numbers using the random module, and leveraging mathematical functions in the math module are essential tools in Python. Understanding these concepts enables students to write more efficient and organized programs and opens up vast opportunities for developing diverse applications. Through continued practice, writing their own functions, and exploring standard library modules, students will significantly enhance their programming skills in Python.

Reference

1. https://www.py4e.com/lessons/intro#
2. https://harmash.com/tutorials/python/functions
3. https://www.programiz.com/python-programming/function
4. https://www.w3schools.com/
5. https://www.youtube.com/watch?v=_XbYzHS5dg4
6. https://www.kholoodtechnotes.net/2023/12/python-loop.html
7. https://realpython.com/python-while-loop/
8. https://www.programiz.com/python-programming/while-loop
9. https://allinpython.com/python-math-module-with-example/#mathpowx_y
10. https://www.cs.fsu.edu/~cop3014p/lectures/ch7/index.html
11. https://www.toppr.com/guides/computer-science/introduction-to-python/void-functions-and-functions-returning-values/
12. https://www.programiz.com/python-programming/global-local-nonlocal-variables
13. https://harmash.com/tutorials/python/functions