برمجة ونمذجة 2

بايثون Python 3

اعـــداد

د/ سهـــام عليـــو عامـــر
مدرس علوم الحاسب
كلية الحاسبات والمعلومات جامعة سوهاج

الباب الثاني

العمليات الحسابية والدوال الرياضية في Python

- استخدام الأعداد شائع جدًا في البرمجة، إذ تُستخدم لتمثيل مختلف القيم، مثل أبعاد حجم الشاشة، والمواقع الجغرافية، والمبالغ المالية، ومقدار الوقت الذي مر منذ بداية فيديو، والألوان وغير ذلك.
- **سنعمل مع أكثر نوعي البيانات استخدامًا في بايثون، وهما الأعداد الصحيحة والأعداد العشرية :**
- **الأعداد الصحيحة**: هي أعداد كاملة يمكن أن تكون موجبة أو سالبة أو معدومة (...، 1-، 0، 1، ...)
- **الأعداد العشرية**: هي أعداد حقيقية تحتوي على فاصلة عشرية (كما في 9.0 أو 2.25-)

العوامل

- **العامل** (operator): هو رمز أو دالة تمثل عملية حسابية. على سبيل المثال، في الرياضيات، علامة الجمع أو + هي العامل الذي يشير إلى عملية الجمع.

العملية	الناتج
x + y	مجموع x مع y
x - y	طرح x من y
-x	تغيير إشارة x
+x	قيمة x نفسها
x * y	ضرب x بـ y
x / y	قسمة x على y
x // y	حاصل القسمة التحتية لـ x على y
x % y	باقي قسمة x على y
x ** y	قيمة x مرفوعةً للقوة y

امثلـــة

```
print(1 + 5)
```

```
6
```

```
i = 3.3
print(+i)    #  3.3
```

```
a = 88
b = 103

print(a + b)
```

```
191
```

```
j = -19
print(+j)     #  -19
```

```
e = 5.5
f = 2.5

print(e + f)    #  8.0
```

```
k = 100.1
l = 10.1

print(k * l)    #  1011.0099999999999
```

امثلــة ..... تابع

```
m = 80
n = 5

print(m / n)    #   16.0
```

- هذا أحد الاختلافات الرئيسية بين بايثون 2 و بايثون 3. الإجابة في بايثون 3 تكون كسرية، فعند استخدام / لتقسيم 11 على 2 مثلًا، فستُعاد القيمة 5.5. أمَّا في بايثون 2، فحاصل التعبير 11/2 هو 5.
- يُجرِي العامل / في بايثون 2 **قسمة تحتية** (floor division)، إذ أنّه إن كان حاصل القسمة يساوي x، فسيكون ناتج عملية القسمة في بايثون 2 أكبر عدد من الأعداد الصحيحة الأصغر من أو تساوي x.
- في بايثون 3، يمكنك استخدام العامل // لإجراء القسمة التحتية. التعبير 40 // 100 سيعيد القيمة 2. القسمة التحتية مفيدة في حال كنت تريد أن يكون حاصل القسمة عددًا صحيحًا.

امثلــة ..... تابع

- معامل باقي القسمة (Modulo):

```
o = 85
p = 15

print(o % p)    #  10
```

- القوة (Power) :

```
s = 52.25
t = 7

print(s ** t)     #  1063173305051.292
```

- تقريب الأعداد:

```
hours = 24
total_bacteria = pow(2,hours)

print(total_bacteria)    # 16777216
```

امثلــة ..... تابع

سنكتب في المثال التالي برنامجًا بسيطًا يمكنه حساب البقشيش:

- ذهب 3 أصدقاء إلى مطعم، وأرادوا تقسيم الفاتورة، والتي تبلغ 87.93 دولارًا بالتساوي، بالإضافة إلى إكرامية (بقشيش) بنسبة 20٪.

```
bill = 87.93                    # إجمالي الفاتورة
tip = 0.2                       # بقشيش 20 ٪
split = 3                       # عدد الناس الذين سيتشاركون الفاتورة

total = bill + (bill * tip)     # حساب الفاتورة الإجمالية
each_pay = total / split        # حساب ما ينبغي أن يدفعه كل شخص
print(each_pay)                 # ما ينبغي أن يدفعه كل شخص قبل التقريب
print(round(each_pay,2))        # تقريب العدد - لا يمكننا تقسيم الفلسات
```

والمخرجات ستكون:

```
35.172000000000004
35.17
```

الـ Modules وFunction

- تقسم لغة البرمجة بايثون الى عدة Modules حيث تحتوي كل واحدة على تنفيذ عدة وظائف معينة مثال:

  Math : مسؤولة عن التعامل مع الارقام والعمليات الرياضية.

- يتم استدعاء ال Module بالعبارة Import ويكون الاستخدام على الشكل التالي:

Import Module

Module.function

- **مثال:**

Import math

math.sqrt

امثلـــة

```
print(pow(3,4))

print(abs(-12))

print(abs(6))
```

```
#--------------

import math

print(math.floor(17.6))

print(math.sqrt(81))

#Module.function
```

```
#Use variable

t=math.sqrt

print(t(9))
```

أسبقية العوامل

```
u = 10 + 10 * 5
```

```
60
```

```
u = (10 + 10) * 5
print(u)     #  100
```

امثلـــة .... تابع

عامل الإسناد (Assignment Operators)

```
w = 5
w += 1
print(w)    #  6
```

لدى بايثون عامل إسناد مركب مقابل لكل من العوامل الحسابية التي تم التطرق إليها في هذا الفصل :

```
y += 1      # إضافة القيمة ثم إسنادها
y -= 1      # طرح القيمة ثم إسنادها
y *= 2       # ضرب القيمة ثم إسنادها
y /= 3       # تقسيم القيمة ثم إسنادها
y // = 5     # تقسيم سفلي القيمة ثم إسنادها
y **= 2      # تنفيذ عامل الأس على القيمة ثم إسنادها
y %= 3      # إعادة باقي قسمة القيمة ثم إسناده
```

الدوال الرياضية المضمنة في بايثون 3

تتضمّن بايثون 3 العديد من الدوال التي يمكنك استخدامها بسهولة في أي برنامج. تتيح لك بعض تلك الدوال تحويل أنواع البيانات، والبعض الآخر خاص بنوع معين، مثل السلاسل النصية.

- **abs( )** : للحصول على القيمة المطلقة.
- **divmod( )** : للحصول على الحاصل والباقي في وقت واحد.
- **pow( )** : لرفع عدد لقوة معينة.
- **round()** : لتقريب عدد بمنازل عشرية محددة.
- **sum()** : لحساب مجموع العناصر في كائن قابلٍ للتكرار(iterable) .

امثلــة

- على سبيل المثال، إذا كنت تحاول الوصول إلى مكان يبعد 58 ميلًا، ولكنك تجاوزت ذلك المكان، وسافرت 93 فرسخًا .فإن حسبت عدد الفراسخ التي ينبغي أن تقطعها الآن للوصول إلى الوجهة المقصودة، فسوف ينتهي بك المطاف بعدد سالب، لكن لا يمكنك السفر عددًا سالبًا من الفراسخ!.

```
miles_from_origin = 58   # عدد الفراسخ التي تفصلنا عن الوجهة انطلاقًا من المُنطلق
miles_travelled = 93       # الفراسخ المقطوعة من المُنطَلق إلى الوجهة

# حساب عدد الفراسخ من الموقع الحالي
miles_to_go = miles_from_origin - miles_travelled

print(miles_to_go)           # طباعة عدد الفراسخ المتبقية - عدد سالب
print(abs(miles_to_go))    # حساب القيمة المطلقة للعدد السالب
```

المخرجات ستكون:

```
-35
35
```

أمثلة ..... تابع

```
divmod(a,b)
```

حساب حاصل القسمة والباقي بدالة واحدة

تكافئ هذه الدالة العمليتين التاليتين:

```
a // b

a % b
```

أمثلة ..... تابع

- لنفترض أننا كتبنا كتابًا يحتوي 80 ألف كلمة. يريد الناشر أن تحتوي كل صفحة من الكتاب ما بين 300 و 250 كلمة، ونود أن نعرف عدد الصفحات التي ستشكل الكتاب بحسب عدد كلمات الصفحة الذي اخترناه. باستخدام الدالة divmod( ) ، يمكننا أن نعرف على الفور عدد الصفحات في الكتاب، وعدد الكلمات المتبقية التي ستُنقل إلى صفحة إضافية.

```
words = 80000     # كم عدد الكلمات في كتابنا
per_page_A = 300    # 300 الخيار A: عدد كلمات الصفحة هو
per_page_B = 250    # 250 الخيار B: عدد كلمات الصفحة هو

print(divmod(words,per_page_A)) # A حساب الخيار
print(divmod(words,per_page_B)) # B حساب الخيار
```

وسينتج عن هذه الشيفرة:

```
(266, 200)
(320, 0)
```

العمليات المنطقية في بايثون

- هناك قيمتان فقط لنوع البيانات المنطقية، وهما **True** و **False.** تُستخدم الحسابات المنطقية في البرمجة لإجراء الموازنات، والتحكم في مسار البرنامج.
- تمثِّل القيم المنطقية قيم الحقيقة (Truth Values) في علم المنطق في الرياضيات. وتُكتب القيمتان True و False دائمًا بالحرفين الكبيرين T و F على التوالي، لأنهما قيمتان خاصتان في بايثون.

- **عوامل الموازنة**

العامل	الشرح
==	تساوي
=!	تخالف
>	أصغر من
<	أكبر من
=>	أصغر من أو تساوي
=<	أكبر من أو تساوي

امثلــة

```
x = 5
y = 8

print("x == y:", x == y)
print("x != y:", x != y)
print("x < y:", x < y)
print("x > y:", x > y)
print("x <= y:", x <= y)
print("x >= y:", x >= y)
```

```
x == y: False
x != y: True
x < y: True
x > y: False
x <= y: True
x >= y: False
```

والمخرجات هي:

امثلـــة ..... تابع

- المثال التالي يوضح كيفية موازنة السلاسل النصية:

```
Sammy = "Sammy"
sammy = "sammy"

print("Sammy == sammy: ", Sammy == sammy)          # Sammy == sammy:  False
```

السلسلة "Sammy" أعلاه لا تساوي السلسلة النصية "sammy" ، لأنهما ليستا متماثلتين تمامًا؛ فإحداهما تبدأ بحرف كبير S ، والأخرى بحرف صغير s . لاحظ الفرق بين العاملين = و== .

```
x = y   # إسناد قيمة y إلى x
x == y  # تتحقق مما إذا كان y و x متساويين
```

العوامل المنطقية

العامل	الشرح	الصياغة
and	إن كان كلا التعبيرين صحيحين True	x and y
or	إن كان أحد التعبيرين على الأقل صحيحًا True	x or y
not	إن كان التعبير خطأ True	not x

امثلــة:

```
print((9 > 7) and (2 < 4))     # كلا التعبيرين صحيحان
print((8 == 8) or (6 != 6))    # أحد التعبيرين صحيح
print(not(3 <= 1))             # التعبير الأصلي خاطئ
```

والمخرجات هي:

```
True
True
True
```

جداول الحقيقة (Truth Tables)

- المنطق مجال واسع، وسنكتفي في هذا الجزء ببعض الأفكار المهمة التي يمكن أن تساعدك على تحسين طريقة تفكيرك وخوارزمياتك.
- **جدول الحقيقة الخاص بالعامل== :**

x	==	y	القيمة المُعادة
True	==	True	True
True	==	False	False
False	==	True	False
False	==	False	True

جدول الحقيقة الخاص بالعامل AND :

x	and	y	القيمة المُعادة
True	and	True	True
True	and	False	False
False	and	True	False
False	and	False	False

جدول الحقيقة الخاص بالعامل OR

x	or	y	القيمة المُعادة
True	or	True	True
True	or	False	True
False	or	True	True
False	and	False	False

جدول الحقيقة الخاص بالعامل NOT :

not	x	القيمة المُعادة
not	True	False
not	False	True

استخدام العوامل المنطقية للتحكم في مسار البرنامج

للتحكم في مسار ونتائج البرنامج عبر العبارات الشرطية (flow control statements)، يمكننا استخدام **شرط** (condition) متبوعًا **بعبارة برمجية** (clause).

```
if grade >= 65:                    # شرط
    print("Passing grade")         # بند

else:
    print("Failing grade")
```

Thank you

So Much

For your attention