MOONHACK 2020

PYTHON WATER ON MARS

HINDI

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

मंगल पर पानी - मूनहॉक पायथन परियोजना (Water on Mars – Moonhack Python Project)

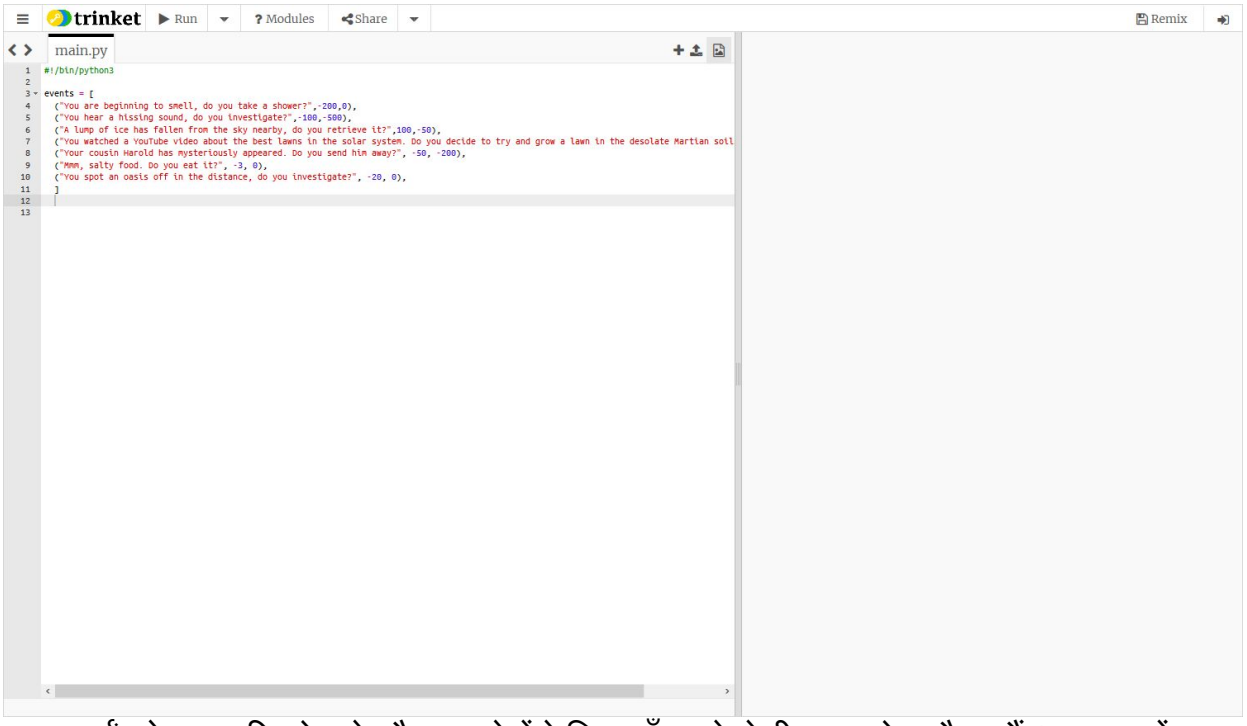
परिचय

पानी एक अविश्वसनीय रूप से कीमती संसाधन है जो जीवन के लिए महत्वपूर्ण है। पानी का संरक्षण करना महत्वपूर्ण है, और अधिक महत्वपूर्ण हो जायेगा जैसे हमारी प्रजाति अंतरिक्ष में और आगे जाने का साहस करेंगी। आज हम एक पाठ-आधारित उत्तरजीविता खेल बना रहे हैं, जिसमें खिलाड़ी को मंगल पर पानी खत्म होने से पहले यथासंभव लंबे समय तक जीवित रहने के लिए पानी के प्रति सचेत निर्णय लेने होंगे।

चरण 1: पानी का ट्रैक रखना

इस पहले चरण में, हम अपने खेल के साथ शुरुआत करने जा रहे हैं, और एक वेरिएबल सेट करते हैं जिसका उपयोग हम इस बात पर कर सकते हैं कि हमारे पास कितना पानी है।

- bit.ly/pythonwater पर स्टार्टर प्रोजेक्ट (starter project) खोलें । आपका कोड इस प्रकार दिखना चाहिए:



बाईं ओर आपकी कोड क्षेत्र है; आप देखेंगे कि यहाँ पहले से ही कुछ कोड मौजूद हैं (हम बाद में इसपर वापस आएँगे)। दाईं ओर आपकी निष्पादन (execution) क्षेत्र है; यह वह जगह है जहाँ आपके गेम का आउटपुट दिखाई देगा। शीर्ष पर रन (run) बटन है; आप अपने कोड को चलाने के लिए इस बटन का उपयोग करेंगे।

- हमें अपने खिलाड़ी को शुरू करने के लिए कुछ पानी देने की ज़रुरत है
- चलो 1000L के एक अच्छे दौर की संख्या के साथ शुरू करते हैं अपने कार्यक्रम के नीचे निम्नलिखित हाइलाइट किए गए कोड जोड़ें:

```
10    ("You spot an oasis
11    ]
12
13    water = 1000
```

- हम चाहते हैं कि हमारे खिलाडी खेल को तब तक खेलते रहें जब तक उनके पास पानी है। जब उनके पास पानी नहीं बचेगा, तो खेल खत्म हो जाएगा। आइए अपना मुख्य गेम लूप बनाएं, जो तब तक जारी रहेगा जब तक खिलाडी के पास पानी है:

```
13  water = 1000
14
15  while water > 0:
```

हमारी पंक्ति के अंत में कॉलन (:) देखें। यह छोटी सी चीज़ भले ही आपको महत्वपूर्ण ना लगे, लेकिन अगर हम इसे भूल जाते हैं, तो पायथन शिकायत करेगा!

- अब चलो हमारे मुख्य गेम लूप में कुछ पानी का उपयोग जोड़ें। यह एक ऐसी मात्रा होगी जो खिलाड़ी के चयन की परवाह किए बिना हर दिन घटाई जाएगी, क्योंकि खिलाड़ी को पानी पीने की ज़रूरत होती है। अपने लूप के अंदर निम्नलिखित कोड जोड़ें:

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

पंक्ति 16 के सामने दो रिक्त स्थान पर ध्यान दें। हम इसे **इंडेंट (indent) कहते हैं** । यह पायथन (python) को बताता है कि उस लाइन पर निर्देश लूप के अंदर है। दूसरे शब्दों में यह हर बार होता है जब हम मुख्य गेम लूप के चारों ओर जाते हैं (यदि आपने स्क्रैच का उपयोग किया है, तो यह नारंगी लूप ब्लॉक के समान है)।

- अब कुछ नतीजे का समय है। खिलाडी को बताएं कि जब उनके पास पानी नहीं बचेगा तो खेल खत्म हो जायेगा। अपने मुख्य गेम लूप के बाहर निम्नलिखित कोड जोड़ें (इंडेंट को हटाना याद रखें!):

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

- अपने कोड का परीक्षण करें। क्या होता है?
- पायथन सिर्फ हमारे कोड को जल्द से जल्द चलाने की कोशिश करेगा। मुख्य लूप में किसी भी आउटपुट के बिना, हमारे पास यह जानने का कोई तरीका नहीं है कि आखिर क्या हो रहा है। आइए हमारे मुख्य लूप में कुछ आउटपुट जोड़ें।

```
15  while water > 0:
16      water -= 5
17      print("A day has passed.")
18
19  print ("Game Over :(")
```

- अपने कोड का फिर से परीक्षण करें। अब आपको उन सभी दिनों को देखना चाहिए जो खेल समाप्त होने से पहले गुजरते हैं।
- आइए कोड की उस अंतिम पंक्ति को बदलते हैं जो हमने यह बताने के लिए डाला कि हमने कितना पानी जोड़ा। हम अपने वेरिएबल के मूल्य को उस पाठ के साथ जोड़ सकते हैं जिसे हम + चिह्न का उपयोग करके प्रिंट कर रहे हैं।

```
15 while water > 0:
16   water -= 5
17   print("A day has passed. Your Water " + water + "L")
18
19 print ("Game Over :(")
```

- अपने कोड का फिर से परीक्षण करें। क्या आपको कोई गलती मिलती है?
- प्रोग्रामिंग का एक बड़ा हिस्सा यह जानने की कोशिश कर रहा है कि हमें जो गलतियां मिल रही हैं, यह क्यों हैं। इस मामले में, त्रुटि हमें बता रही है कि हम "str 'और' int 'ऑब्जेक्ट्स (objects) को नहीं जोड़ सकते हैं"। जब प्रोग्रामिंग करते हैं, तो आपको अक्सर पता करना होगा कि इन अजीब त्रुटि संदेशों का क्या मतलब है।
  यह त्रुटि (error) बता रही है कि हम दो अलग-अलग चीजों को संयोजित करने का प्रयास कर रहे हैं जो कि पायथन जोड़ना नहीं जानता, पाठ (स्ट्रिंग को छोटा करके str) और वह संख्या जो हम अपने पानी का ट्रैक रखने के लिए उपयोग कर रहे हैं (इन्टिजर को छोटा करके int)। इसे ठीक करने के लिए, हम पायथन को निम्न कोड के साथ हमारे पूर्णांक को एक स्ट्रिंग में बदलने के लिए कह सकते हैं:

```
15 while water > 0:
16   water -= 5
17   print("A day has passed. Your Water " + str(water) + "L")
18
19 print ("Game Over :(")
```

- अपने कोड का परीक्षण करें। खिलाड़ी का पानी ख़तम होने से पहले आपको यह नतीजा आ राहा होगा कि हम प्रत्येक दिन कितना पानी इस्तेमाल कर रहे हैं।

चुनौती: दैनिक पानी के उपयोग को बदलें

वर्तमान में, हम प्रति दिन 5 लीटर पानी का उपयोग कर रहे हैं। आपको क्या लगता है कि व्यक्ति को जीवित रहने के लिए कितने न्यूनतम की आवश्यकता है? क्या आप कोड को बदल सकते हैं ताकि खिलाड़ी पानी की मात्रा का उपयोग करे?

चरण 2: दिनों का ध्यान रखना

अब तक, हमने एक खिलाड़ी द्वारा उपयोग किए गए पानी की मात्रा पर नज़र रखी है, लेकिन यह देखने के लिए कि उन्होंने कितना अच्छा काम किया है, हम उन दिनों की संख्या पर नज़र रखने जा रहे हैं जो बीत चुके हैं।

- सबसे पहले, हम अपने दिन वेरिएबल का प्रारंभिक मूल्य 1 निर्धारित करने जा रहे हैं।

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

- अगला, हर दिन जो बीतता है, हमें वर्तमान दिन के मूल्य को एक से बढ़ाना हैं। अपने मुख्य गेम लूप में निम्न कोड जोड़ें:

```
16 while water > 0:
17   water -= 5
18   print("A day has passed. Your Water " + str(water) + "L")
19   day += 1
20
21 print ("Game Over :(")
```

- अंत में, हम खिलाड़ी को यह बताने जा रहे हैं कि उन्होंने कितना अच्छा प्रदर्शन किया है। हम अंतिम प्रिंट स्टेटमेंट को एक बयान के साथ प्रतिस्थापित करेंगे, जिसमें शामिल होने वाले दिनों की संख्या शामिल है।

```
19    day += 1
20
21  print("you lasted " + str(day) + " days")
```

ध्यान दें कि हमने फिर से str (day) का उपयोग 'number' (पूर्णांक) को 'text'(स्ट्रिंग) में बदलने के लिए किया ।

- अपने कोड का परीक्षण करें। आपको हर दिन एक आउटपुट मिलना चाहिए, और फिर एक अंतिम आउटपुट जो कहता है कि खिलाड़ी कितने दिनों तक टिका रहा है।

चुनौती: प्रतिदिन की 'दिन' संख्या का आउटपुट

अभी हम दिन के वेरिएबल का उपयोग केवल अंत में कर रहे हैं। क्या आप प्रतिदिन दिन संख्या को आउटपुट करने के लिए कोड बदल सकते हैं?

अभी आउटपुट होगा:

```
A day has passed. Your Water 950L
```

क्या आप इसे यह कहने के लिए, बदल सकते हैं:

```
Day: 10. Your Water 950L
```

चरण 3: आपका दैनिक इवेंट (event)

अब हम खेल को और रोचक बनाने जा रहे हैं। हर दिन, हम खिलाड़ी को एक विकल्प देने जा रहे हैं, और यह विकल्प निर्धारित करेगा कि वे कितना पानी खो देते हैं या प्राप्त करते हैं।

- आपने शायद देखा है कि शुरू में बहुत सारा कोड है जो आपने नहीं लिखा। यह उन इवेंट्स की एक सूची है जो मंगल पर हो सकते हैं। आइए हमारे मुख्य लूप में एक और प्रिंट स्टेटमेंट (print statement) जोड़ें। यह हमारी इवेंट्स की सूची पर चलेगा और किसी भी एक का चयन करेगा।

```
18    print("A day has passed. Your Water " + str(water) + "L")
19    print(random.choice(events))
20    day += 1
```

उफ़्फ़ो! एक और त्रुटि। यह कहता है कि "name 'random' is not defined"। ऐसा इसलिए है क्योंकि random डिफ़ॉल्ट रूप से पायथन का हिस्सा नहीं है, यह एक **मॉड्यूल का हिस्सा है** । हमें इसका उपयोग करने में सक्षम होने के लिए इसे import करने की आवश्यकता है।

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

- अपने कोड का परीक्षण करें। हर दिन, आपको एक random इवेंट का प्रिंट आउटपुट मिलना चाहिए।
- हमें दिए गए दिन में कई बार ईवेंट का उपयोग करना होगा। यदि हर बार इवेंट का उपयोग करते समय, हम random.choice (घटनाओं) का उपयोग करते, तो हर बार हम एक अलग इवेंट प्राप्त करते! इसके बजाय, उस इवेंट को एक event नामक वेरिएबल में संग्रहीत करते हैं, जिसे हम कई बार उपयोग कर सकते हैं।

```
19    print("A day has passed. Your Water " + str(water) + "L")
20    event = random.choice(events)
21    print(event)
22    day += 1
```

- अपने कोड का फिर से परीक्षण करें। आपको पिछली बार के समान परिणाम मिलना चाहिए।
- आपने शायद देखा है कि हमारे आउटपुट में event अतिरिक्त सूचना और प्रतीकों के एक पूरे समूह से घिरी हुई है। ऐसा इसलिए है क्योंकि हम सवाल और पानी की मात्रा जो खिलाडी के हाँ या ना के अनुसार या तो मिलेगी या खोनी पड़ेगी, दोनों को एक ही जगह पर स्टोर कर रहे हैं। हम वास्तव में केवल खिलाड़ी को अतिरिक्त जानकारी दिखाए बिना प्रश्न पूछना चाहते हैं। प्रश्न का उपयोग करने के लिए, हम **index** notation का उपयोग कर सकते हैं । Index सिर्फ वह क्रमांक है जिसमें element रहता है, और जो 0 पर शुरू होता है। अपनी प्रिंट लाइन को निम्न में बदलें:

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

- अपने कोड का फिर से परीक्षण करें। यह बहुत बेहतर है, लेकिन खिलाड़ी सवाल का जवाब नहीं दे सकता है।

- प्रिंट स्टेटमेंट का उपयोग करने के बजाय, आइए एक इनपुट स्टेटमेंट का उपयोग करें। यह पायथन को उपयोगकर्ता से कुछ लिखने की उम्मीद करने के लिए कहता है।

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

अब, हम उस इनपुट को एक वेरिएबल में संग्रहीत करते हैं, क्योंकि हम इसे बाद में उपयोग करना चाहते हैं।

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

- अपने कोड का फिर से परीक्षण करें। आपके खेल को अब हर दिन के लिए खिलाड़ी के जवाब का इंतजार करना चाहिए।

चरण 4: खिलाड़ी के उत्तर का उपयोग करना

खिलाड़ी सवाल का जवाब दे रहा है, अब हमें उनकी प्रतिक्रिया का उपयोग उनके पानी के उपयोग को मापने के लिए करना है।

- उपयोगकर्ता को सवाल का जवाब हां या ना में देना चाहिए। यदि वे "yes" का जवाब देते हैं, तो हमें इवेंट में पहला नंबर जोड़ना चाहिए:

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

ध्यान दें कि भले ही हम जोड़ रहे हैं, इवेंट्स में बहुत सारी संख्याएं नेगेटिव हैं। जैसे मैथ्स में नेगेटिव नंबर जोड़ना वैसा ही होता है, जैसे कि उस नंबर का पॉजिटिव घटाना।

- यदि खिलाड़ी "no" का जवाब देता है, तो हमें इवेंट में दूसरा नंबर जोड़ना चाहिए।

```
22    if response == "yes":
23        water += event[1]
24    elif response == "no":
25        water += event[2]
26    day += 1
```

- अपने कोड का परीक्षण करें। खिलाड़ी को अब सवालों के जवाब देने में सक्षम होना चाहिए। आप देखेंगे कि खिलाड़ी के पास पानी की मात्रा उनके निर्णयों के अनुसार बदल जाएगी। अगर खिलाड़ी हां या ना के अलावा किसी और चीज का जवाब देता है तो क्या होगा?

चरण 5: उपयोगकर्ता के इनपुट को सत्यापित करें

अब आपका खेल ठीक प्रकार से चल रहा है, लेकिन खिलाड़ी किसी भी तरह का जवाब दे सकता है। हम उत्तर के रूप में केवल "yes" और "no" स्वीकार करना चाहते हैं।

- केवल दो विकल्प हैं, हां और नहीं, इसलिए हम उपयोगकर्ता को इन्ही में से जवाब देने के लिए संकेत कर सकते हैं। अपनी प्रतिक्रिया पंक्ति को निम्न में बदलें:

```
20    event = random.choice(events)
21    response = input(event[0] + " (yes/no): ")
22    if response == "yes":
```

- यह बेहतर है, लेकिन खिलाड़ी अभी भी किसी भी तरह का जवाब दे सकता है जो वे चाहते हैं। खेल को धोखा देने के लिए एक खिलाड़ी कुछ और दर्ज कर सकता है, या उनके जवाब को गलत बता सकता है। आइए हां या नहीं के अलावा किसी भी प्रतिक्रिया को पकड़ने के लिए "else" जोड़ें।

```
24    elif response == "no":
25      water += event[2]
26    else:
27      print("Please answer yes or no!")
28    day += 1
```

- अपने कोड का परीक्षण करें। अगर खिलाड़ी हां या ना के अलावा किसी और चीज का जवाब देता है तो क्या होगा?
- अब हम उपयोगकर्ता से अपनी प्रतिक्रिया सही करने के लिए कह रहे हैं, लेकिन फिर हम उनसे अगला सवाल भी पूछ रहे हैं! हमें उनसे वही सवाल पूछते रहने की जरूरत है जब तक वे हमें एक जवाब नहीं दे देते, जिसे हम सुनना चाहते हैं। हम इसे लूप के उपयोग से कर सकते हैं।

```
20    event = random.choice(events)
21    while True:
22      response = input(event[0] + " (yes/no): ")
23      if response == "yes":
24        water += event[1]
25      elif response == "no":
26        water += event[2]
27      else:
28        print("Please answer yes or no!")
29    day += 1
```

ध्यान दें कि हमने प्रतिक्रिया रेखा और यदि / अन्यथा कथन का संकेत दिया है। इसका मतलब है कि यह सब लूप में है। आप इसे उन पंक्तियों पर प्रकाश डालकर कर सकते हैं जिन्हें आप अपने कीबोर्ड पर इंडेंट करना चाहते हैं और 'टैब' दबा रहे हैं।

- अपने कोड का परीक्षण करें। यदि आप किसी चीज़ का उत्तर हाँ या ना *के अलावा* देते हैं तो क्या होता है? यह आपको फिर से सवाल पूछना चाहिए। लेकिन क्या होता है अगर आपहां या ना में उत्तर दें? यह आपको फिर से सवाल पूछेगा। अरे बाप रे!
- "while True" लूप एक लूप है जो हमेशा लूपिंग करता रहेगा। यदि आप हमेशा के लिए चलाना चाहते हैं तो ये काफी उपयोगी हैं, लेकिन हम हमेशा के लिए नहीं चलाना चाहते हैं, हम केवल तब तक चलाना चाहते हैं जब तक उपयोगकर्ता हां या ना में जवाब न दे दें! सौभाग्य से, हमारे पास एक विशेष कमान है जो हमें एक लूप से बाहर निकाल सकती है, भले ही वह वो हो जो अन्यथा हमेशा के लिए चलती जाएगी। वह आदेश "break" है। जब खिलाड़ी हां या ना में जवाब देता है तो एक ब्रेक कमांड जोड़ें।

```
22      response = input(event[0] + " (yes/no): ")
23      if response == "yes":
24          water += event[1]
25          break
26      elif response == "no":
27          water += event[2]
28          break
29      else:
```

- अपने कोड का फिर से परीक्षण करें। यदि आपको हां या नहीं में जवाब नहीं आता है तो ही सवाल फिर से करना चाहिए।
  चीजें अब अच्छी तरह से काम कर रही हैं, लेकिन एक और छोटा सा मुद्दा है जो कुछ खिलाड़ियों के खिलाफ आ सकता है। क्या होता है, अगर "yes" का जवाब देने के बजाय, खिलाड़ी एक वाई के साथ "Yes" का जवाब देता है?
- पायथन केस संवेदनशील (case sensitive) है। इसका मतलब है कि यदि आप दो स्ट्रिंग की तुलना कर रहे हैं, तो यह सोचेंगे कि वे अलग-अलग स्ट्रिंग हैं क्योंकि उनके पास अलग-अलग पूंजीकरण है। यह तय करना आसान है। पायथन में एक कमांड है जिसे आप स्ट्रिंग्स पर उपयोग कर सकते हैं ताकि वे सभी निचले केस बना सकें। अपनी प्रतिक्रिया पंक्ति को निम्न में बदलें:

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

- अपने कोड का परीक्षण करें। इसे अब “Yes”, “YES”, “yeS” और के किसी भी अन्य संयोजन को स्वीकार करना चाहिए।

बधाई हो!

आपने यह प्रोजेक्ट पूरा कर लिया है। बहुत बढ़िया! अपने खेल को खेलने की कोशिश करें और देखें कि क्या आप अपने दोस्तों से अधिक समय तक खेल सकते हैं। और चाहिए? नीचे दी गई चुनौतियों का प्रयास करें, या मूनहॉक के पिछले वर्षों के हमारे स्क्रैच प्रोजेक्ट या परियोजनाओं पर एक नज़र डालें।

चुनौती: अधिक इवेंट्स को जोड़ें

कुछ अलग ईवेंट हैं, लेकिन क्या आप और जोड़ सकते हैं?

संकेत: यदि आपको कठिनाई है, तो पहले से मौजूद इवेंट्स में से एक को कॉपी करने और संपादित करने का प्रयास करें।

चुनौती: टाइमर गेम

क्या आप मंगल ग्रह के बजाय पृथ्वी के बारे में होने वाले खेल को संशोधित कर सकते हैं? उन इवेंट्स के प्रकारों के बारे में सोचें जो हो सकती हैं, और एक खिलाड़ी को जिन विकल्पों का सामना करना पड़ सकता है।

उन्नत चुनौती: इसे फिर से चलायें

क्या आप खिलाड़ी को खेल हारने के बाद भी खेलते रहने का विकल्प दे सकते हैं?

संकेत: इसके लिए आपका पूरा खेल एक लूप में हो।