مقدمة في البرمجة

م. غنام الجعبري

البرمجة

- البرمجة (Programming) هي كتابة الكود البرمجي باستخدام احد لغات البرمجة مثل لغة جافا وسي وبايثون
- البرنامج (program) هو سلسلة من التعليمات المكتوبة باحد لغات البرمجة لتنفيذ مهمة معينة على جهاز الحاسوب
- يطلق على برامج الحاسوب مصطلح البرمجيات (software)، ويطلق على تعليمات البرنامج مصطلح الكود المصدر (source code)
- يجب على المبرمج قبل كتابة الكود ان يقوم بتحديد الخطوات التي تؤدي الى الحل او صياغة الخوارزمية (algorithm) التي سوف يستخدمها في البرنامج

لغة بايثون

- بايثون (Python) لغة برمجة عالية المستوى، سهلة التعلم، متعددة الاستخدامات، مفتوحة المصدر، كائنية التوجه (OOP)
- قام بتطويرها الهولندي جويدو فان روزم (Guido van Rossum) في عام 1989

ميزات بايثون

- تعمل لغة بايثون على كافة انظمة التشغيل مثل لينكس ووينـدوز، ويمكن كتابة الكود على نظام تشغيل وتشغيله على نظام آخر دون اجراء تعديلات
- تمتاز لغة بايثون بانها لغة سهلة القراءة والفهم، ويمكن كتابة الكود بعدد اقل من الاسطر مقارنة بلغات برمجة اخرى
- تستخدم لغة بايثون في العديد من التطبيقات مثل برمجة الويب، والوسائط المتعددة، والامن السيبراني، والذكاء الاصطناعي
- يتم تنفيذ النصوص المكتوبة بلغة بايثون سطرا سطرا باستخدام المفسر (Interpreter)
- تعد لغة بايثون اكثر لغات البرمجة استخداما من قبل المبتدئين وحتى الخبراء

ادوات البرمجة

- الخطوة الاولى للعمل مع اي لغة برمجة هي اعداد الادوات اللازمة للبرمجة
- يستخدم معظم المبرمجين بيئة تطوير متكاملة (IDE) مثل Visual Studio Code لانشاء البرامج وكتابة الكود لانها توفر ميزة الاكمال التلقائي والكشف عن الاخطاء
- قبل البدء بكتابة الكود باستخدام المحرر او بيئة التطوير، يجب تثبيت مفسر بايثون الذي يقوم بترجمة الكود الى تعليمات برمجية يمكن للحاسوب تنفيذها
- يمكن الحصول على مفسر بايثون مجانا من موقعه على الانترنت www.python.org
- يمكن الحصول على برنامج Visual Studio Code مجانا من موقعه على الانترنت https://code.visualstudio.com
- بعد تثبيت برنامج Visual Studio Code، يجب تثبيت Python extension في البرنامج لاضافة ميزة الاكمال التلقائي والكشف عن الاخطاء عند كتابة الكود بلغة بايثون

عناصر لغة البرمجة

- تتألف لغة البرمجة من مجموعة من العناصر اهمها:
  - بناء البرنامج (Basic Syntax)
  - الكلمات المحجوزة (Keywords)
  - انواع البيانات (Data Types)
  - المتغيرات (Variables)
  - الادخال والاخراج (Input/Output)
  - العوامل (Operators)
  - اتخاذ القرار (Decision Making)
  - الحلقات (Loops)
  - المصفوفات (Arrays)
  - السلاسل (Strings)
  - الدوال (Functions)

بناء البرنامج

- تستخدم لغة بايثون المسافات البادئة (indentation) في تحديد بداية الكود ونهايته بدلا من الاقواس المتعرجة { } في لغات البرمجة الاخرى
- يفضل استخدام 4 فراغات (whitespace) كمسافة بادئة بدلا من علامة التبويب (tab)

```
for i in range(1,11):
    print(i)
    if i == 5:
        break
```

التعليقات

- تستخدم التعليقات (Comments) لتوضيح ما يقوم به جزء من الكود او لعدم تنفيذه
- يتم تجاهل التعليقات بالكامل اثناء تنفيذ البرنامج
- تبدأ التعليقات في لغة بايثون بالرمز # في بداية السطر

```
# This is a comment
# print out Hello
print("Hello")
```

الكلمات المحجوزة

- الكلمات المحجوزة (Keywords) هي كلمات معرفة مسبقا لا يمكن استخدامها في تسمية المتغيرات (variables) او الدوال (functions) لانها جزء من بناء البرنامج

False	await	else	import	pass
None	break	except	in	raise
True	class	finally	is	return
and	continue	for	lambda	try
as	def	from	nonlocal	while
assert	del	global	not	with
async	elif	if	or	yield

المتغيرات

- تستخدم المتغيرات (Variables) في تخزين القيم بشكل مؤقت في الذاكرة
- لا يتم التصريح عن المتغيرات في لغة بايثون مثل لغات البرمجة الاخرى، ويتم تحديد نوع البيانات للمتغيرات بشكل تلقائي

```
number = 25
name = "Ali"
```

- الثوابت (Constants) هي متغيرات قيمتها ثابتة مثل ثابت الدائرة (Pi) وثابت الجاذبية
- يفضل تسمية الثوابت بالاحرف الكبيرة لان الثوابت غير معرفة مسبقا في لغة بايثون

```
PI = 3.14
GRAVITY = 9.8
```

المتغيرات

- المعرفات (Identifiers) هي اسماء لتعريف المتغيرات والدوال
- في تسمية المعرفات يجب اتباع القواعد التالية:
  - ان يحتوي المعرف على احرف صغيرة او كبيرة او ارقام وعلامة الشرطة السفلية فقط
  - ان لا يبدأ المعرف برقم وانما حرف او علامة الشرطة السفلية
  - ان لا يكون المعرف احد الاسماء المحجوزة
  - لا يوجد تحديد لطول المعرف
- لغة بايثون من اللغات المميزة لحالة الحرف (case-sensitive) اي ان كلمة name تختلف عن Name في تسمية المتغيرات

انواع البيانات

- العدد الصحيح (Integer): عدد بدون كسور عشرية
- العدد الحقيقي (Float): عدد مع كسور عشرية
- السلسلة (String): نص محاط بعلامتي اقتباس مفردة (') او مزودجة (")
- القيم المنطقية (Boolean): قيمة صائبة (True) او خاطئة (False)

```
age = 42
price = 76.5
message = "Hello, Python!"
is_raining = True
```

- تتضمن لغة بايثون انواع اخرى من البيانات تشمل القائمة (List) والمجموعة (Tuple) والمجموعة الجزئية (Set) والدليل (Dictionary)

الادخال والاخراج

- تحتوي لغة بايثون على مجموعة من الدوال المعرفة مسبقا مثل دوال الادخال والاخراج
- تستخدم دالة ()print في طباعة نص او قيمة على الشاشة، وعند طباعة اكثر من قيمة يتم الفصل بين القيم باستخدام الفاصلة

```
print("Hello, Python!")
print(42)
print("Value:", 3.14)
```

- يمكن استخدام الوسيط sep لتحديد القيمة الفاصلة في طباعة اكثر من قيمة، والوسيط end لتحديد القيمة النهائية بعد طباعة القيم (القيمة الافتراضية هي سطر جديد '\n')

```
print("Hello", "Python", sep="-")  # Output: Hello-Python

print("Hello", end=" ")
print("World")  # Output: Hello World
```

الادخال والاخراج

- يستخدم الحرف f في بداية النص لتنسيق النص او دمج المتغيرات والعبارات في النص باستخدام الاقواس المتعرجة { }

```
name = "Ali"
age = 20
# Using f-string
print(f"My name is {name} and I am {age} years old.")
```

- يمكن ايضا تنسيق الاعداد باستخدام الحرف f مثل تحديد عدد المنازل العشرية باستخدام الصيغة :.2f لتقريب العدد الحقيقي الى اقرب منزلتين عشريتين

```
PI = 3.14159265
print(f"Pi is approximately {PI:.2f}")
```

الادخال والاخراج

- تستخدم دالة input() في ادخال نص او قيمة من المستخدم مع طباعة رسالة على الشاشة

```
name = input("Enter your name: ")
age = input("Enter your age: ")
```

- لتحويل النص او القيمة المدخلة الى عدد نستخدم دالة int() لتحويلها الى عدد صحيح او الدالة float() لتحويلها الى عدد حقيقي

```
age = int(input("Enter your age: "))
price = float(input("Enter product price: "))
```

تمارين: الادخال والاخراج

- اكتب برنامج بلغة بايثون لادخال اسم المستخدم (name) ثم طباعة رسالة ترحيب على الشاشة، مثلا اذا ادخل المستخدم كلمة Ali يقوم البرنامج بطباعة الرسالة Hello, Ali
- اكتب برنامج بلغة بايثون لادخال العدد (a) والعدد (b) ثم ايجاد مجموع العددين (sum)
- اكتب برنامج بلغة بايثون لادخال سعر المنتج (price) والكمية (quantity) ثم حساب السعر الاجمالي (total)
- اكتب برنامج بلغة بايثون لادخال درجة الحرارة (temp) ثم تحويل درجة الحرارة من مئوي (c) الى فهرنهايت (f) باستخدام المعادلة f = (temp * 9/5) + 32
- اكتب برنامج بلغة بايثون لادخال نصف القطر (radius) ثم حساب مساحة الدائرة (a) باستخدام المعادلة a = 3.14 * radius ** 2

العوامل

- العوامل (Operators) هي رموز خاصة او كلمات تستخدم في تنفيذ العمليات على القيم
- العوامل الحسابية (Arithmetic)
  - الجمع (+): يقوم بجمع العددين
  - الطرح (-): يقوم بطرح العدد الثاني من العدد الاول
  - الضرب (*): يقوم بضرب العددين
  - قسمة الاعداد الحقيقية (/): يقوم بقسمة العدد الاول على العدد الثاني
  - قسمة الاعداد الصحيحة (//): يقوم بقسمة العدد الاول على العدد الثاني
  - باقي القسمة (%): يرجع باقي قسمة العدد الاول على العدد الثاني
  - القوة او الأس (**): يقوم برفع العدد الثاني الى العدد الاول

```
x = 10
y = 3
sum = x + y
product = x * y
remainder = x % y
```

العوامل

- عوامل المقارنة (Comparison)
  - يساوي (==): يفحص اذا كانت القيمتين متساويتين
  - لا يساوي (!=): يفحص اذا كانت القيمتين غير متساويتين
  - اصغر من (<): يفحص اذا كانت القيمة الاولى اصغر من الثانية
  - اكبر من (>): يفحص اذا كانت القيمة الاولى اكبر من الثانية
  - اصغر من او يساوي (<=): يفحص اذا كانت القيمة الاولى اصغر من او يساوي الثانية
  - اكبر من او يساوي (>=): يفحص اذا كانت القيمة الاولى اكبر من او يساوي الثانية

```
a = 5
b = 10
is_equal = a == b
is_less_than = a < b
is_greater_than = a > b
```

العوامل

- العوامل المنطقية (Logical)
  - منطق و (and): يرجع True اذا كان كلا الشرطين True
  - منطق أو (or): يرجع True اذا كان احد الشرطين True
  - منطق النفي (not): يرجع True اذا كان الشرط False وبالعكس

```
age = 25
is_adult = age >= 18 and age <= 60
```

```
is_raining = True
has_umbrella = False

should_go_outside = not is_raining or has_umbrella
```

العوامل

- عوامل التعيين (Assignment)
  - التعيين (=): يقوم بتعيين القيمة الى المتغير
  - التعيين مع الجمع (+=): يقوم بعملية الجمع ثم تعيين النتيجة الى المتغير
  - التعيين مع الطرح (-=): يقوم بعملية الطرح ثم تعيين النتيجة الى المتغير
  - التعيين مع الضرب (*=): يقوم بعملية الضرب ثم تعيين النتيجة الى المتغير
  - التعيين مع قسمة الاعداد الحقيقية (/=): يقوم بعملية القسمة ثم تعيين النتيجة الى المتغير
  - التعيين مع قسمة الاعداد الصحيحة (//=): يقوم بعملية القسمة ثم تعيين النتيجة الى المتغير
  - التعيين مع باقي القسمة (%=): يقوم بعملية باقي القسمة ثم تعيين النتيجة الى المتغير

```
x = 10
x += 5  # Equivalent to x = x + 5
x *= 5  # Equivalent to x = x * 5
```

اتخاذ القرار

- تستخدم الجمل الشرطية في اتخاذ القرارات بناء على الشروط المذكورة
- تؤدي الجمل الشرطية الى التحكم في انسياب البرنامج من خلال تنفيذ كود مختلف بناء على الشرط ان كان صائبا او خاطئا
- الجمل الشرطية في لغة بايثون:
  - جملة if
  - جملة else
  - جملة elif

جملة if

- تستخدم جملة if لتنفيذ الكود اذا كان الشرط صائبا (True)
- يمكن ان يتبعها بشكل اختياري جملة else او جملة elif (اختصارا للكلمتين else if)

```
age = 20
if age >= 18:
    print("You are an adult.")
```

```
age = 25
if age >= 18 and age <= 60:
    print("You are in the working age group.")
```

جملة else

- تستخدم جملة else لتنفيذ الكود اذا كان الشرط في جملة if او جملة elif خاطئا (False)

```
age = 20
if age >= 18:
    print("You are an adult.")
else:
    print("You are a child.")
```

```
age = 25
if age >= 18 and age <= 60:
    print("You are in the working age group.")
else:
    print("You are not in the working age group.")
```

جملة elif

- تستخدم جملة elif عند فحص اكثر من شرط وتأتي بعد جملة if او جملة elif سابقة
- اذا كان الشرط في جملة if او جملة elif السابقة خاطئا، يتم فحص الشرط في جملة elif لتنفيذ الكود اذا كان الشرط صائبا

```
score = 85
if score >= 90:
    print("A")
elif score >= 80:
    print("B")
elif score >= 70:
    print("C")
elif score >= 60:
    print("D")
else:
    print("F")
```

تمارين: الجمل الشرطية

- اكتب برنامج بلغة بايثون لادخال عدد ثم فحص هل العدد سالب (negative)
- اكتب برنامج بلغة بايثون لادخال عدد ثم فحص هل العدد زوجي (even) او فردي (odd)
- اكتب برنامج بلغة بايثون لادخال كلمة المرور (password) ثم فحصها، اذا كانت python123 يتم طباعة (Access granted)، غير ذلك يتم طباعة (Access denied)
- اكتب برنامج بلغة بايثون لادخال عددين ثم يقوم المستخدم باختيار العملية الحسابية على العددين بادخال رمز العملية الحسابية (الجمع، الطرح، الضرب، القسمة) لايجاد الناتج
- اكتب برنامج بلغة بايثون لادخال عدد ثم فحص هل العدد موجب (positive)، واذا كان العدد موجبا هل العدد زوجي (even) او فردي (odd)

الحلقات

- تستخدم الحلقات لتكرار تنفيذ الكود عدة مرات او حتى يتحقق الشرط
- تعد الحلقات من لبنات التحكم الاساسية في لغة بايثون وتستخدم عادة مع المصفوفات
- تدعم لغة بايثون نوعين من الحلقات:
  - حلقة for: تستخدم لتكرار تنفيذ الكود عدد معين من المرات
  - حلقة while: تستخدم لتكرار تنفيذ الكود طالما الشرط يتحقق

حلقة for

- تستخدم دالة range() في توليد سلسلة ارقام لتحديد عدد مرات تنفيذ الكود في حلقة for
  - range(stop): تؤدي الى توليد سلسلة ارقام تبدأ من 0 وتنتهي عند stop-1

```
for i in range(10):
    print(i)
```

  - range(start, stop): تؤدي الى توليد سلسلة ارقام تبدأ من start وتنتهي عند stop-1

```
for i in range(1, 10):
    print(i)
```

  - range(start, stop, step): تؤدي الى توليد سلسلة ارقام تبدأ من start وتنتهي عند stop-1 مع زيادة بمقدار step كل دورة

```
for i in range(1, 10, 2):
    print(i)
```

حلقة for

- يمكن استخدام دالة range() في توليد سلسلة ارقام بترتيب تنازلي او عكسي

```
for i in range(10, 0, -1):
    print(i)
```

- يمكن استخدام الحلقات داخل حلقات اخرى، وفي هذه الحالة تدعى حلقات متداخلة (nested loops)
- تستخدم الحلقات المتداخلة عادة مع المصفوفات ذات البعدين (2D)

```
for i in range(3):
    for j in range(3):
        print(i, j)
```

```
for i in range(1,5):
    for j in range(i):
        print("*", end=" ")
    print()
```

حلقة for

- يمكن التحكم بانسياب الحلقة باستخدام جملتين هما:
  - جملة break: تؤدي الى انهاء الحلقة رغم تحقق شرط التكرار

```
for num in range(5):
    if num == 3:
        break
    print(num)
```

- جملة continue: تؤدي الى تجاوز بقية الكود في الدورة الحالية والانتقال الى الدورة التالية

```
for num in range(5):
    if num == 3:
        continue
    print(num)
```

حلقة while

- يستمر تنفيذ حلقة while اذا كان الشرط صائبا (True) وعندما يصبح خاطئا (False) يتم انهاء الحلقة

```
count = 0
while count < 5:
    print(count)
    count += 1
```

- تستخدم جملة break لانهاء الحلقة التي يتحقق بها الشرط باستمرار (infinite loop)

```
while True:
    user_input = input("Enter 'quit' to exit: ")
    if user_input == 'quit':
        break
    else:
        print("You entered:", user_input)
```

تمارين: الحلقات

- اكتب برنامج بلغة بايثون لادخال عدد من المستخدم ثم طباعة جدول الضرب للعدد
- اكتب برنامج بلغة بايثون لطباعة الاعداد الفردية (odd) من 1 الى 10
- اكتب برنامج بلغة بايثون لجمع الاعداد الفردية من 1 الى 10 ثم طباعة النتيجة
- اكتب برنامج بلغة بايثون لطباعة جدول الضرب للاعداد من 1 وحتى 10
- اكتب برنامج بلغة بايثون لطباعة مثلث قاعدته في الاعلى ورأسه في الاسفل باستخدام الرمز *
- اكتب برنامج بلغة بايثون لادخال عدد من المستخدم واضافة العدد الى المجموع (sum) مع تكرار عملية الادخال طالما كان العدد لا يساوي 0 ثم طباعة المجموع

القوائم

- تستخدم القوائم (Lists) في لغة بايثون مثل المصفوفات (arrays) في لغات اخرى، لكن القوائم تستطيع تخزين عدة قيم في المتغير من انواع بيانات مختلفة تشمل الاعداد والنصوص وحتى قوائم اخرى
- لانشاء قائمة يجب تحديد سلسلة من القيم بين قوسين مربعين [ ] وفصلها عن بعض باستخدام الفاصلة

```
fruits = ["apple", "banana", "orange"]
numbers = [1, 2, 3, 4, 5]
mixed = [1, "apple", True]
```

- يمكن الوصول الى عناصر القائمة باستخدام الفهرس (index) الذي يبدأ بالرقم 0 للعنصر الاول

```
first_fruit = fruits[0]  # Access the first element
second_number = numbers[1]  # Access the second element]
```

القوائم

- يمكن الوصول الى العنصر الاخير في القائمة باستخدام الفهرس 1- والعنصر قبل الاخير باستخدام الفهرس 2- وهكذا

```
last_fruit = fruits[-1]  # Access the last element
```

- القوائم قابلة للتعديل اي يمكن تعيين قيمة جديدة للعنصر باستخدام الفهرس

```
fruits[0] = "kiwi"  # Change the first element to 'kiwi'
```

- يمكن دمج قائمتين او اكثر مع بعض باستخدام الرمز +

```
all_items = fruits + numbers  # Concatenate 'fruits' and 'numbers' lists
```

- يمكن طباعة القائمة باستخدام دالة ()print

```
print(fruits)
print(numbers)
```

القوائم

- يمكن فحص هل العنصر موجود في القائمة باستخدام الكلمة in

```
fruits = ["apple", "banana", "orange"]
if "apple" in fruits:
    print("Yes, 'apple' is in the list.")
```

- يمكن استعراض عناصر القائمة باستخدام حلقة for

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

```
numbers = [1, 2, 3, 4, 5]
sum = 0

for num in numbers:
    sum += num
print("The sum is:", sum)
```

القوائم

- يمكن تحديد طول القائمة او عدد عناصرها باستخدام دالة len()

```
numbers = [1, 2, 3, 4, 5]
print(len(numbers))
```

- يمكن تعديل عناصر القائمة باستخدام حلقة for

```
numbers = [1, 2, 3, 4, 5]

for i in range(len(numbers)):
    numbers[i] += 1
print(numbers) # [2, 3, 4, 5, 6]
```

القوائم

- يمكن اضافة عنصر الى القائمة باستخدام دالة append()

```
numbers = [1, 2, 3]
numbers.append(4)
print(numbers)
```

- او باستخدام الرمز + (دمج القوائم)

```
numbers = [1, 2, 3]
numbers = numbers + [4]
print(numbers)
```

تمارين: القوائم

- اكتب برنامج بلغة بايثون لطباعة قائمة تحتوي على الاعداد من 1 الى 10
- اكتب برنامج بلغة بايثون لايجاد مجموع عناصر القائمة [10, 26, 34, 45, 5] ثم طباعة القائمة والمجموع (sum)
- اكتب برنامج بلغة بايثون للبحث عن اكبر عدد في القائمة [56, 23, 8, 67, 45, 12] ثم طباعة القائمة واكبر عدد (max)
- اكتب برنامج بلغة بايثون لتعديل عناصر القائمة [1, 2, 3, 4, 5] الى مربع العنصر ثم طباعة القائمة، مثلا اذا كان العنصر في القائمة 2 يصبح 4
- اكتب برنامج بلغة بايثون لايجاد معدل علامات طالب في قائمة مثلا [76, 92, 84] ثم طباعة المعدل (average) والتقدير باستخدام الاحرف

الدوال

- الدالة (Function) هي وحدة من الكود التي تؤدي مهمة معينة ويمكن اعادة استخدامها
- تتيح الدوال تقسيم البرنامج الى وحدات يمكن فهمها وادارتها بسهولة، على سبيل المثال يمكن تقسيم برنامج لرسم دائرة وتلوينها الى دالتين: دالة رسم الدائرة ودالة تلوين الدائرة
- يتم تعريف الدالة في لغة بايثون باستخدام الكلمة def ثم تسمية الدالة وتحديد المتغيرات الوسيطة (parameters) او المدخلات بين الاقواس ( )

```
def greet(name):
    print("Hello, " + name)
```

- تستخدم الدالة عبر استدعائها بالاسم مع القيم الفعلية (arguments)

```
greet("Alice")  # Calling the greet function with the argument "Alice"
```

الدوال

- يمكن تعيين قيم افتراضية للمتغيرات الوسيطة في الدالة عند استدعائها بدون قيم

```
def greet(name="Guest"):
    print("Hello, " + name)

greet()          # Prints "Hello, Guest"
greet("Alice")   # Prints "Hello, Alice"
```

- تستخدم جملة return في تعريف الدالة لارجاع القيم او النتائج

```
def add(a, b):
    return a + b

result = add(3, 5)
```

الدوال

- المتغيرات التي يتم تعريفها في داخل الدالة تدعى متغيرات محلية ولا يمكن الوصول اليها من خارج الدالة (local scope)
- المتغيرات التي يتم تعريفها في خارج كل الدوال تدعى متغيرات عامة ويمكن الوصول اليها من اي جزء في الكود (global scope)

```
global_var = 10

def my_function():
    local_var = 5
    print(global_var)  # Accessing a global variable
    print(local_var)   # Accessing a local variable

my_function()
```

تمارين: الدوال

- اكتب برنامج بلغة بايثون لتعريف الدالة print_numbers التي تستقبل العدد n وتقوم بطباعة الاعداد من 0 وحتى n-1
- اكتب برنامج بلغة بايثون لتعريف الدالة square التي تستقبل العدد n وترجع مربع العدد، مثلا الدالة square(2) ترجع القيمة 4
- اكتب برنامج بلغة بايثون لتعريف الدالة is_even التي تستقبل العدد n وترجع القيمة True اذا كان العدد زوجي او القيمة False اذا كان غير ذلك، مثلا الدالة is_even(2) ترجع القيمة True
- اكتب برنامج بلغة بايثون لتعريف الدالة calculate_price التي تستقبل سعر السلعة (item_price) ونسبة الضريبة (tax_rate) وترجع السعر (price)

الوحدات النمطية

- الوحدة النمطية (Module) هي ملف يحتوي على مجموعة من الدوال والمتغيرات المعرفة
- تتيح الوحدات النمطية تقسيم الكود الى ملفات اصغر يمكن صيانتها بسهولة، ومشاركة الكود واعادة استخدامه في برامج مختلفة
- تحتوي لغة بايثون على مكتبة قياسية غنية بالوحدات النمطية التي توفر العديد من الوظائف المختلفة مثل (math, random, os, datetime)
- يتم استيراد الوحدة النمطية بالكامل في لغة بايثون باستخدام الكلمة import

```
import math

num = 25
result = math.sqrt(num)
print(f"The square root of {num} is {result}")

radius = 3
area = math.pi * radius ** 2
print(f"The area of the circle with radius {radius} is {area}")
```

الوحدات النمطية

- يمكن استيراد بعض الدوال والمتغيرات من الوحدة النمطية باستخدام الكلمة from

```
from math import sqrt, pi

num = 25
result = sqrt(num)
print(f"The square root of {num} is {result}")

radius = 3
area = pi * radius ** 2
print(f"The area of the circle with radius {radius} is {area}")
```

- يمكن استيراد الوحدة النمطية بالكامل مع استخدام اسم بديل (alias) او مختصر للوحدة

```
import math as m

print(f"Factorial of 5 is {m.factorial(5)}")
```

التاريخ والوقت

- تستخدم الوحدة النمطية datetime في العمل مع التاريخ والوقت

```
from datetime import datetime

current_datetime = datetime.now()
print("Current Date and Time:", current_datetime)
```

- يمكن تنسيق التاريخ والوقت باستخدام الدالة strftime() او باستخدام الحرف f

```
from datetime import datetime

current_datetime = datetime.now()
formatted_date = current_datetime.strftime("%Y-%m-%d %H:%M:%S")
print("Formatted Date:", formatted_date)

# Using f-string
print(f"Formatted Date: {current_datetime:%Y-%m-%d %H:%M:%S}")
```

السلسلة

- السلسلة (String) هي مصفوفة من الاحرف، ولأن لغة بايثون لا تحتوي على نوع بيانات خاص بالاحرف مثل باقي لغات البرمجة فان الحرف في بايثون هو سلسلة طولها يساوي 1
- يمكن فحص هل النص موجود في سلسلة باستخدام الكلمة in

```
text = "Python is fun"
if "fun" in text:
    print("The text contains 'fun'")
```

- يمكن تحويل الاحرف في السلسلة الى احرف كبيرة باستخدام الدالة ()upper

```
text = "Python is fun"
print(text.upper())
```

- يمكن تحويل الاحرف في السلسلة الى احرف صغيرة باستخدام الدالة ()lower

```
text = "Python is fun"
print(text.lower())
```

السلسلة

- يمكن ازالة المسافات من بداية النص ونهايته باستخدام الدالة strip()

```
text = " Python is fun "
print(text.strip())
```

- يمكن تقسيم السلسلة بناء على فاصل معين باستخدام الدالة splite() (القيمة الافتراضية للفاصل في الدالة هي مسافة)

```
text = "Python is fun"
print(text.split(" "))
```

- يمكن تقطيع السلسلة الى اجزاء اصغر باستخدام الفهرس

```
text = "Python is fun"
# slice[start:stop:step]
print(text[0:6])
print(text[::2])
print(text[::-1])
```

تمارين: السلسلة

- اكتب برنامج بلغة بايثون لادخال جملة (sentence) من المستخدم ثم تقسيم الجملة الى كلمات (words) وطباعة عدد الكلمات في الجملة
- اكتب برنامج بلغة بايثون لادخال جملة من المستخدم ثم تقسيم الجملة الى كلمات وطباعة اطول كلمة في الجملة (largest)
- اكتب برنامج بلغة بايثون لادخال جملة من المستخدم ثم يقوم المستخدم باختيار تحويل الجملة الى احرف صغيرة (lowercase) او احرف كبيرة (uppercase)
- اكتب برنامج بلغة بايثون لادخال كلمة من المستخدم ثم فحص هل الكلمة تقرأ من اليمين الى اليسار او من اليسار الى اليمين بنفس الطريقة (palindrome) مثل كلمة madam

الملفات

- تستخدم جملة with في فتح الملف للتأكد من اغلاقه بشكل تلقائي بعد الاستخدام
- يمكن فتح الملف للقراءة (read) او الكتابة (write) او الاضافة (append)

```
# Open a file in read mode ('r')
with open('example.txt', 'r') as file:
    content = file.read() # or use readlines() to get a list of lines
print(content)
```

```
# Open a file in write mode ('w')
with open('example.txt', 'w') as file:
    file.write('This is a new content!\nYou can add more lines too.')
```

```
# Open a file in append mode ('a')
with open('example.txt', 'a') as file:
    file.write('\nThis line is appended to the file.')
```

الملفات

- يمكن قراءة الاسطر من الملف باستخدام حلقة for او دالة readline()

```
with open('example.txt', 'r') as file:
    # Read lines using a loop
    for line in file:
        print(line.strip()) # removes leading and trailing whitespaces
```

```
with open('example.txt', 'r') as file:
    # Read the first line
    line1 = file.readline().strip()
    # Read the second line
    line2 = file.readline().strip()
    # Read the third line
    line3 = file.readline().strip()

print("Line 1:", line1)
print("Line 2:", line2)
print("Line 3:", line3)
```

تمارين: الملفات

- اكتب برنامج بلغة بايثون لقراءة محتوى الملف source.txt ثم نسخ المحتوى الى الملف destination.txt
- اكتب برنامج بلغة بايثون لادخال ثلاثة اعداد من المستخدم ثم كتابة هذه الاعداد في الملف output.txt وكل عدد في سطر
- اكتب برنامج بلغة بايثون لقراءة الاعداد من الملف numbers.txt ثم ايجاد مجموع هذه الاعداد علما بان كل عدد في سطر
- اكتب برنامج بلغة بايثون لادخال ثلاثة اعداد من المستخدم وكتابة هذه الاعداد في الملف input.txt ثم قراءة الاعداد من الملف وايجاد مجموع هذه الاعداد

الاستثناء

- الاستثناء (Exception) هو محاولة التقاط الاخطاء اثناء تنفيذ البرنامج والتعامل معها بدلا من توقف البرنامج فجأة مثل قسمة عدد على صفر (ZeroDivisionError)
- تستخدم جملة try في لغة بايثون لمحاولة تنفيذ الكود وجملة except لالتقاط الخطأ في الكود والتعامل معه

```
try:
    number = int(input("Enter a number: "))
    result = 10 / number
    print("Result:", result)
except ZeroDivisionError:
    print("Error: You can't divide by zero.")
```

- يمكن استخدام جملة else في الاستثناء لتنفيذها اذا لم يحدث خطأ او جملة finally لتنفيذها دائما سواء حدث خطأ او لم يحدث

الاستثناء

- يستخدم الاستثناء في العمل مع الملفات لتجنب حدوث الاخطاء اثناء فتح الملف مثل الملف مفقود (FileNotFoundError) او صلاحية الوصول الى الملف (PermissionError)

```
try:
    with open("example.txt", "r") as file:
        content = file.read()
        print(content)
except FileNotFoundError:
    print("The file was not found.")
except PermissionError:
    print("You don't have permission to access this file.")
```

- يمكن اضافة جملة except الى الاستثناء للتعامل مع اي خطأ قد يحدث

```
except Exception as e:
    print("An error occurred:", e)
```

الدليل

- يستخدم الدليل (Dictionary) في لغة بايثون لتخزين ازواج من المفاتيح (keys) والقيم المرتبطة بكل مفتاح (values)
- لانشاء دليل يجب تحديد ازواج من المفاتيح والقيم بين قوسين متعرجين { } مع فصل المفتاح عن القيمة باستخدام النقطتين وفصل الازواج عن بعضهما باستخدام الفاصلة

```
person = {
    "name": "Ali",
    "age": 30,
    "city": "Hebron"
}
```

- يمكن الوصول الى القيم في الدليل باستخدام المفتاح

```
person["age"] = 31
print(person["name"])
```

الدليل

- يمكن استعراض المفاتيح والقيم في الدليل باستخدام حلقة for

```
for key, value in person.items():
    print(key, ":", value)
```

- يمكن تحديد عدد الازواج في الدليل باستخدام الدالة ()len

```
print(len(person))
```

- يمكن تخزين دليل داخل دليل آخر في لغة بايثون

```
students = {
    "student1": {"name": "Ali", "grade": "A"},
    "student2": {"name": "Sara", "grade": "B"}
}
```

تمارين: الدليل

- اكتب برنامج بلغة بايثون لانشاء دليل هاتف (contacts) يحتوي على الاسم (name) ورقم الهاتف (phone) ثم طباعة الاسماء والارقام في الدليل
- اكتب برنامج بلغة بايثون لانشاء دليل هاتف يحتوي على الاسم ورقم الهاتف ثم البحث عن الاسم وطباعة رقم الهاتف ان كان الاسم موجودا في الدليل
- اكتب برنامج بلغة بايثون لايجاد معدل العلامات في دليل علامات (scores) مثلا {'Math': 90, 'Science': 85, 'English': 88} ثم طباعة المعدل (average)