MOONHACK 2020

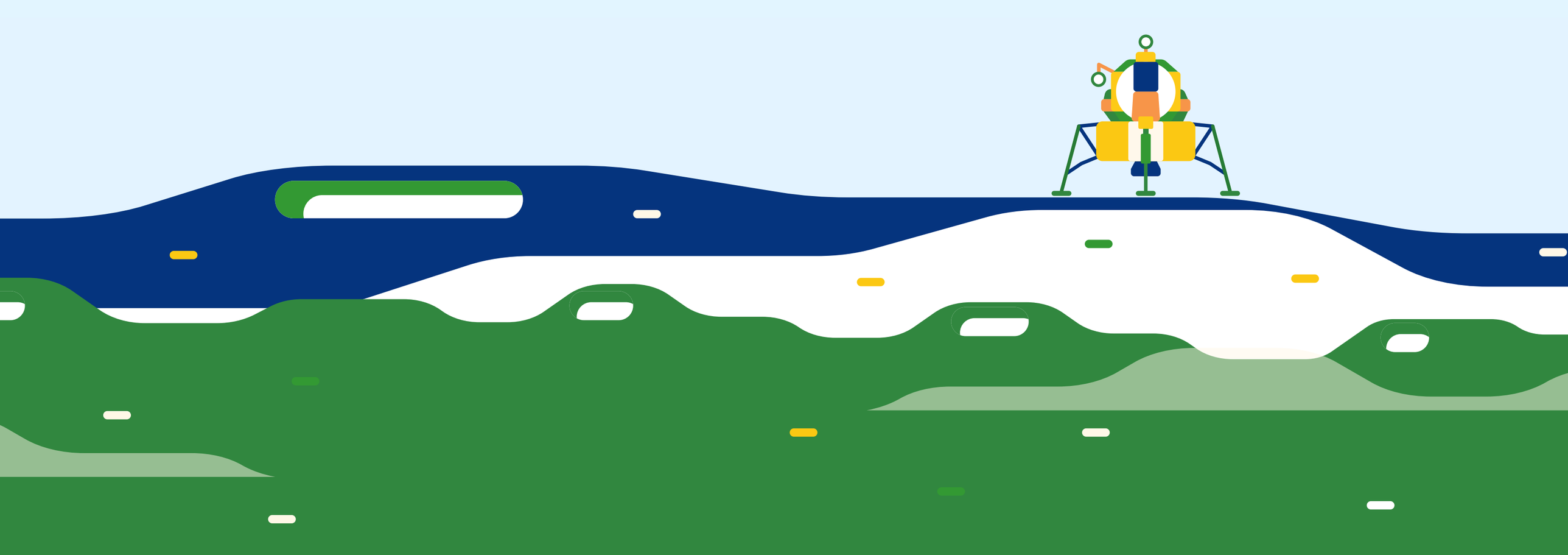
PYTHON WATER ON MARS

ARABIC

BROUGHT TO YOU BY CODE CLUB AUSTRALIA
POWERED BY TELSTRA FOUNDATION

code club / AUSTRALIA | POWERED BY TELSTRA FOUNDATION

SUBMIT AND BE COUNTED AT
MOONHACK.COM



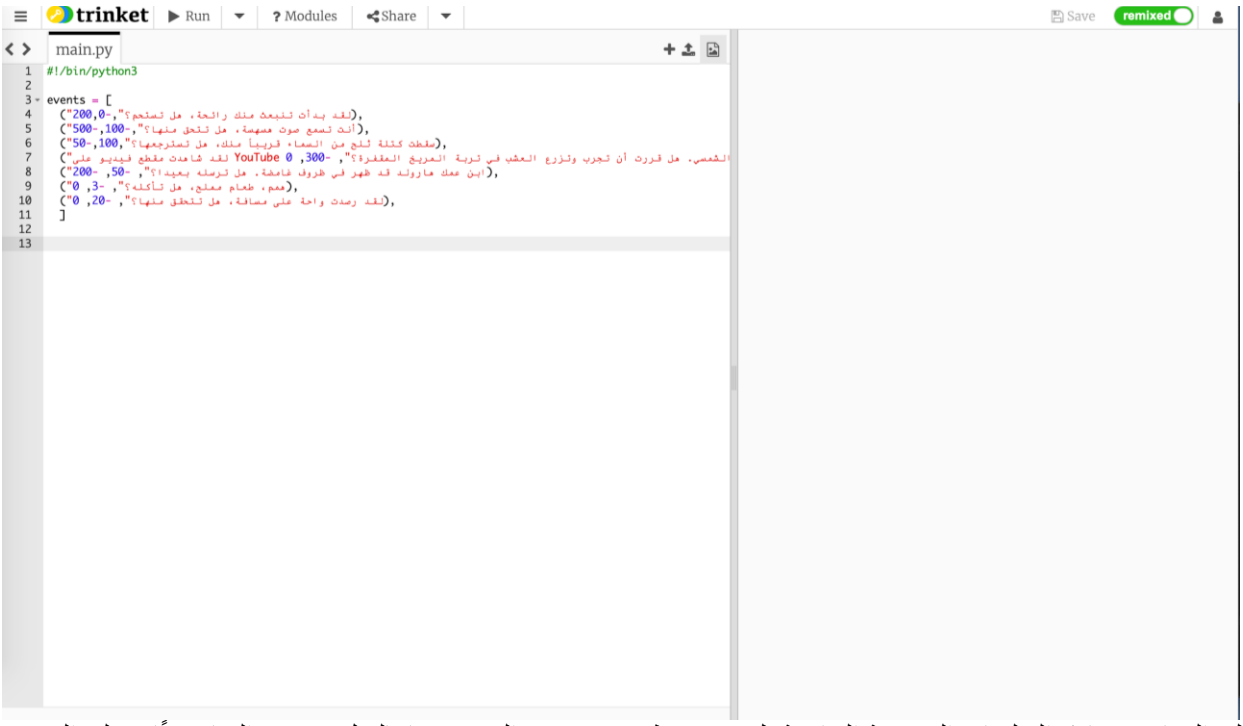
المقدمة

الماء مورد ثمين بشكل لا يصدق وهو أمر مهم للحياة. والحفاظ على المياه أمر مهم، وسوف يصبح أكثر أهمية لأن أجناسنا سوف تغامر في الفضاء. سنقوم اليوم بصنع لعبة بقاء تعتمد على النصوص، حيث يتعين على اللاعب اتخاذ قرارات واعية بالمياه من أجل البقاء لأطول فترة ممكنة دون نفاد المياه على المريخ.

خطوة ١: تتبع المياه

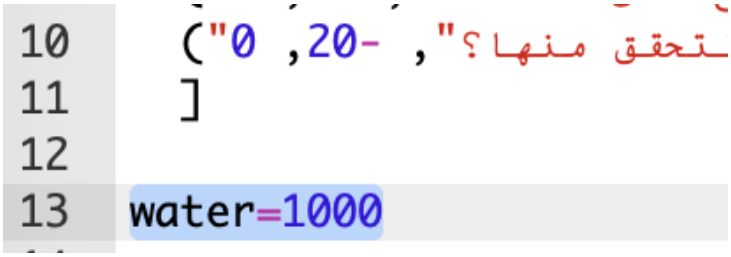
في الخطوة الاولى، سنبدأ لعبتنا، ونضع متغير يمكننا استخدامه لتتبع كمية المياه المتوفرة لنا.

- قم بفتح مشروع البداية على bit.ly/pythonwater . يجب أن تبدو نافذتك هكذا:

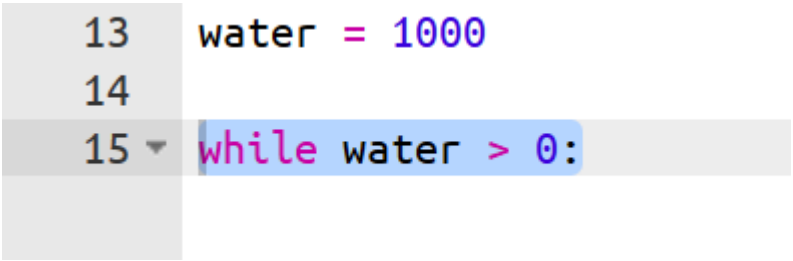


على اليسار هو نافذة التعليمات البرمجية الخاصة بك. ستلاحظ وجود بعض الرموز هنا بالفعل (سنعود إليها لاحقًا). على اليمين نافذة التنفيذ الخاصة بك ؛ هذا هو المكان الذي ستظهر به مخرجات اللعبة. في الجزء العلوي هو زر التشغيل. سوف تستخدم هذا الزر لتشغيل التعليمات البرمجية الخاصة بك.

- نحن بحاجة إلى إعطاء لاعبنا بعض الماء ليبدأ به. دعنا نبدأ برقم جولة مناسب 1000 لتر، أضف الكود المظلل التالي في أسفل البرنامج:



- نحن نرغب أن يستمر اللاعب باللعب طالما لديهم ماء. عندما تنفذ المياه، تنتهي اللعبة. دعونا نقوم بانشاء الحلقة التكرارية الرئيسية، والتي ستستمر طالما لدينا مياه:



لاحظ النقطتين الشارحة (:) في نهاية السطر. قد يبدو هذا الرمز غير مهم، ولكن لو نسيناه فإن برنامج بايثون Python سوف يتذمر!

- بعد ذلك ، دعونا نضيف بعض استخدامات المياه إلى حلقة لعبتنا الرئيسية. ستكون هذه كمية يتم طرحها يوميًا ، بغض النظر عن اختيارات اللاعب ، لأن اللاعب يحتاج للشرب. أضف الكود التالي داخل الحلقة الخاصة بك:

```
15 while water > 0:
16   water -= 5
```

لاحظ المساحات الموجودة أمام السطر ١٦ نسميها مسافة بادئة **indent**. هذا يخبر بايثون أن التعليمات على هذا الخط هي داخل الحلقة. بمعنى آخر ، يحدث ذلك في كل مرة تدور فيها حول حلقة اللعبة الرئيسية (إذا استخدمت Scratch ، فهي تشبه كتلة كرر حتى).

- الآن حان الوقت لبعض المخرجات. دعنا نقول للمستخدم أن اللعبة قد انتهت عندما تنفذ المياه. أضف الكود التالي خارج حلقة اللعبة الرئيسية (تذكر أن تتخلص من المسافة البادئة!):

```
15 while water>0:
16   water-=5
17
18 print(":( إنتهت اللعبة")
```

- قم بتنفيذ البرنامج ماذا يحدث؟

- سيحاول برنامج بايثون Python تشغيل التعليمات الخاصة بنا في اسرع وقت ممكن. بدون أي مخرجات من الحلقة الرئيسية ، فليس لدينا أي طريقة لمعرفة ما يحدث حتى نصل إلى النهاية. دعنا نضيف بعض المخرجات إلى الحلقة الرئيسية.

```
15 while water>0:
16   water-=5
17   print("لقد مرّ يوم.")
18
19 print(":( إنتهت اللعبة")
```

- شغّل البرنامج مرة أخرى. الآن يجب أن ترى كل الأيام التي تمر قبل انتهاء اللعبة

- دعنا نغير السطر الأخير من البرنامج الذي أدخلناه لإخبارنا عن كمية المياه التي أضفناها. يمكننا دمج قيمة المتغيرات الخاصة بنا مع النص الذي نقوم بطباعته باستخدام علامة +.

```
16 while water>0:
17   water-=5
18   print("كمية المياه لديك" " لقد مرّ يوم. " + water +" لتر ")
19
20 print("إنتهت اللعبة ")
```

- شغّل البرنامج مرة أخرى، هل تحصل على خطأ؟

- الجزء الكبير من البرمجة هو معرفة سبب حصولنا على الأخطاء التي نرتكبها. في هذه الحالة ، يخبرنا الخطأ أننا (لا نستطيع أن ندمج كائنات " str "و" int ("). عند البرمجة، سيتعين عليك في كثير من الأحيان معرفة معنى رسائل الخطأ الغريبة هذه.

يخبرنا هذا الخطأ أننا نحاول الجمع بين شيئين مختلفين لا تعرف بايثون Python كيفية الجمع بينهما، وهما النص (السلسلة، المختصرة بـ str) والرقم الذي نستخدمه لتتبع مياهنا (عدد صحيح، المختصر بـ int). لإصلاح ذلك، يمكننا أن نطلب من بايثون Python تحويل العدد الصحيح إلى سلسلة من خلال الكود التالي:

```
16 while water>0:
17   water-=5
18   print("كمية المياه لديك " + str(water) + " لتر ")
19
20 print("إنتهت اللعبة ")
```

- قم بتنفيذ البرنامج. يجب أن تحصل على ناتج من المخرجات يخبرنا مقدار كمية المياه التي نستخدمها في كل يوم، قبل أن تنفذ كمية المياه.

تحدي: غيّر كمية المياه المستخدمة يومياً

حاليًا ، نستخدم 5 لترات من المياه يوميًا. ما رأيك هو الحد الأدنى المطلق الذي يحتاجه الشخص للبقاء على قيد الحياة؟ هل يمكنك تغيير الكود بحيث يستخدم اللاعب هذه الكمية من الماء؟

خطوة ٢: تتبع الأيام

حتى الآن ، تابعنا كمية المياه التي استخدمها اللاعب ، ولكن لمعرفة مدى نجاحهم ، سنقوم بتتبع عدد الأيام التي انقضت.

- أولاً، سنقوم بتعيين القيمة الابتدائية لمتغير اليوم day، والذي يبدأ من اليوم ١.

```
13 water = 1000
14 day = 1
15
16 while water > 0:
```

- بعد ذلك، كل يوم يمر، نريد زيادة قيمة اليوم الحالي بواحد. أضف الكود التالي إلى حلقة لعبتك الرئيسية:

```
16 while water>0:
17   water-=5
18   print("كمية المياه لديك " + str(water) + " لتر ")
19   day+=1
20
21 print("إنتهت اللعبة ")
```

- أخيرًا ، سوف نسمح للاعب بمعرفة مدى نجاحه. سنفعل ذلك عن طريق استبدال جملة الطباعة النهائية بجملة تتضمن عدد الأيام التي انقضت.

```
19   day+=1
20
21 print("لقد صمدت " + str(day) + " أيام ")
```

لاحظ أنه كان علينا استخدام str(day) مرة أخرى ، لتحويل الرقم إلى نص.

- قم بتشغيل البرنامج الخاص بك. يجب أن تحصل على مخرجات لكل يوم ، ثم مخرج نهائي يوضح عدد الأيام التي استغرقها اللاعب.

تحدي: قم بإخراج رقم اليوم لكل يوم

حاليًا ، نحن نستخدم متغير اليوم فقط في النهاية. هل يمكنك تغيير الكود لإخراج رقم اليوم كل يوم؟

حاليا ، فإن رسائل الإخراج تقوم:
لقد مر يوم. كمية المياه لديك ٩٥٠ لتر

هل يمكنك تغييره ليقول:
يوم: ١٠. كمية المياه لديك ٩٥٠ لتر

• خطوة ٣: الاحداث اليومية الخاصة بك

الآن سنضيف بعض أحداث اللعب. كل يوم ، سنعطي اللاعب خيارًا ، وسيحدد هذا الخيار كمية المياه التي يفقدها أو يكسبها.

- ربما لاحظت وجود مجموعة كاملة من التعليمات البرمجية في البداية لم تكتبها. هذه قائمة بالأحداث التي قد تحدث على المريخ. دعنا نضيف جملة طباعة آخرى إلى الحلقة الرئيسية لدينا. سيؤدي ذلك إلى الوصول إلى قائمة الأحداث الخاصة بنا واختيار واحدة بشكل عشوائي.

```
18    print("اليوم " + str(day) + "." + "كمية المياه لديك " + str(water) + " لتر ")
19    print(random.choice(events))
20    day+=1
```

أوه لا! خطأ آخر. هذا واحد يقول "name 'random' is not defined" بمعنى أن المكتبة عشوائي random غير معرّفة. ذلك لأن هذه المكتبة ليست جزءًا افتراضياً من البايثون، وهي جزء من وحدة نمطية نحتاج إلى استيرادها حتى نتمكن من استخدامها.

```
1   #!/bin/python3
2   import random
3
4   events = [
```

- قم بتشغيل البرنامج، كل يوم يجب أن يكون لديك حدث عشوائي يجري طباعته.

- سنحتاج إلى استخدام الحدث عدة مرات في يوم معين. إذا أردنا استخدام random.choice (أحداث) في كل مرة أردنا استخدام الحدث ، فسنحصل على حدث مختلف! بدلاً من ذلك ، دعونا نخزن هذا الحدث في متغير يسمى الحدث event، والذي يمكننا استخدامه عدة مرات دون الحاجة إلى القلق.

```
19    print("اليوم " + str(day) + "." + "كمية المياه لديك " + str(water) + " لتر ")
20    event=random.choice(events)
21    print(event)
22    day+=1
```

- شغّل البرنامج مرة أخرى. يجب أن تحصل على نتيجة مماثلة للمرة السابقة.

- ربما لاحظت أن الحدث في الإخراج الخاص بنا محاط بمجموعة كاملة من المعلومات والرموز الإضافية. ذلك لأننا نخزن السؤال والقيم الخاصة بكمية المياه التي يجب كسبها أو خسارتها إذا أجاب اللاعب بنعم أو لا في نفس المكان. ما نريده حقًا هو طرح السؤال على اللاعب دون إظهار معلومات إضافية له. للوصول إلى السؤال، يمكننا استخدام ما يسمى تدوين الفهرس **index**. الفهرس هو الموضع الذي يوجد فيه العنصر فقط ، بدءًا من 0. غير جملة الطباعة إلى ما يلي:

```
20    event = random.choice(events)
21    print(event[0])
```

- قم بتشغيل البرنامج مجدداً. هذا أفضل بكثير، ولكن اللاعب لا يستطيع الاجابة على السؤال.
- بدلاً من استخدام جملة طباعة ، دعنا نستخدم جملة الإدخال. هذا يقول لبرنامج البايثون أن نتوقع من المستخدم أن يكتب شيئا.

```
20    event = random.choice(events)
21    input(event[0])
22    day += 1
```

الآن ، دعونا نخزن هذا الإدخال في متغير ، لأننا نريد استخدامه لاحقًا.

```
20    event = random.choice(events)
21    response = input(event[0])
22    day += 1
```

- شغّل البرنامج مرة أخرى. يجب أن تنتظر اللعبة الآن حتى يرد اللاعب كل يوم.

خطوة ٤: استخدام جواب اللاعب

يرد اللاعب على السؤال ، والآن نحتاج إلى استخدام ردهم للحكم على استخدامهم للمياه.

- يجب على المستخدم الإجابة إما بنعم أو لا على السؤال. إذا كانت الإجابة "نعم" ، فيجب إضافة الرقم الأول في الحدث:

```
22    response=input(event[0])
23    if response=="نعم":
24      water+=event[1]
25    day+=1
```

لاحظ أنه على الرغم من أننا نضيف ، إلا أن الكثير من الأرقام في الأحداث سالبة. تمامًا كما هو الحال في الرياضيات ، فإن إضافة رقم سالب هو نفس طرح العدد الموجب لذلك العدد

- إذا كانت إجابة اللاعب "لا"، علينا إضافة الرقم الثاني في الحدث.

```
23    if response=="نعم":
24      water+=event[1]
25    elif response=="لا":
26      water+=event[2]
27    day+=1
```

- قم بتشغيل البرنامج الخاص بك. يجب أن يكون اللاعبون الآن قادرين على الإجابة على الأسئلة. ستلاحظ أن كمية الماء التي يمتلكها اللاعب ستتغير وفقًا لقراراته. ماذا يحدث إذا أجاب اللاعب عن شيء بخلاف نعم أو لا؟

خطوة ٥: التحقق من صحة إدخال المستخدم

لديك الآن لعبة تعمل، لكن يمكن للاعب الإجابة بأي طريقة يريدها. نريد فقط قبول "نعم" و "لا" كإجابات.

- هناك خياران فقط ، نعم و لا، لذلك يمكننا مطالبة المستخدم بالإجابة على طريقتنا. وذلك بتعديل سطر الرد الخاص بنا إلى ما يلي:

```
21    event=random.choice(events)
22    response=input(event[0] + " نعم/لا: ")
23    if response=="نعم":
```

- هذا أفضل ، لكن لا يزال بإمكان اللاعب الإجابة بأي طريقة يريدها. يمكن للاعب إدخال شيء آخر من أجل خداع اللعبة، أو حدوث خطأ في الإجابة. دعنا نضيف عبارة "آخرى" لالتقاط أي رد بخلاف نعم أو لا.

```
25    elif response=="لا":
26      water+=event[2]
27    else:
28      print("يرجى الإجابة بـــ نعم أو لا!")
29
30    day+=1
```

- قم بتشغيل البرنامج. ماذا يحدث إذا قام اللاعب بالاجابة بشيء غير نعم أو لا؟
- نحن نطلب الآن من المستخدم تصحيح إجابته، ولكن بعد ذلك نطرح عليه السؤال التالي في كل الأحوال! نحتاج إلى الاستمرار في طرح نفس السؤال حتى يقدموا لنا إجابة نريد أن نسمعها. يمكننا أن نفعل هذا مع حلقة تكرارية.

```
21    event=random.choice(events)
22    while True:
23      response=input(event[0] + " نعم/لا: ")
24      if response=="نعم":
25        water+=event[1]
26      elif response=="لا":
27        water+=event[2]
28      else:
29        print("يرجى الإجابة بـــ نعم أو لا!")
30    day+=1
```

لاحظ أننا قمنا بوضع مسافة بادئة لسطر جملة response وسطر if / else. هذا يعني أن جميع هذه الجمل داخل الحلقة التكرارية. يمكنك القيام بذلك عن طريق تحديد الجمل التي تريد وضع مسافة بادئة لها والضغط على "علامة التبويب tab" الموجودة على لوحة المفاتيح.

- شغّل البرنامج. ماذا يحدث إذا أجبت عن شيء آخر بخلاف نعم أم لا؟ يجب أن يطرح عليك السؤال مرة أخرى. ولكن ماذا يحدث إذا أجبت بنعم أو لا؟ ما زال يطرح عليك السؤال نفسه مرة أخرى! يا ويلي!

- حلقة التكرار "while True" هي حلقة تبقى تعمل مستمرة إلى الأبد. هي مفيدة للغاية إذا كنت تريدها أن تعمل إلى الأبد، لكننا لا نريدها أن تعمل إلى الأبد، نريد فقط أن تعمل حتى يجيب المستخدم بنعم أو لا! لحسن الحظ، لدينا أمر خاص يمكن أن يخرجنا من حلقة التكرار، حتى لو كان أمرًا سيعمل إلى الأبد. هذا الأمر هو "break". دعنا نضيف أمر فاصل عندما يجيب اللاعب بنعم أو لا.

```
23      response=input(event[0] + " نعم/لا: ")
24      if response=="نعم":
25        water+=event[1]
26        break
27      elif response=="لا":
28        water+=event[2]
29        break
30      else:
```

- شغّل الكود مرة أخرى. يجب عليك الحصول على نفس السؤال مرة أخرى إذا كنت لا تجيب بنعم أو لا. الأمور تسير على ما يرام الآن.

- ~~Python is case sensitive. That means that if you are comparing two strings, it will think they are different strings just because they have different capitalisation. Fortunately, this is easy to fix. Python has a command that you can use on strings to make them all lower case. Change your response line to the following:~~

```
21    while True:
22      response = input(event[0] + " (yes/no): ").lower()
23      if response == "yes":
```

- ~~Run your code. It should now be accepting "Yes", "YES", "yeS" and any other combinations of capitalisation.~~

تهانينا!

لقد انتهيت من هذا المشروع. أحسنت! حاول أن تلعب لعبتك ومعرفة ما إذا كان يمكنك البقاء لفترة أطول من أصدقائك. تريد المزيد؟ جرب التحديات أدناه ، أو ألقي نظرة على مشاريع Scratch أو مشاريعنا من السنوات السابقة من مسابقة Moonhack.

تحدي: إضافة أحداث أخرى

هناك بعض الأحداث المختلفة ، ولكن هل يمكنك إضافة المزيد؟

تلميح: إذا كنت تواجه صعوبة ، فحاول نسخ أحد الأحداث الموجودة بالفعل وتعديله.

تحدي: لعبة الارض

هل يمكنك تعديل اللعبة لتكون حول الأرض بدلاً من المريخ؟ فكر في أنواع الأحداث التي قد تحدث ، والخيارات التي قد يواجهها اللاعب.

التحدي المتقدم: العب مرة أخرى، سام

هل يمكنك منح اللاعب خيارًا للاستمرار في اللعب بعد أن خسر اللعبة؟

تلميح: سوف تحتاج لعبتك أن تكون بأكملها في حلقة.