Table Of Contents

പാഠം അഞ്ച്

ജനിച്ച വർഷം കണ്ടുപിടിക്കുന്നതെങ്ങനെ?

കഴിഞ്ഞ പാഠത്തിന്റെ കമന്റുകളിൽ "Free" എന്ന വായനക്കാരി (രൻ?) അവതരിപ്പിച്ച പ്രവർത്തനം ഒന്നു നോക്കാം. ചോദ്യം ഇതാണ്: " ഒരാളുടെ പേരും വയസും ഇൻപുട്ട് ആയി എടുത്ത് അയാൾ ജനിച്ച വർഷം ഔട്പുട്ട് തരുന്ന പ്രോഗ്രാം എഴുതുക." ഈ പ്രവർത്തനത്തിന് "digitaleye" എന്ന വായനക്കാരൻ (രി?) തന്ന ഉത്തരം കുറച്ചുകൂടി വിശദമാക്കി എഴുതിയ ഒരു പ്രോഗ്രാം ഇതാ. പ്രോഗ്രാം ഒരു ഫയലിലേക്ക് സേവ് ചെയ്ത് പല പ്രാവശ്യം പ്രവർത്തിപ്പിച്ചു നോക്കൂ. ഇതിനുള്ള എളുപ്പവഴി IDLE ഉപയോഗിക്കുക എന്നതാണ്, ഇവിടെ പറഞ്ഞിരിക്കുന്നതുപോലെ.

```
# This program asks for the name and age (in number of
# years) of the user, and outputs the year when the user was
# born.

name = raw_input("Hello, what is your name? ")

question = name + ", please tell me your age in years: "

age = input(question)

current_year = 2010

year_of_birth = current_year - age

print name, ", you were born in ", year_of_birth
print "Goodbye!"
```

പ്രോഗ്രാമിന്റെ അഞ്ചാമത്തെ വരിയിൽ ഉപയോക്താവിന്റെ പേര് `raw_input()` ഉപയോഗിച്ച് പ്രോഗ്രാമിലെ `name` എന്ന ചരത്തിലേക്ക് എത്തിക്കുന്നു. ഇവിടെ ഉപയോക്താവിനോട് ചോദിക്കേണ്ട ചോദ്യം ഒരു string ആയി തയ്യാറാക്കിയിരിക്കുന്ന രീതി പ്രത്യേകം ശ്രദ്ധിക്കുക. `+` എന്ന സംകാരകം (operator) സംഖ്യകളുടെ മാത്രമല്ല, string-കളെ കൂട്ടിച്ചേർത്ത് ഒറ്റ string ആക്കാനും പൈത്തണിൽ ഉപയോഗിക്കാം. ഉദാഹരണം:

```
Python 2.6.5 (r265:79063, Apr 16 2010, 13:09:56)
[GCC 4.4.3] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> 1+1
2
>>> "1" + "1"
'11'
>>>
```

മുകളിലെ പ്രോഗ്രാമിൽ ഉപയോക്താവിനെ പേരുൾപ്പടെ സംബോധന ചെയ്യാനായി ഈ വിദ്യ പ്രയോഗിച്ചിരിക്കുന്നു. ഏഴാമത്തെ വരിയിൽ ഇൻപുട്ട് ആയി കിട്ടിയ പേരും ചോദിക്കേണ്ട ചോദ്യവും `+` എന്ന സംകാരകം ഉപയോഗിച്ച് ഒറ്റ string ആക്കി `question` എന്ന ചരത്തിൽ സൂക്ഷിക്കുന്നു. ഒൻപതാമത്തെ വരിയിൽ ഉപയോക്താവിനോട് വയസ് ചോദിച്ച് `age` എന്ന ചരത്തിൽ സൂക്ഷിക്കുന്നു. ഇതിനു വേണ്ടി മുമ്പത്തെ പാഠത്തിൽ കണ്ടതുപോലെ `input()` ഉപയോഗിക്കുന്നു. `input()` -ന്റെ ബ്രായ്ക്കറ്റിനുള്ളിൽ ഉപയോക്താവിനോട് ചോദിക്കേണ്ട ചോദ്യം കൊടുക്കേണ്ടിടത്ത് `question` എന്ന ചരം കൊടുത്തിരിക്കുന്നത് ശ്രദ്ധിക്കുക. ഈ ചരത്തിന്റെ വില (മൂല്യം, value) നാം ഏഴാമത്തെ വരിയിൽ `+` ഉപയോഗിച്ച് തയ്യാറാക്കിയ string ആണ്. പ്രോഗ്രാമിൽ ഏതെങ്കിലും വില വേണ്ടതായ സ്ഥലങ്ങളിലൊക്കെ അതിനു പകരം അതേ വിലയുള്ള ചരം ഉപയോഗിക്കാം. ഇവിടെ നാം ചെയ്തിരിക്കുന്നതും ഇതുതന്നെ.

പ്രോഗ്രാമിന്റെ പതിനൊന്നാമത്തെ വരിയിൽ നടപ്പുവർഷം ഏതെന്ന് `current_year` എന്ന ചരത്തിൽ സൂക്ഷിച്ചുവെയ്ക്കുന്നു. അർത്ഥവത്തായ പേരുകളുള്ള ചരങ്ങളുടെ ഉപയോഗം പ്രോഗ്രാം വായന സുകരമാക്കാൻ സഹായിക്കുന്നു. നമ്മുടെ പ്രോഗ്രാം വളരെ ചെറിയ ഒന്നായതുകൊണ്ട് ഇങ്ങനെ ചെയ്യുന്നതുകൊണ്ട് — `2010` എന്ന് നേരിട്ട് പറയുന്നതിനു പകരം `current_year` എന്ന ചരത്തിൽ ഈ വില സൂക്ഷിച്ചുവെച്ച് നടപ്പുവർഷം വേണ്ടിടത്തെല്ലാം ഈ ചരം ഉപയോഗിക്കുന്നതുകൊണ്ട് — ഉള്ള പ്രയോജനം എന്താണെന്നത് അത്രകണ്ട് വ്യക്തമല്ല. എന്നാൽ കുറച്ച വലിയ ഒരു പ്രോഗ്രാമിലോ? പേരില്ലാത്ത കുറെയേറെ സംഖ്യകൾ പ്രോഗ്രാമിൽ ഉണ്ടെങ്കിൽ അത് പ്രോഗ്രാം വായിക്കുന്ന ആളെ കഴക്കാനേ ഉപകരിക്കൂ. ഓരോ സംഖ്യയും എന്തർത്ഥത്തിലാണ് ഉപയോഗിച്ചിരിക്കുന്നത് എന്നറിയാൻ ആ സംഖ്യ എവിടെയൊക്കെയാണ് ഉപയോഗിച്ചിരിക്കുന്നതെന്ന് കണ്ടുപിടിച്ച് തല പുകയ്ക്കേണ്ടിവരും. അതുകൊണ്ടുതന്നെ ഇങ്ങനെയുള്ള സംഖ്യക്കളെ അർത്ഥം വ്യക്തമാകുന്ന രീതിയിലുള്ള പേരുകൾ കൊടുത്ത് പ്രോഗ്രാമിൽ ഉപയോഗിക്കുന്നതാണ് മെച്ചം, ഇവിടെ ചെയ്തതുപോലെ.

പ്രോഗ്രാമിന്റെ പതിമൂന്നാമത്തെ വരിയിൽ ഉപയോക്താവ് ജനിച്ച വർഷം കണ്ടുപിടിക്കാനായി നടപ്പുവർഷത്തിൽനിന്ന് വയസ് കുറയ്ക്കുന്നു. അവസാനത്തെ രണ്ടു വരികളിൽ ഇങ്ങനെ കിട്ടിയ ഉത്തരം ഔട്പുട്ട് ചെയ്യുകയും ചെയ്യുന്നു.

ബഗ്!

ഇത് താരതമ്യേന വളരെ ചെറിയതായ, ലളിതമായ ഒരു പ്രോഗ്രാം ആണ്. "Free" അവതരിപ്പിച്ച പ്രവർത്തനത്തിന് ഈ പ്രോഗ്രാം മതിയായ ഉത്തരവും ആണ്. എന്നാൽ ഈ

പ്രോഗ്രാമിന് അത്ര ചെറുതല്ലാത്ത ഒരു പ്രശ്നം ഉണ്ട്. ഈ പ്രശ്നം കാരണം ഒട്ടേറെപ്പേരുടെ കാര്യത്തിൽ ഈ പ്രോഗ്രാം തെറ്റായ ഉത്തരം തരും! ഒന്നാന്തരം ഒരു ബഗ് (കീടം?) നമ്മുടെ പ്രോഗ്രാമിൽ ഉണ്ട് എന്നർത്ഥം. ഇതുപോലെയുള്ള പ്രശ്നങ്ങൾ പ്രോഗ്രാമിന്റെ ലോജിക്കിൽ നഴഞ്ഞുകയറാതെ ശ്രദ്ധിക്കാൻ പഠിക്കുക എന്നത് നല്ല രീതിയിൽ പ്രോഗ്രാം എഴുതാൻ പഠിക്കുന്നതിന്റെ അതിപ്രധാനമായ ഭാഗമാണ്. ഒരാൾ ജനിച്ച വർഷം കണ്ടുപിടിക്കാനുള്ള മുകളിൽക്കൊടുത്ത പ്രോഗ്രാം എപ്പോഴൊക്കെയാണ് തെറ്റായ ഉത്തരം തരുന്നത് എന്ന് ആലോചിച്ചു നോക്കൂ. ഒരു പത്തുമിനിട്ട് മനസ്സിരുത്തി ആലോചിച്ച്, പ്രോഗ്രാമിന് കിട്ടാവുന്ന പല തരത്തിലുള്ള (ശരിയായ) ഇൻപുട്ടുകൾ ഏതൊക്കെയാണ്, അവയെല്ലാം പ്രോഗ്രാം ശരിയായ മറുപടി തരുമോ എന്നൊക്കെ തിരിച്ചും മറിച്ചും ചിന്തിച്ചുനോക്കി പ്രശ്നം പിടികിട്ടുന്നില്ലെങ്കിൽ മാത്രം അടുത്ത ഖണ്ഡിക നോക്കുക.

...
...
...
...
...
...

നാം പ്രോഗ്രാം പ്രവർത്തിപ്പിക്കുന്നത് 2010 ഓഗസ്റ്റ് തുടക്കത്തിൽ ആണെന്ന് കരുതുക. പ്രോഗ്രാമിലെ പ്രശ്നം എന്താണെന്ന് മനസ്സിലാക്കാൻ ഹരി, ജോൺ എന്ന രണ്ടു സുഹൃത്തുക്കളുടെ കാര്യമെടുക്കാം. ഹരിക്ക് പ്രായം 15 വയസും 10 മാസവും. ജോണിന് 15 വയസും 2 മാസവും (അതുകൊണ്ടു തന്നെ മാത്സ് ബ്ളോഗിനു പിന്നിലെ ഹരി, ജോൺ സാറന്മാരല്ല ഇവരെന്ന് മനസ്സിലായല്ലോ.). നമ്മുടെ പ്രോഗ്രാമിന്റെ ചോദ്യത്തിന് രണ്ടുപേരും 15 എന്ന് ഉത്തരം കൊടുക്കും. രണ്ടുപേരും ജനിച്ചത് 1995-ൽ ആണെന്ന് പ്രോഗ്രാം പറയുകയും ചെയ്യും. ജോണിന്റെ കാര്യത്തിൽ ഇത് ശരിയാണ്. ഹരിയുടെ കാര്യത്തിൽ ശരിയല്ലതാനും! ഹരി ജനിച്ചത് 1994 സെപ്റ്റംബർ അവസാനം ആണ്.

ഈ കീടത്തെ എങ്ങനെ ശരിപ്പെടുത്താം?

ബഗ് ഉണ്ടെന്ന് മനസ്സിലായ സ്ഥിതിക്ക് അത് ഒഴിവാക്കുന്ന തരത്തിൽ പ്രോഗ്രാം തിരുത്തി എഴുതേണ്ടത് നമ്മുടെ കടമയാണ് (ഈ പ്രക്രിയയെ "കീടോച്ചാടനം" എന്നു വിളിക്കാമോ?). ഹരിയുടെ കാര്യത്തിൽ പ്രോഗ്രാമിൽനിന്ന് തെറ്റായ ഉത്തരം കിട്ടാൻ എന്താണ് കാരണം? പ്രോഗ്രാമിന് ശരിയായ ഉത്തരം കണ്ടുപിടിക്കാൻ ആവശ്യമായ വിവരങ്ങൾ എല്ലാം ഇൻപുട്ട് ആയി നാം കൊടുക്കാത്തതുതന്നെ. വയസിന്റെ കറച്ചുകൂടി കൃത്യമായ വിവരം — എത്ര വർഷം എന്നതിന് പുറമേ എത്ര മാസം എന്നതുകൂടി — ഇൻപുട്ട് ആയി പ്രോഗ്രാമിന് കിട്ടിയാൽ മേൽപ്പറഞ്ഞ തെറ്റ് ഒഴിവാക്കുന്ന പ്രോഗ്രാം നമുക്കെഴുതാം. ഇങ്ങനെയുള്ള ഒരു പ്രോഗ്രാമിന്റെ ലോജിക് ഏകദേശം ഇതുപോലെയിരിക്കും:

1. ഉപയോക്താവിന്റെ പ്രായം എത്ര വർഷം, എത്ര മാസം എന്നത് ചോദിച്ച് മനസ്സിലാക്കുക.
    - വർഷത്തിനെ `years` എന്ന ചരത്തിൽ സൂക്ഷിച്ചു വെയ്ക്കുക.
    - മാസത്തിനെ `months` എന്ന ചരത്തിൽ സൂക്ഷിച്ചു വെയ്ക്കുക.
2. നടപ്പുവർഷവും മാസവും `current_year` , `current_month` എന്ന ചരങ്ങളിൽ സൂക്ഷിച്ചു വെയ്ക്കുക.
3. `months` എന്നതിന്റെ വില `current_month` എന്നതിന്റെ വിലയേക്കാൾ കുറവാണെങ്കിൽ
    - `current_year - years` എന്നതാണ് ഉപയോക്താവ് ജനിച്ച വർഷം; ഈ വിലയെ `year_of_birth` എന്ന ചരത്തിൽ സൂക്ഷിച്ചു വെയ്ക്കുക.
4. മറിച്ച്, `months` എന്നതിന്റെ വില `current_month` എന്നതിന്റെ വിലയ്ക്ക് തുല്യമോ അതിൽ കൂടുതലോ ആണെങ്കിൽ
    - `current_year - years - 1` എന്നതാണ് ഉപയോക്താവ് ജനിച്ച വർഷം; ഈ വിലയെ `year_of_birth` എന്ന ചരത്തിൽ സൂക്ഷിച്ചു വെയ്ക്കുക.
5. `year_of_birth` എന്ന ചരത്തിന്റെ വിലയാണ് ഉപയോക്താവ് ജനിച്ച വർഷം എന്ന് ഉത്തരം പറയുക.

പ്രവർത്തനങ്ങൾ

പ്രവ. **1.**

ഈ ലോജിക് ശ്രദ്ധിച്ച് വായിക്കുക. ഇത് ഹരി ജനിച്ച വർഷം ശരിയായി കണ്ടുപിടിക്കുമോ എന്ന് ചിന്തിക്കുക.

പ്രവ. **2.**

ഇപ്പറഞ്ഞ ലോജിക് എങ്ങനെ ഒരു പ്രോഗ്രാം ആയി എഴുതാം എന്ന് ആലോചിക്കുക.

if ... else

മുകളിലുള്ള ലോജിക്കിന് ആകെ അഞ്ചു ഭാഗങ്ങൾ ഉള്ളതിൽ ആദ്യത്തെ രണ്ടു ഭാഗങ്ങൾ പ്രോഗ്രാമിൽ എഴുതേണ്ടതെങ്ങനെ എന്ന് നമുക്കറിയാം – മുമ്പു കണ്ട പ്രോഗ്രാമുകളിൽ നാം ഇതേപോലെയുള്ള കാര്യങ്ങൾ ചെയ്തിട്ടുണ്ട്. ലോജിക്കിന്റെ അഞ്ചാമത്തെ ഭാഗവും — കണ്ടുപിടിച്ച ഉത്തരം ഉപയോക്താവിന് പറഞ്ഞുകൊടുക്കുക എന്നുള്ളതും — ഇപ്പോൾ നമുക്ക് നിഷ്പ്രയാസം ചെയ്യാൻ അറിയാം.

പ്രശ്നക്കാരായിട്ട് ഉള്ളത് മൂന്നും നാലും ഭാഗങ്ങളാണ്. ഒരു പ്രത്യേക കാര്യം — `months` എന്നതിന്റെ വില `current_month` എന്നതിന്റെ വിലയേക്കാൾ കുറവാണോ എന്നത് — ശരിയാണോ എന്ന് പരിശോധിച്ച്, ഇത് ശരിയാണെങ്കിൽ ഒരു കാര്യവും, തെറ്റാണെങ്കിൽ വേറൊരു കാര്യവും ചെയ്യുക എന്നതാണ് ഈ രണ്ട് ഭാഗങ്ങളിൽ സംഭവിക്കുന്നത്. ഇത് എങ്ങനെയാണ് പ്രോഗ്രാമിൽ എഴുതുക എന്ന് നാം ഇതുവരെയുള്ള പാഠങ്ങളിൽ പഠിച്ചിട്ടില്ല. ഇക്കാര്യം ഈ പാഠത്തിൽ പഠിക്കാം.

നാം ഇതുവരെ കണ്ട പൈത്തൺ പ്രോഗ്രാമുകളെല്ലാം ഒരു നേർരേഖാ സ്വഭാവം ഉള്ളവയാണ്. പ്രോഗ്രാമിന്റെ പ്രവർത്തനം തുടങ്ങിയാൽപ്പിന്നെ പ്രോഗ്രാമിലെ ഓരോ വരിയും പ്രോഗ്രാമിൽ എഴുതിയിരിക്കുന്ന ക്രമത്തിൽ പ്രവർത്തിച്ചുപോകുക എന്ന രീതിയാണ് അവയ്ക്കൊക്കെ ഉള്ളത്. ഇങ്ങനെ "നേർബുദ്ധി" മാത്രമുള്ള പ്രോഗ്രാമുകളെക്കൊണ്ട് വളരെ പരിമിതങ്ങളായ കാര്യങ്ങളേ ചെയ്യാൻ കഴിയൂ. ഇങ്ങനെയല്ലാത്ത — രേഖീയമല്ലാത്ത — രീതിയിൽ പ്രോഗ്രാം പ്രവർത്തിപ്പിക്കാനുള്ള പൈത്തൺ ഭാഷാശ്രുപങ്ങളിൽ ഏറ്റവും ലളിതമായതാണ് `if ... else` എന്ന പ്രയോഗം.
ഒരു വ്യവസ്ഥ (condition) ശരിയാണോ എന്ന് പരിശോധിച്ച്, ശരിയാണെങ്കിൽ ഒരു കൂട്ടം കാര്യങ്ങളും, തെറ്റാണെങ്കിൽ മറ്റൊരുകൂട്ടം കാര്യങ്ങളും ചെയ്യാൻ പൈത്തണിൽ (മറ്റു പ്രോഗ്രാമിംഗ് ഭാഷകളിലും) ഉള്ള ഉപാധിയാണ് `if ... else` എന്ന പ്രയോഗം. ഉദാഹരണമായി, `if ... else` ഉപയോഗിച്ച് ഒരു സംഖ്യ ഒറ്റസംഖ്യയോ ഇരട്ടസംഖ്യയോ എന്ന് കണ്ടുപിടിക്കുന്ന ഒരു പ്രോഗ്രാം ഇതാ. രണ്ടാം പാഠത്തിൽ നാം പരിചയപ്പെട്ട ശിഷ്ടം കാണാനുള്ള `%` എന്ന സംകാരകവും (operator) ഇവിടെ ഉപയോഗിച്ചിരിക്കുന്നു:

```
# This program finds if a given number is odd or even.  The
# program demonstrates the use of the "if ... else" program
# construct.  It is assumed that the input number is an
# integer!

number = input("Please input an integer: ")

if (number % 2) == 0 :
    print number, " is an even number"
else :
    print number, " is an odd number"
```

ഈ പ്രോഗ്രാമിൽ എട്ടു മുതൽ പതിനൊന്നു വരെയുള്ള വരികളിലാണ് `if ... else` ഉള്ളത്. ഈ വരികളിൽ എന്താണ് ചെയ്യുന്നതെന്ന് നമുക്ക് വിശദമായി നോക്കാം:

1. എട്ടും ഒൻപതും വരികളിൽ ചെയ്യുന്നത് ഇതാണ്: `number` എന്ന ചരത്തിൽ സൂക്ഷിച്ചിരിക്കുന്ന സംഖ്യയെ രണ്ടു കൊണ്ടു ഹരിച്ചാൽ ശിഷ്ടം പൂജ്യമാണെങ്കിൽ `number` ഇരട്ടസംഖ്യയാണെന്ന് പറയുക.
    - എട്ടാമത്തെ വരിയിൽ ശരിയാണോ എന്ന് പരിശോധിക്കേണ്ട വ്യവസ്ഥ `if` എന്നതിനും ശേഷവും `:` എന്നതിനും മുമ്പായും കൊടുത്തിരിക്കുന്നു.
        - `%` എന്ന ചിഹ്നം പൈത്തണിൽ ശിഷ്ടം കാണാനുള്ള സംകാരകത്തെ സൂചിപ്പിക്കുന്നു. `(number % 2)` എന്ന വ്യഞ്ജകത്തിന്റെ വില `number` -നെ രണ്ടുകൊണ്ടു ഹരിച്ചാൽ ശിഷ്ടം കിട്ടുന്ന സംഖ്യയാണ്. ഇത് പൂജ്യമോ ഒന്നോ ആയിരിക്കും (എന്തുകൊണ്ട്?).
        - `==` എന്ന ചിഹ്നം (രണ്ടു സമചിഹ്നങ്ങൾ അടുപ്പിച്ച് എഴുതിയത്) പൈത്തണിൽ തുല്യത പരിശോധിക്കുന്ന സംകാരകത്തെ സൂചിപ്പിക്കുന്നു.
        - എട്ടാമത്തെ വരി വായിക്കേണ്ടത് " number-നെ രണ്ടുകൊണ്ട് ഹരിച്ചാൽ ശിഷ്ടം പൂജ്യമാണെങ്കിൽ " എന്നാണ്.
    - വ്യവസ്ഥ ശരിയാണെങ്കിൽ ചെയ്യേണ്ട കാര്യങ്ങൾ — `number` ഇരട്ടസംഖ്യയാണെന്ന് പറയുക എന്നുള്ളത് — `if ... :` കഴിഞ്ഞുള്ള വരിയിൽ വലതുവശത്തേക്ക് കുറച്ച് മാറ്റി കൊടുത്തിരിക്കുന്നു.
2. പത്തും പതിനൊന്നും വരികളിൽ ചെയ്യുന്നത് ഇതാണ്. എട്ടാമത്തെ വരിയിൽ പരിശോധിച്ച വ്യവസ്ഥ — `number` എന്ന ചരത്തിൽ സൂക്ഷിച്ചിരിക്കുന്ന സംഖ്യയെ രണ്ടു കൊണ്ടു ഹരിച്ചാൽ ശിഷ്ടം പൂജ്യമാണോ എന്നത് — തെറ്റാണെങ്കിൽ `number` ഒറ്റസംഖ്യയാണെന്ന് പറയുക.
    - "എട്ടാമത്തെ വരിയിലെ `if` -ൽ പരിശോധിച്ച വ്യവസ്ഥ തെറ്റാണെങ്കിൽ" എന്ന് പ്രോഗ്രാമിൽ ചുരുക്കിപ്പറയാനായി `else :` എന്ന് പ്രയോഗിക്കുന്നു.
    - ഈ വ്യവസ്ഥ തെറ്റുമ്പോൾ ചെയ്യേണ്ട കാര്യങ്ങൾ — `number` ഒറ്റസംഖ്യയാണെന്ന് പറയുക എന്നുള്ളത് — `else :` കഴിഞ്ഞുള്ള വരിയിൽ വലതുവശത്തേക്ക് കുറച്ച് മാറ്റി കൊടുത്തിരിക്കുന്നു.

`if ... else` : വിശദമായി

`if ... else` പ്രോഗ്രാമിൽ ഉപയോഗിക്കുന്ന വിധം നമുക്ക് കുറച്ചുകൂടി വിശദമായി പരിശോധിക്കാം:

1. `if` വരിയുടെ വ്യാകരണം ഇതാണ്: `if condition :` . ഇവിടെ `condition` എന്നത് നമുക്ക് ശരിയോ തെറ്റോ എന്ന് അറിയേണ്ടതായ വ്യവസ്ഥ ആണ്. മുകളിലെ ഉദാഹരണത്തിൽ ഈ വ്യവസ്ഥ `(number % 2) == 0` എന്നതാണ്. `condition` കഴിഞ്ഞുള്ള `:` പ്രത്യേകം ശ്രദ്ധിക്കുക .
2. `if` വരി കഴിഞ്ഞുള്ള വരികളിൽ `condition` എന്ന വ്യവസ്ഥ ശരിയാണെങ്കിൽ ചെയ്യേണ്ടതായ കാര്യങ്ങൾ എഴുതണം. ഇത് ഒന്നിലധികം വരികളും ആവാം. ഇങ്ങനെയുള്ള വരികൾ എല്ലാം തന്നെ ഈ `if` വരിയെ അപേക്ഷിച്ച് ഒരു നിശ്ചിത അകലം വലതുവശത്തേക്ക് മാറി ആയിരിക്കണം തുടങ്ങേണ്ടത്.
3. മുകളിലെ ഉദാഹരണത്തിൽ നാലു സ്പേസ് വലത്തേക്ക് മാറിയാണ് എഴുതിയിട്ടുള്ളത് (ഒൻപതാമത്തെ വരി). ഇങ്ങനെ നാലു സ്പേസ് വിട്ടെഴുതുന്നതാണ് പൈത്തൺ മാനകം (standard).
4. IDLE ഉപയോഗിച്ച് പ്രോഗ്രാം എഴുതുകയാണെങ്കിൽ `if condition :` എന്നെഴുതി Enter അമർത്തുമ്പോൾ IDLE തനിയെ തന്നെ എഴുതിത്തുടങ്ങാനുള്ള സൂചകം (cursor) പുതിയ

വരിയിൽ നാലു സ്പേസ് വലത്തേക്ക് മാറ്റിത്തരുന്നത് കാണാം. ഇതൊന്ന് പരീക്ഷിച്ചു നോക്കൂ! ഇങ്ങനെ മാറ്റുന്നില്ലെങ്കിൽ തൊട്ടുമുമ്പത്തെ വരിയുടെ വ്യാകരണം തെറ്റിയതാവും കാരണം. മിക്കവാറും ഇത് അവസാനം കൊടുക്കേണ്ടതായ `:` വിട്ടുപോയതുകൊണ്ടാവും.

5. `if` -ൽ പരിശോധിച്ച വ്യവസ്ഥ ശരി ആണെങ്കിൽ ചെയ്യേണ്ടതായ കാര്യങ്ങൾ എഴുതിക്കഴിഞ്ഞാൽ പിന്നീടുള്ള വരി `if` വരി തുടങ്ങുന്ന അതേ അകലം ഇടതുവശത്തുനിന്ന് വിട്ട് വേണം തുടങ്ങാൻ. അതായത്, നാലു സ്പേസ് വലത്തേക്ക് മാറി എഴുതുന്നത് നിർത്തണം എന്നർത്ഥം. `if` വ്യവസ്ഥ ശരി ആണെങ്കിൽ ചെയ്യേണ്ടതായ കാര്യങ്ങൾ ഏതൊക്കെയാണെന്ന് പൈത്തൺ മനസ്സിലാക്കുന്നത് **`if`** -നു ശേഷം **`if`** -ന്റെ അതേ നിരപ്പിലുള്ള ആദ്യത്തെ വരി കാണുന്നതിന് മുൻപായി നാലു സ്പേസ് വലത്തേക്ക് മാറി വരുന്ന വരികൾ ഏതൊക്കെയാണ് എന്ന് നോക്കിയിട്ടാണ്.

6. മുകളിലെ ഉദാഹരണത്തിൽ `if` വ്യവസ്ഥ ശരി ആണെങ്കിൽ ചെയ്യേണ്ടതായ ഒരേ ഒരു കാര്യമേ ഉള്ളൂ — `number` ഇരട്ടസംഖ്യയാണെന്ന് പറയുക എന്നത്. ഈ ഒരു വരി മാത്രം (ഒൻപതാമത്തെ വരി) അതുകൊണ്ട് നാലു സ്പേസ് വലത്തേക്ക് തള്ളി എഴുതിയിരിക്കുന്നു.

7. `else` വരിയുടെ വ്യാകരണം ഇതാണ്: `else : `. `else` കഴിഞ്ഞുള്ള `:` പ്രത്യേകം ശ്രദ്ധിക്കുക .

8. `else` -ന്റെ അതേ നിരപ്പിൽ എഴുതിയിട്ടുള്ള, തൊട്ടു മുൻപുള്ള `if` വരിയുടെ വ്യവസ്ഥ തെറ്റാണെങ്കിൽ ചെയ്യേണ്ടതായ കാര്യങ്ങളാണ് `else` വരി കഴിഞ്ഞ് എഴുതേണ്ടത്. ഇത് ഒന്നിലധികം വരികളും ആവാം. ഇങ്ങനെയുള്ള വരികൾ എല്ലാം തന്നെ ഈ `else` വരിയെ അപേക്ഷിച്ച് ഒരു നിശ്ചിത അകലം വലതുവശത്തേക്ക് മാറി ആയിരിക്കണം തുടങ്ങേണ്ടത്.

9. മുകളിലെ ഉദാഹരണത്തിൽ പൈത്തൺ മാനകത്തിനനുസരിച്ച് നാലു സ്പേസ് വലത്തേക്ക് മാറിയാണ് എഴുതിയിട്ടുള്ളത് (പതിനൊന്നാമത്തെ വരി).

10. IDLE ഉപയോഗിച്ച് പ്രോഗ്രാം എഴുതുകയാണെങ്കിൽ `else :` എന്നെഴുതി Enter അമർത്തുമ്പോൾ IDLE തനിയെ തന്നെ എഴുതിത്തുടങ്ങാനുള്ള സൂചകം (cursor) പുതിയ വരിയിൽ നാലു സ്പേസ് വലത്തേക്ക് മാറ്റിത്തരുന്നത് കാണാം. പരീക്ഷിച്ചു നോക്കുക. ഇങ്ങനെ മാറുന്നില്ലെങ്കിൽ തൊട്ടുമുമ്പത്തെ വരിയുടെ വ്യാകരണം തെറ്റിയതാവും കാരണം. മിക്കപ്പോഴും ഇത് അവസാനം കൊടുക്കേണ്ടതായ `:` വിട്ടുപോയതുകൊണ്ടാവും.

11. `if` -ൽ പരിശോധിച്ച വ്യവസ്ഥ തെറ്റാണെങ്കിൽ ചെയ്യേണ്ടതായ കാര്യങ്ങൾ എഴുതിക്കഴിഞ്ഞാൽ പിന്നീടുള്ള വരി `if` വരി തുടങ്ങുന്ന അതേ അകലം ഇടതുവശത്തുനിന്ന് വിട്ട് വേണം തുടങ്ങാൻ. അതായത്, നാലു സ്പേസ് വലതേക്ക് മാറി എഴുതുന്നത് നിർത്തണം എന്നർത്ഥം. `if` വ്യവസ്ഥ തെറ്റാണെങ്കിൽ ചെയ്യേണ്ടതായ കാര്യങ്ങൾ ഏതൊക്കെയാണെന്ന് പൈത്തൺ മനസ്സിലാക്കുന്നത് **`else`** -നു ശേഷവും **`else`** -ന്റെ അതേ നിരപ്പിലുള്ള ആദ്യത്തെ വരി കാണുന്നതിന് മുൻപും നാലു സ്പേസ് വലത്തേക്ക് മാറി വരുന്ന വരികൾ ഏതൊക്കെയാണ് എന്ന് നോക്കിയിട്ടാണ്.

12. മുകളിലെ ഉദാഹരണത്തിൽ `if` വ്യവസ്ഥ തെറ്റാണെങ്കിൽ ചെയ്യേണ്ടതായ ഒരേ ഒരു കാര്യമേ ഉള്ളൂ — `number` ഒറ്റസംഖ്യയാണെന്ന് പറയുക എന്നത്. ഈ ഒരു വരി മാത്രം (പതിനൊന്നാമത്തെ വരി) അതുകൊണ്ട് നാലു സ്പേസ് വലത്തേക്ക് തള്ളി എഴുതിയിരിക്കുന്നു.

13. ഇവിടെ നാലു സ്പേസ് എന്ന് പറഞ്ഞയിടത്തൊക്കെ അതിനു പകരം വേറെ ഏതെങ്കിലും ഒരു നിശ്ചിത അകലം ഇതേ ആവശ്യത്തിന് ഉപയോഗിക്കാം. ഉദാഹരണത്തിന്, ഒരു ടാബ് (കംപ്യൂട്ടറിന്റെ Tab കീ അമർത്തിയാൽ കിട്ടുന്നത്) ഇതിനായി ഉപയോഗിക്കാം. ഒരേ പ്രോഗ്രാമിൽ ടാബുകളും സ്പേസുകളും രണ്ടുംകൂടി ഈ ആവശ്യത്തിന് ഉപയോഗിക്കരുത്. ഈ ആവശ്യത്തിന് നാലു സ്പേസ് ഉപയോഗിക്കുന്നതാണ് നല്ല പൈത്തൺ ശൈലിയായി കണക്കാക്കുന്നത്.

14. `if ... else` -ലെ `else` ആവശ്യമില്ലെങ്കിൽ വിട്ടുകളയാവുന്നതാണ് (optional). (തികച്ചും കൃത്രിമമായ ഒരു) ഉദാഹരണമായി, തന്നിരിക്കുന്ന സംഖ്യ ഇരട്ടസംഖ്യയാണെങ്കിൽ അത് വിളിച്ചുപറയുകയും ഒറ്റസംഖ്യയാണെങ്കിൽ ഒന്നും പറയാതിരിക്കുകയും ചെയ്യുന്ന ഒരു പ്രോഗ്രാം ഇതാ:

```
# When the given number is even, this program reports that
# fact. Otherwise it does not report anything. Used to
# demonstrate an "if" without a corresponding "else" part.
# It is assumed that the input number is an integer!

number = input("Please input an integer: ")

if (number % 2) == 0 :
    print number, " is an even number"
```

ഒറ്റയും ഇരട്ടയും സംഖ്യകൾ ഈ പ്രോഗ്രാമിന് ഇൻപുട്ട് ആയി കൊടുക്കുമ്പോൾ എന്താണ് സംഭവിക്കുന്നത് എന്ന് പ്രവർത്തിപ്പിച്ചു കണ്ടുപിടിക്കൂ.

തുലന സംകാരകങ്ങൾ

ഒരു സംഖ്യ ഒറ്റയോ ഇരട്ടയോ എന്ന് കണ്ടുപിടിക്കാനുള്ള പ്രോഗ്രാമിൽ നാം `==` എന്ന തുല്യത പരിശോധിക്കാനുള്ള സംകാരകം (operator) ഉപയോഗിച്ചല്ലോ. `==` -നു പുറമേ മറ്റ് തുലനസംകാരകങ്ങളും (comparison operators) പൈത്തണിൽ ലഭ്യമാണ്. ഈ സംകാരകങ്ങൾ എല്ലാംതന്നെ `True, False` എന്നീ രണ്ടു പ്രത്യേക മൂല്യങ്ങൾ ഉത്തരമായി തരുന്നവയാണ്. പേര് സൂചിപ്പിക്കുന്നതുപോലെ തന്നെ ശരി, തെറ്റ് എന്നീ അർത്ഥങ്ങളാണ് യഥാക്രമം `True, False` എന്ന മൂല്യങ്ങൾക്ക് പൈത്തണിൽ ഉള്ളത്. `==` എന്ന സംകാരകം നമുക്ക് ഒറ്റനോട്ടത്തിൽ തോന്നുന്നതുപോലെ തന്നെയാണ് പെരുമാറുന്നതും: `a, b` എന്നിവ സ്ഥിരാങ്കങ്ങളോ (constants) ചരങ്ങളോ ആയിക്കൊള്ളട്ടെ. `a == b` എന്നതിന്റെ വില

- `a, b` എന്നിവയുടെ വിലകൾ രണ്ടും തുല്യമാണെങ്കിൽ `True` ആയിരിക്കും.
- `a, b` എന്നിവയുടെ വിലകൾ രണ്ടും തുല്യമല്ലെങ്കിൽ `False` ആയിരിക്കും.

പൈത്തൺ പ്രോഗ്രാമുകളിൽ ധാരാളമായി ഉപയോഗിക്കുന്ന ചില തുലനസംകാരകങ്ങളുടെ പ്രയോഗം വ്യക്തമാക്കുന്ന പട്ടിക. ഇവിടെ `a, b` എന്നിവ സ്ഥിരാങ്കങ്ങളോ (constants) ചരങ്ങളോ (variables) ആകാം:

		രണ്ടാം കോളത്തിലെ വ്യഞ്ജകത്തിന്റെ വില

തുലന സംകാരകം	പ്രയോഗം	True ആകുന്നത്	False ആകുന്നത്
==	a == b	a, b എന്നിവയുടെ വിലകൾ തുല്യമാണെങ്കിൽ	a, b എന്നിവയുടെ വിലകൾ തുല്യമല്ലെങ്കിൽ
!=	a != b	a, b എന്നിവയുടെ വിലകൾ തുല്യമല്ലെങ്കിൽ	a, b എന്നിവയുടെ വിലകൾ തുല്യമാണെങ്കിൽ
<	a < b	a -യുടെ വില b -യുടെ വിലയെക്കാൾ കുറവാണെങ്കിൽ	a -യുടെ വില b -യുടെ വിലയെക്കാൾ കുറവല്ലെങ്കിൽ
>	a > b	a -യുടെ വില b -യുടെ വിലയെക്കാൾ കൂടുതലാണെങ്കിൽ	a -യുടെ വില b -യുടെ വിലയെക്കാൾ കൂടുതലല്ലെങ്കിൽ
<=	a <= b	a -യുടെ വില b -യുടെ വിലയ്ക്ക് തുല്യമോ അതിൽ കുറവോ ആണെങ്കിൽ	a -യുടെ വില b -യുടെ വിലയെക്കാൾ കൂടുതലാണെങ്കിൽ
>=	a >= b	a -യുടെ വില b -യുടെ വിലയ്ക്ക് തുല്യമോ അതിൽ കൂടുതലോ ആണെങ്കിൽ	a -യുടെ വില b -യുടെ വിലയെക്കാൾ കുറവാണെങ്കിൽ

പൈത്തണിലെ ചില തുലനസംകാരകങ്ങൾ

== എന്ന സംകാരകത്തെ പ്രോഗ്രാമിൽ കാണുമ്പോൾ ഇംഗ്ലീഷിൽ വായിക്കുന്നത് "equals" അല്ലെങ്കിൽ "equal to" എന്നാണ്. != -നെ വായിക്കുന്നത് "not equal to" എന്നും. ഇങ്ങനെയോ തത്തുല്യങ്ങളായ മലയാള വാക്കുകളായോ (ഏതാണ് നമുക്ക് കൂടുതൽ സ്വാഭാവികമായി തോന്നുന്നത് എന്നതനുസരിച്ച്) ഈ ചിഹ്നങ്ങളെ വായിക്കാം.

മുമ്പ പറഞ്ഞതുപോലെ, "നേർരേഖാ" പ്രോഗ്രാമുകളെക്കൊണ്ട് കുറച്ച കാര്യങ്ങളേ ചെയ്യാൻ കഴിയൂ. മറ്റ പ്രോഗ്രാമുകളിലൊക്കെ ഒന്നല്ലെങ്കിൽ മറ്റൊരു രീതിയിൽ തുലനം (comparison) കടന്നു വരുമെന്ന് ഉറപ്പിക്കാം. അതുകൊണ്ട തന്നെ ഇനി നാം എഴുതുന്ന പ്രോഗ്രാമുകളിലെല്ലാം ഒരു തുലനസംകാരകമെങ്കിലും ഉപയോഗിക്കേണ്ടി വരും എന്ന് നിസ്സംശയം പറയാം. തുലനസംകാരകങ്ങൾ ശരിയായി പ്രയോഗിക്കേണ്ടതെങ്ങനെ എന്ന് അനവധി ഉദാഹരണങ്ങളിലൂടെ നമുക്ക് പഠിച്ചെടുക്കാം.

ഒരു സംഖ്യ ന്യൂനസംഖ്യയാണോ, അധിസംഖ്യയാണോ, ഇനി ഇതു രണ്ടുമല്ല പൂജ്യമാണോ എന്ന് കണ്ടുപിടിക്കുന്ന ഒരു പ്രോഗ്രാം നമുക്കൊന്ന് എഴുതി നോക്കാം:

```
1  # This program tells if a given number is negative, zero, or
2  # positive. The program demonstrates the use of the "if
3  # ... else" program construct.
4
5  number = input("Please input a number: ")
6
7  if number < 0 :
8      print number, " is a negative number"
9  else :
10     if number == 0 :
11         print number, " is zero"
12     else :
13         print number, " is a positive number"
```

ഈ പ്രോഗ്രാമിൽ പല അളവിൽ സ്പേസ് വിട്ട് വരികൾ എഴുതിയിരിക്കുന്നത് ശ്രദ്ധിക്കുക. മുകളിൽക്കൊടുത്ത വിശദീകരണം അനുസരിച്ച് ഇത് മനസ്സിലാക്കാൻ ശ്രമിക്കുക. സംശയമുണ്ടെങ്കിൽ ചോദിക്കുക.

ജനിച്ച വർഷം കണ്ടുപിടിക്കുന്നതെങ്ങനെ? രണ്ടാം പതിപ്പ്

ജോൺ, ഹരിമാരുടെ പ്രശ്നം പരിഹരിക്കുന്ന പ്രോഗ്രാം എഴുതാൻ വേണ്ടത്ര പൈത്തൺ നാം പഠിച്ചുകഴിഞ്ഞു. പാഠത്തിന്റെ മൂന്നാം ഭാഗത്ത് കൊടുത്ത ലോജിക് ഒരു പ്രോഗ്രാമാക്കി ഒന്ന് എഴുതി നോക്കിയാലോ? ലോജിക് വിശദമായിത്തന്നെ നാം എഴുതിയതുകൊണ്ട് ഇത് പ്രോഗ്രാമാക്കുക എന്നത് തികച്ചും അനായാസം തന്നെ. കുറച്ച വലിയ പ്രോഗ്രാമുകൾ എഴുതാനുള്ള ഏറ്റവും എളുപ്പ വഴിയും ഇതു തന്നെയാണ്. ആദ്യം പ്രോഗ്രാമിന്റെ ലോജിക് നമുക്കിഷ്ടമുള്ള ഭാഷയിൽ എഴുതുക. പിന്നെ അതിന്റെ ഓരോ ഭാഗവും പൈത്തണിലേക്ക് മൊഴിമാറ്റം നടത്തുക. ഒരു വലിയ പ്രോഗ്രാം ഒറ്റയടിക്ക് പൈത്തണിൽ എഴുതുന്നതിലും എളുപ്പം — തെറ്റുകൾ കുറയ്ക്കാനുള്ള എളുപ്പ വഴിയും — ഇങ്ങനെ ചെയ്യുന്നതാണ്.

```
1  # This program asks for the name and age in years and months
2  # of the user, and outputs the year when the user was born.
3
4  name = raw_input("Hello, what is your name? ")
5
6  print name, ", please tell me your age in years and months."
7
8  years = input("How many  years? ")
9  months = input("And how many months? ")
10
11
12 current_year = 2010
13 current_month = 8
14
15 if months < current_month :
16     year_of_birth = current_year - age
17 else :
18     year_of_birth = current_year - age - 1
19
```

```
print name, ", you were born in ", year_of_birth
print "Goodbye!"
```

പ്രവർത്തനങ്ങൾ

പ്രവ. **3.**

ഈ പ്രോഗ്രാം ശ്രദ്ധിച്ച് വായിക്കുക. ഇത് പാഠത്തിന്റെ മൂന്നാം ഭാഗത്ത് കൊടുത്ത ലോജിക്കുമായി യോജിച്ചുപോകുന്നുണ്ടോ എന്ന് തിട്ടപ്പെടുത്തുക.

പ്രവ. **4.**

ഈ പ്രോഗ്രാം പല തവണ പ്രവർത്തിപ്പിക്കുക. പല (വർഷം, മാസം) ജോടികൾക്ക് ഇത് ശരിയായ ഉത്തരം തരുന്നുണ്ടോ എന്ന് പരിശോധിക്കുക.

പ്രവ. **5.**

ഈ പ്രോഗ്രാം എപ്പോഴാണ് തെറ്റായ ഉത്തരം തരുന്നത്? മുൻപുള്ള പ്രോഗ്രാമിനെ അപേക്ഷിച്ച് ഈ പ്രോഗ്രാം തെറ്റായ ഉത്തരം തരുന്ന അവസരങ്ങൾ കൂടുതലോ കുറവോ? ഈ തെറ്റിനെ ഒഴിവാക്കുന്ന രീതിയിൽ പ്രോഗ്രാം തിരുത്തിയെഴുതുക.

ബൂളിയൻ സംകാരകങ്ങൾ

ശരി, തെറ്റ് (`True, False`) എന്നീ രണ്ട് സ്ഥിരാങ്കങ്ങളെ ബൂളിയൻ മൂല്യങ്ങൾ (Boolean values)എന്ന് പറയുന്നു. ഈ രണ്ടു വിലകളെ അടിസ്ഥാനപ്പെടുത്തിയുള്ള ലോജിക് വികസിപ്പിച്ചെടുത്ത ഇംഗ്ലീഷ് ഗണിതശാസ്ത്രജ്ഞനായ ജോർജ് ബൂളിന്റെ ബഹുമാനാർത്ഥമാണ് ഇത്. ഇവന സംകാരകങ്ങൾ പ്രയോഗിക്കുമ്പോൾ കിട്ടുന്നത് ബൂളിയൻ മൂല്യങ്ങളാണല്ലോ. ഇത്തരത്തിലുള്ള മൂല്യങ്ങളെ സംയോജിപ്പിക്കാൻ ഉപയോഗിക്കുന്ന സംകാരകങ്ങളാണ് ബൂളിയൻ സംകാരകങ്ങൾ (Boolean operators). പേരുകേട്ടാൽ ഭയം തോന്നുമെങ്കിലും ഇവ വളരെ ലളിതങ്ങളായ സംഭവങ്ങളാണ്; ഈ പേരിലല്ലെങ്കിലും ഇവയെ നമുക്കെല്ലാവർക്കും കുട്ടിക്കാലം മുതൽക്കേ പരിചയമുള്ളതാണെന്നതാണ്!

പൈത്തൺ പ്രോഗ്രാമുകളിൽ ധാരാളമായി ഉപയോഗിക്കുന്ന ചില ബൂളിയൻ സംകാരകങ്ങളുടെ പ്രയോഗം വ്യക്തമാക്കുന്ന പട്ടിക. ഇവിടെ `a, b` എന്നിവ `True, False` എന്നീ വിലകളുള്ള സ്ഥിരാങ്കങ്ങളോ ചരങ്ങളോ ആകാം:

ബൂളിയൻ സംകാരകം	പ്രയോഗം	രണ്ടാം കോളത്തിലെ വ്യഞ്ജകത്തിന്റെ വില	
		True ആകുന്നത്	**False** ആകുന്നത്
not	`not a`	`a` യുടെ വില `False` ആണെങ്കിൽ	`a` യുടെ വില `True` ആണെങ്കിൽ
and	`a and b`	`a, b` രണ്ടിന്റെയും വിലകൾ `True` ആണെങ്കിൽ	`a, b` എന്നിവയിൽ ഒന്നിന്റെയെങ്കിലും വില `False` ആണെങ്കിൽ
or	`a or b`	`a, b` എന്നിവയിൽ ഒന്നിന്റെയെങ്കിലും വില `True` ആണെങ്കിൽ	`a, b` രണ്ടിന്റെയും വിലകൾ `False` ആണെങ്കിൽ

പൈത്തണിലെ ചില ബൂളിയൻ സംകാരകങ്ങൾ

ബൂളിയൻ സംകാരകങ്ങളെക്കുറിച്ച് പ്രത്യേകം ശ്രദ്ധിക്കേണ്ടതായ ഒരു കാര്യം: മുകളിലെ പട്ടിക കണ്ടാൽ തോന്നുന്നതിന്റെ പത്തിലൊന്ന് കാഠിന്യം ഇവ പ്രോഗ്രാമിൽ ഉപയോഗിക്കുമ്പോൾ തോന്നുകയില്ല. and, or, not എന്നീ വാക്കുകൾ ഇംഗ്ലീഷിൽ ഉപയോഗിക്കുന്നതുപോലെ തന്നെ പ്രോഗ്രാമിലും രണ്ടാമതൊന്ന് ആലോചിക്കാതെ ഉപയോഗിച്ചു പോവുകയാണ് പതിവ്. കുറച്ച് ഉദാഹരണങ്ങൾ കൊണ്ട് ഇത് വ്യക്തമാകും. എപ്പോഴെങ്കിലും സംശയം വന്നാൽ ഈ പട്ടിക നോക്കുകയും ചെയ്യാം.

തന്നിരിക്കുന്ന സംഖ്യ 2, 3, 5 ഇവ മൂന്നിന്റേയും ഗുണിതമാണോ എന്ന് പരിശോധിക്കുന്ന ഒരു പ്രോഗ്രാം എഴുതി നോക്കാം:

```
# This program tells if a given number is a multiple of 2,
# 3, and 5. The program demonstrates the use of the "if
# ... else" program construct.

number = input("Please input a number: ")

if (number % 2) == 0 :
    if (number % 3) == 0 :
        if (number % 5) == 0 :
            print number, " is a multiple of 2, 3, and 5"
        else :
            print number, " is not a multiple of 2, 3, and 5"
    else :
        print number, " is not a multiple of 2, 3, and 5"
else :
    print number, " is not a multiple of 2, 3, and 5"
```

പ്രവർത്തനം

പ്രവ. **6.**

ഈ പ്രോഗ്രാം ശ്രദ്ധിച്ച് വായിക്കുക. ഇവിടെ സ്പേസ് ഉപയോഗിച്ചിരിക്കുന്ന രീതി ശരിക്ക് മനസ്സിലായി എന്ന് ഉറപ്പുവരുത്തുക. ഓരോ `if` -ന്റെയും `else` ഏതാണ്? ഈ പ്രോഗ്രാമിന്റെ ലോജിക് എന്താണ്?

ഇതേ പ്രശ്നത്തിന് ഉത്തരം തരുന്ന മറ്റൊരു പ്രോഗ്രാം നോക്കൂ. പല പ്രാവശ്യം "if ..." എന്നെഴുന്നതിനു പകരം ഇവിടെ "and" ഉപയോഗിച്ചിരിക്കുന്നു. ഫലമോ? മുമ്പത്തെ പ്രോഗ്രാമിന്റെയത്ര ദൈർഘ്യമില്ലാത്തതും വായിച്ചു മനസ്സിലാക്കാൻ അത്രതന്നെ പ്രയാസപ്പെടേണ്ടാത്തതുമായ ഒരു പ്രോഗ്രാം. പ്രോഗ്രാമിലെ ഏഴാമത്തെ വരി അധികം വലത്തേക്ക് നീണ്ടു പോകാതിരിക്കാനായി (ബ്രൗസറിൽ വായിക്കാനുള്ള സൗകര്യത്തിനായി മാത്രം) "\" ഉപയോഗിച്ച് രണ്ടായി മുറിച്ചിരിക്കുന്നു:

```
1  # This program tells if a given number is a multiple of 2,
2  # 3, and 5. The program demonstrates the use of the "and"
3  # boolean operator.
4
5  number = input("Please input a number: ")
6
7  if (number % 2) == 0 and (number % 3) == 0 \
8          and (number % 5) == 0 :
9      print number, " is a multiple of 2, 3, and 5"
10 else :
11     print number, " is not a multiple of 2, 3, and 5"
```

പ്രവർത്തനങ്ങൾ

പ്രവ. **7.**

തന്നിരിക്കുന്ന സംഖ്യ 2, 3, 5 ഇവയിൽ ഒന്നിന്റെയെങ്കിലും ഗുണിതമാണോ എന്ന് പരിശോധിക്കുന്ന ഒരു പ്രോഗ്രാം എഴുതുക. ഇതിനായി "or" എന്ന ബൂളിയൻ സംകാരകം ഉപയോഗിക്കുക.

പ്രവ. **8.**

*ഏഴാം പ്രവർത്തനത്തിലെ അതേ പ്രശ്നത്തിനുള്ള പ്രോഗ്രാം **"or"** ഉപയോഗിക്കാതെ എഴുതുക.*

കൂടുതൽ പ്രവർത്തനങ്ങൾ

`if` -നെ പരിചയപ്പെട്ട നിലയ്ക്ക് ഇനി നമുക്ക കൂടുതൽ രസകരങ്ങളായ പ്രവർത്തനങ്ങൾ ചെയ്തു തുടങ്ങാം. പ്രോഗ്രാമിംഗ് പ്രവർത്തനങ്ങൾ ചെയ്യുമ്പോൾ പ്രത്യേകം ശ്രദ്ധിക്കേണ്ടതായ ഒരു കാര്യമുണ്ട്. വെറുതേ പ്രോഗ്രാം എഴുതിയതുകൊണ്ടു മാത്രം പ്രവർത്തനത്തിന്റെ ഫലം നമുക്ക് കിട്ടുന്നില്ല. എഴുതിയ പ്രോഗ്രാമിന് ശരിയായ എന്ത് ഇൻപുട്ട് കൊടുത്താലും അത് ശരിയായ ഉത്തരം തരും എന്ന് ഉറപ്പു വരുത്തണം. ഇതിനായി രണ്ടു കാര്യങ്ങൾ ചെയ്യണം:

1. പ്രോഗ്രാം വളരെ ശ്രദ്ധയോടെ, വിമർശനബുദ്ധ്യാ വായിച്ചുനോക്കണം. നാമെഴുതിയ പ്രോഗ്രാം എന്തുകൊണ്ട് ശരിയാണ് എന്നല്ല നോക്കേണ്ടത്, മറിച്ച് അതിലെ തെറ്റ് എവിടെയാണ് എന്നാണ്. ഇങ്ങനെ കണിശതയോടെ സ്വന്തം പ്രോഗ്രാം വായിക്കാൻ പഠിക്കുക എന്നത് തെറ്റില്ലാതെ പ്രോഗ്രാം എഴുതാൻ ഒഴിച്ചുകൂടാനാവാത്ത ഒന്നാണ്.
2. പ്രോഗ്രാമിന് പല തരത്തിലുള്ള ഇൻപുട്ടുകൾ കൊടുത്ത് പ്രവർത്തിപ്പിച്ചുനോക്കണം. നാം തന്നെ എഴുതിയ പ്രോഗ്രാം ആയതുകൊണ്ട് അത് ശരിയായ ഉത്തരം തരുന്ന മട്ടിലുള്ള ഇൻപുട്ടുകൾ കൊടുക്കാനാണ് നമുക്ക് സ്വതവേ തോന്നുക. എന്നാൽ ഇങ്ങനെയല്ലാത്ത, പ്രോഗ്രാം തെറ്റിപ്പോകാൻ സാധ്യതയുള്ള തരം ഇൻപുട്ടുകളും ആലോചിച്ചെടുത്ത് അവ പ്രോഗ്രാമിന് കൊടുത്തുനോക്കണം.

ഇതു വരെ പഠിച്ച പൈത്തൺ ഉപയോഗിച്ച് ചെയ്യാവുന്ന കുറച്ച് പ്രവർത്തനങ്ങൾ ഇതാ:

പ്രവർത്തനങ്ങൾ

താഴെപ്പറയുന്ന തരം പ്രോഗ്രാമുകൾ എഴുതുക:

പ്രവ. **9.**

- ഇൻപുട്ട്: ഒരു കുട്ടിയുടെ ഒരു വിഷയത്തിലുള്ള മാർക്ക്.
- ഔട്പുട്ട്: ആ കുട്ടിക്ക് ആ വിഷയത്തിൽ കിട്ടിയ ഗ്രേഡ്.

മാർക്കിനെ ഗ്രേഡാക്കി മാറ്റാൻ നിങ്ങളുടെ സ്കൂളിൽ (അല്ലെങ്കിൽ പരിചയത്തിലെ) ഉപയോഗിക്കുന്ന ഏതെങ്കിലും ഒരു രീതി തെരഞ്ഞെടുക്കുക.

പ്രവ. **10.**

- ഇൻപുട്ട്: ഈ വർഷത്തിലെ ഒരു ദിവസം — വർഷം, മാസം, തീയതി എന്ന രീതിയിൽ.
- ഔട്പുട്ട്: തന്നിരിക്കുന്ന ദിവസത്തിന്റെ അടുത്ത ദിവസം — വർഷം, മാസം, തീയതി എന്ന രീതിയിൽ.

പ്രവ. **11.**

- ഇൻപുട്ട്: ഒരു ദിവസം — വർഷം, മാസം, തീയതി എന്ന രീതിയിൽ.
- ഔട്പുട്ട്: തന്നിരിക്കുന്ന ദിവസത്തിന്റെ അടുത്ത ദിവസം — വർഷം, മാസം, തീയതി എന്ന രീതിയിൽ.

പ്രവ. **12.**

- ഇൻപുട്ട്: മൂന്നു സംഖ്യകൾ.

- ഔട്പുട്ട്: തന്നിരിക്കുന്നതിൽവച്ച് ഏറ്റവും വലിയ സംഖ്യ.

പ്രവ. **12.**

- ഇൻപുട്ട്: മൂന്നു സംഖ്യകൾ.
- ഔട്പുട്ട്: തന്നിരിക്കുന്നതിൽവച്ച് ഏറ്റവും ചെറിയ സംഖ്യ.

Posted by ഫിലിപ്പ് at 5:00 AM

Home Older Post

Subscribe to: Post Comments (Atom)