அகர முதலே Python

1. பைதானில் எண்கள் (Numeric Data Types)

கணக்கீடு, கணிதம், வாழ்க்கை: எண்களின் ஆளுமை

அன்புள்ள எதிர்கால விஞ்ஞானிகளே,

ஒரு வினாடி யோசித்துப் பாருங்கள்… நம்மைச் சுற்றியிருக்கும் அத்தனையும்—நேரம், பணம், தொழில் நுட்பம், அறிவியல்—இவை அனைத்தும் எண்களின் அடிப்படையில் தான் இயங்குகின்றன, இல்லையா? இந்தக் கேள்வி உங்கள் மனதிலும் நிச்சயம் எழுந்திருக்கும்.

“காலை 6 கிலோமீட்டர் நடைபயிற்சி… மாலை 6 கிலோமீட்டர் நடைபயிற்சி… இது ஆரோக்கியத்திற்கு நல்லதா?”

“இந்த மாதம் எத்தனை ரூபாய் சேமித்தேன்? அல்லது எத்தனை ரூபாய் செலவழித்தேன்?”

“என் கைப்பேசிக்கு ஒரு நிமிடத்தில் 10 மெசேஜ்கள் வந்துவிட்டன, என்ன பதில் சொல்வது?”

“இந்த ஆடைக்கு 20% தள்ளுபடி கிடைக்குமா?”

இவை அனைத்தும் எண்களின் மொழியில் பேசுகின்றன. நம்மை அறியாமலேயே, நாம் எப்போதும் எண்களுடன்தான் வாழ்கிறோம். கணிதப் பாடத்தில் 1+1=2 என்று கற்றுக்கொண்ட முதல் நாளிலிருந்து, எண்கள் நம் வாழ்வின் அடிப்படைக் கட்டமைப்பாகவே இருக்கின்றன. ஒரு சாதாரணக் கடைக்காரரின் பணக் கணக்கீடு முதல், உலகமே வியக்கும் அணு ஆராய்ச்சி வரை - எல்லாமே எண்களின் ஆதாரத்தில்தான் நிற்கின்றன. எண்கள் என்பவை வெறும் குறியீடுகள் அல்ல; அவை தகவல்கள், அளவீடுகள், உறவுகள் மற்றும் முடிவுகள்.

எண்களின் பிறப்பு: மனித நாகரிகத்தின் முதல் மைல்கல்

மனித நாகரிகத்தின் தொடக்கத்திலிருந்தே எண்களின் தேவை உணரப்பட்டது. பழங்கால மனிதன் வேட்டையாடிய விலங்குகளின் எண்ணிக்கையைக் கணக்கிட, தனது மந்தையில் உள்ள கால்நடைகளைக் கண்காணிக்க, விளைச்சலை அளக்க எனப் பல தேவைகளுக்காக எண்களைப் பயன்படுத்தத் தொடங்கினான். ஆரம்பத்தில் கற்கள், குச்சிகள், விரல்கள் போன்றவற்றைப் பயன்படுத்தி எண்ணிய மனிதன், படிப்படியாக கீறல்கள் (tallies), களிமண் பலகைகள் ஆகியவற்றில் குறியீடுகளைப் பொறித்து எண்களைப் பதிவு செய்ய ஆரம்பித்தான்.

மெசபடோமியா, எகிப்து, சிந்து சமவெளி போன்ற பண்டைய நாகரிகங்களில் எண்கணிதத்தின் ஆரம்ப வடிவங்கள் செழித்து வளர்ந்தன. குறிப்பாக, இந்தியக் கணிதவியலாளர்கள் சுழியம் (Zero) மற்றும் இட மதிப்பு (Place Value System) என்ற புரட்சிகரமான கருத்துக்களைக் கண்டறிந்தனர். இந்த கண்டுபிடிப்புகள் தான் இன்று நாம் பயன்படுத்தும் தசம எண் முறைக்கு (Decimal System) அடிப்படையாக அமைந்தன. இம்முறை, உலகின் ஒவ்வொரு மூலைக்கும் பரவி, நவீன அறிவியல் மற்றும் தொழில்நுட்ப வளர்ச்சிக்கு வழி வகுத்தது.

கணினியின் DNA: எண்களின் புரிதல்

சரி, எண்களுக்கும் கணினி நிரலாக்கத்திற்கும் என்ன தொடர்பு? எப்படி ஒரு வீடு உறுதியாக நிற்க ஒரு வலுவான அஸ்திவாரம் (foundation) தேவையோ, அப்படியே ஒரு மென்பொருளுக்கும் எண்களின் ஆழமான புரிதல் தேவைப்படுகிறது. எண்கள் என்பது ஒரு மென்பொருளின் அடிப்படை ‘DNA’ அல்லது ‘கட்டுமானத் தொகுதிகள்’ (building blocks).

மனிதர்கள் தசம எண் முறையைப் பயன்படுத்துகையில், கணினிகள் இரண்டும எண் முறை (Binary System) என்ற விசித்திரமான மொழியில் பேசுகின்றன. 0 மற்றும் 1 ஆகிய இரண்டு எண்களை மட்டுமே பயன்படுத்தி கணினிகள் அனைத்து தகவல்களையும் சேமிக்கின்றன, செயலாக்குகின்றன. ஒரு கணினியில் நீங்கள் ஒரு திரைப்படத்தைப் பார்த்தாலோ, ஒரு மின்னஞ்சலை அனுப்பினாலோ, அல்லது ஒரு விளையாட்டை விளையாடினாலோ, பின்னணியில் நடக்கும் அத்தனை செயல்பாடுகளும் இந்த 0 மற்றும் 1 என்ற இரும எண்களின் தொடர் வரிசையில்தான் நிகழ்கின்றன.

இது மின்சாரம் "ஆன்" (1) மற்றும் "ஆஃப்" (0) என்ற இரண்டு நிலைகளில் இருப்பதன் அடிப்படையிலான ஒரு எளிய தர்க்கம்.

நீங்கள் ஒரு பெரிய நிறுவனத்திற்கான Billing System (பணம் செலுத்தும் அமைப்பு) உருவாக்குகிறீர்கள் என்று எடுத்துக்கொள்ளுங்கள். வாடிக்கையாளரின் பில் தொகையைக் கணக்கிடுவது, GST-ஐச் சேர்ப்பது, தள்ளுபடிகளைக் கழிப்பது, இறுதித் தொகையை 'rounding' செய்வது – இந்த எல்லாச் செயல்பாடுகளும் எண்களை வைத்துதான் நடக்கின்றன. இல்லையென்றால், உங்கள் கைப்பேசியில் நீங்கள் பயன்படுத்தும் ஒரு சாதாரண Calculator App-ஐ பைதான் மூலம் எழுதுகிறீர்கள் என்று வைத்துக்கொள்வோம். அதில் தசம எண்கள் (decimal), முழு எண்கள் (integer), சதவீதங்கள் (percentage) ஆகியவற்றைச் சரியாகப் பிரித்துப் புரிந்துகொள்ளாமல் எப்படி அந்தச் செயலியை வடிவமைக்க முடியும்? தவறான கணக்கீடு, பெரிய நிதி இழப்புக்கோ, அல்லது தவறான முடிவுகளுக்கோ இட்டுச் செல்லலாம்.

எண்கள் பற்றிய தெளிவான புரிதல், எந்தவொரு நிரலாக்க மொழிக்கும், அதன்மூலம் நாம் உருவாக்கும் மென்பொருளுக்கும் இன்றியமையாதது. ஒரு மென்பொருளின் வெற்றிக்கு, அதன் செயல்பாடுகளைப் போலவே, அதன் உள்ளீட்டு எண்களையும், வெளியீட்டு எண்களையும் சரியாகக் கையாள்வது அத்தியாவசியம். இந்த அடிப்படைப் புரிதல் இல்லாமல், ஒரு மென்பொருளை உருவாக்குவது சாத்தியமில்லை.

பைதானின் எண் வகைகள்: ஒரு மாயாஜால ரகசியம்

நம்மில் பலர் நிரலாக்கத்தைக் கற்றுக்கொள்வதற்கு முன்பு, "எண்கள் என்றால் என்ன பெரிய விஷயம்? 1, 2, 3... அவ்வளவுதானே?" என நினைக்கிறோம். ஆனால், பைதான் போன்ற மொழிகளில், ஒவ்வொரு எண்ணுக்கும் ஒரு குறிப்பிட்ட வகை (data type) உண்டு. இந்த வகைகள், கணினி அந்த எண்ணை எப்படிச் சேமிக்க வேண்டும், எப்படிச் செயலாக்க வேண்டும் என்பதை வரையறுக்கின்றன. எண்களைப் பற்றி நாம் நினைப்பதை விடவும், பைதான் அதை மிகவும் நுட்பமாகப் புரிந்துகொள்கிறது.

பைதானில் எண்கள் பொதுவாக `int` (முழு எண்கள்), `float` (தசம எண்கள்), `complex` (கலப்பு எண்கள்) என்ற மூன்று முக்கிய வகைகளாகப் பிரிக்கப்படுகின்றன. ஒவ்வொன்றும் தனித்தன்மை வாய்ந்தது; ஒவ்வொன்றுக்கும் அதற்கேயுரிய பயன்கள் உண்டு.

நீங்கள் யாராக இருந்தாலும் சரி, இந்த மூன்று எண் வகைகளும் உங்கள் பைதான் பயணத்தில் உங்கள் கூடவே வரும். ஒரு நிரலாளர் சிறந்த சிக்கல் தீர்ப்பவராக (problem solver) ஆக வேண்டுமென்றால், இந்த அடிப்படைப் புரிதல் (foundation) மிக அவசியம். இந்த முதல் அத்தியாயத்தில் நாம் அந்த அடித்தளத்தைத்தான் கற்கப்போகிறோம். எண்களின் உலகத்திற்குள் உங்களை வரவேற்கிறோம்! இந்தக் கணினி உலகில் எண்கள் எப்படி வெவ்வேறு வடிவங்களில் இயங்குகின்றன என்பதைப் புரிந்துகொள்ள நீங்கள் தயாரா?

1.1. `int` – முழு எண்கள் (Whole Numbers)

`int` என்று சொன்னாலே நமக்கு சட்டென்று நினைவுக்கு வருபவை 1, 2, -10, 0, 2000 போன்ற முழு எண்கள்தான். அதாவது, தசமப் புள்ளி இல்லாத, நேர்மறை அல்லது எதிர்மறை எண்கள், பூஜ்ஜியம் உட்பட. நம்முடைய வயதைக் குறிக்க (`age = 32`), ஒரு இடத்தின் வெப்பநிலையைக் குறிக்க (`temperature = -5`), ஒரு குறிப்பிட்ட பொருளின் எண்ணிக்கையைக் குறிக்க (`count = 100`), அல்லது ஒரு வருடத்தைக் குறிக்க (`year = 2025`) போன்றவற்றை பைதான் **`int`** வகையைக் கொண்டு சேமிக்கிறது.

Python

```
# வயது - ஒரு நேர்மறை முழு எண்
my_age = 32

# வெப்பநிலை - ஒரு எதிர்மறை முழு எண்
outside_temperature = -5

# ஒரு நகரத்தின் மக்கள் தொகை - ஒரு பெரிய முழு எண்
city_population = 1400000000

print(f"My age: {my_age}")
print(f"Outside temperature: {outside_temperature}°C" )
print(f"City population: {city_population}")
```

பைதானின் அசாத்திய int சக்தி: வரம்பில்லா எண்கள் (Unbounded Integers)

இங்குதான் பைதான் மற்ற பல பாரம்பரிய நிரலாக்க மொழிகளில் இருந்து தனித்து நிற்கிறது! பொதுவாக C, Java போன்ற சில நிரலாக்க மொழிகளில், `int` வகை எண்களுக்கு ஒரு குறிப்பிட்ட வரம்பு உண்டு. உதாரணமாக, ஒரு `int` அதிகபட்சமாக 2 பில்லியனுக்கு (2,000,000,000) சற்றே அதிகமான எண்ணை மட்டுமே சேமிக்க முடியும். அதற்கு மேல் சென்றால், அந்த எண்ணைச் சேமிக்க வேறு சில சிறப்பு வகைகளைப் பயன்படுத்த வேண்டும். இது ஆரம்பக்கட்ட நிரலாளர்களுக்கு சற்று குழப்பமாக இருக்கலாம்.

ஆனால், **பைதான் `int` வகையில் வரம்பில்லா (unbounded) எண்களைச் சேமிக்க முடியும்.** அதாவது, ஒரு `int`-க்கு பைதானில் எந்த வரம்பும் (limit) இல்லை! ஒருசிறு குழந்தையிடம், “வானத்தில் எத்தனை நட்சத்திரங்கள் இருக்கின்றன?” என்று கேட்டால், அது எண்ணி முடிக்க முடியாமல் திணறும். ஆனால், பைதான் அப்படியல்ல! பிரபஞ்சத்தின் அணுக்களின் எண்ணிக்கையைக் கணக்கிட்டாலும் சரி, கோடிக்கணக்கான பரிவர்த்தனைகளைக் கையாண்டாலும் சரி, பைதான் `int` வகையைப் பொறுத்தவரை, உங்கள் கணினியில் **நினைவகம் (memory)** இருக்கிற வரைக்கும் எந்தப் பெரிய எண்ணையும் அது கையாளும்! இது பைதான் மொழி வழங்கும் ஒரு அபாரமான சுதந்திரம். மிகமிகப் பெரிய எண்களைக் கணக்கிட வேண்டிய அறிவியல் ஆய்வுகள், நிதி சார்ந்த மென்பொருட்கள் போன்றவற்றுக்கு இது மிகவும் பயனுள்ளதாக இருக்கும்.

இதோ ஒரு வியக்கத்தக்க உதாரணம், இது பல நிரலாக்க மொழிகளில் சாத்தியமற்றது:

Python

```
# 999-ஐ 999 முறை தன்னால் பெருக்கினால் என்னவாகும்?
# இது ஒரு கற்பனைக்கு எட்டாத அசுரப் பெரிய எண்ணை உருவாக்கும்!
print(999**999)
```

இந்தக் கணக்கீடு, எழுத்தில் அடங்காத ஒரு அசுரப் பெரிய எண்ணை (monster number) அச்சிடும்! ஒரு பாரம்பரிய Programming Language-ல் (எ.கா: C++, Java) இதைச் செய்ய முயற்சித்தால், உடனடியாக ‘overflow error’ வந்துவிடும் (அதாவது, ‘இவ்வளவு பெரிய எண்ணைச் சேமிக்க என்னிடம் இடமில்லை, என் வரம்பு மீறிவிட்டது’ என்று கணினி சொல்லிவிடும்). ஆனால் பைதானில், உங்கள் கணினியில் நினைவகம் இருக்கிற வரைக்கும் `int` எவ்வளவு பெரிய எண்ணையும் விரிவாகச் சேமித்து, கணக்கீடு செய்யும்! இது பைதானின் ஒரு மறைக்கப்பட்ட சூப்பர் பவர்.

பைதானின் எண் ரகசியங்கள்: கணினியின் மொழியை அறிதல்

நம்முடைய அன்றாட உலகில், நாம் அனைவரும் இயல்பாகப் பயன்படுத்தும் எண்ணிக்கை முறை — அதாவது 1, 2, 3...9, 10, 11... — இவை அனைத்தும் **தசம முறை (decimal system)**, அல்லது **`base-10`** என்றே அடிப்படையாகக் கொண்டு அமைகின்றன. ஏன் `base-10`? நம்மிடம் 10 விரல்கள் இருப்பதாலா? ஆம், சில வரலாற்றாசிரியர்களின் கூற்றுப்படி, மனிதர்கள் தசம முறையை வரலாற்றில் ஏற்றுக்கொண்டது, “கணக்கிடுவதற்கு விரல்களைப் பயன்படுத்தியதால்” தான்.

இது ஒரு எளிய மற்றும் உள்ளுணர்வு சார்ந்த முறை.

ஆனால் கணினிகள் அப்படியில்லை! நாம் எப்படி மனிதர்கள் என்றால் தசம முறையை இயல்பாக நினைக்கிறோமோ, கணினிகளுக்கு அது **இருநிலை எண் முறை (binary system)**, அதாவது `base-2` தான். ஒரு கணினி என்பது **வன்பொருள் (hardware)** + **மின்சாரம் (electricity)** + **லாஜிக் கேட்ஸ் (logic gates)** ஆகியவற்றின் தொகுப்பு. இதில் ஓர் அளவில் எல்லாமே "ON or OFF" (மின்சாரம் பாய்கிறதா இல்லையா), "1 or 0" என்ற இருநிலை மொழியிலேயே இயங்குகிறது.

இங்கே 10 இலக்கங்கள் (digits) இல்லை. இரண்டு மட்டுமே: **0 மற்றும் 1**.

அதனால்தான்:

- கணினிக்கு நேரடி தசம (`base-10`) எண் முறை இயல்பாகப் புரிவதில்லை.
- அது எல்லாவற்றையும் இருநிலை (`base-2`) ஆகவோ, அல்லது இருநிலையுடன் தொடர்புடைய **எண்ம (`base-8` / Octal), பதினாறும (`base-16` / Hexadecimal)** ஆகிய மாறான `base`-களில் **மாற்றி (convert)** தான் உபயோகிக்க வேண்டும்.

பைதான் இந்த எல்லா எண் முறைகளையும் **இயல்பாகவே (Built-in-ஆ)** ஆதரிக்கிறது! நாம் பைதானில் எந்த `base`-ல் ஒரு முழு எண்ணைப் (integer) பிரதிநிதித்துவப்படுத்த வேண்டும் என்று சொல்ல, பைதானுக்கு ஒரு சிறப்புச் **"சேர்ப்பின் (prefix)"** மூலம் தெரிவிக்கலாம். இது ஒரு மொழிபெயர்ப்பாளரைப் போல, கணினியின் மொழிக்கும் நம் மொழிக்கும் பாலமாகச் செயல்படுகிறது.

கீழே உள்ள அட்டவணையில், வெவ்வேறு `base`-களில் எண்களை எப்படிப் பைதானில் எழுதுவது என்றும், அவை தசம முறையில் என்ன மதிப்பைக் குறிக்கின்றன என்றும் விளக்கப்பட்டுள்ளது:

Format	Prefix	Example	Output (Decimal Value)	விளக்கம்
Binary (`base 2`)	`0b`	`0b1010`	10	(`1*2^3`) + (`0*2^2`) + (`1*2^1`) + (`0*2^0`) = 8 + 0 + 2 + 0 = **10**. `0b` என்பது இரும எண்ணைக் குறிக்கிறது.
Octal (`base 8`)	`0o`	`0o12`	10	(`1*8^1`) + (`2*8^0`) = 8 + 2 = **10**. `0o` என்பது எண்ம எண்ணைக் குறிக்கிறது.
Hex (`base 16`)	`0x`	`0xA`	10	`A` என்பது 10-ஐக் குறிக்கிறது. (`10*16^0`) = **10**. `0x` என்பது பதினாறும எண்ணைக் குறிக்கிறது.

இந்தச் சேர்ப்புகளைப் பயன்படுத்தி, நாம் பைதானில் எண்களைப் பயன்படுத்தும் எடுத்துக்காட்டுகளைக் கீழே காணலாம். இந்த குறியீடுகளை நீங்கள் உங்கள் பைதான் நிரலில் பயன்படுத்தும்போது, பைதான் தானாகவே அவற்றை தசம மதிப்புகளாகப் புரிந்துகொள்ளும்.

Python

```
print(0b1010)  # இது ஒரு இரும எண் (Binary). பைதான் இதை 10 என அச்சிடும்.
print(0o12)    # இது ஒரு எண்ம எண் (Octal). பைதான் இதை 10 என அச்சிடும்.
print(0xA)     # இது ஒரு பதினாறும எண் (Hexadecimal). பைதான் இதை 10 என அச்சிடும்.
```

இப்போது உங்களுக்குப் புரிந்திருக்கும். நாம் எழுதுவது ஒரு எண் என்றாலும், அது எந்த முறையில் எழுதப்பட்டுள்ளது என்பதைப் பொறுத்து கணினி அதை எப்படிப் புரிந்துகொள்கிறது என்பதில் இந்த `prefix`-கள் மிக முக்கியப் பங்காற்றுகின்றன. இது பைதான் நமக்கு அளிக்கும் ஒரு சக்திவாய்ந்த வசதி!

நீங்கள் ஒரு ஆரம்ப நிலை நிரலாளராக இருந்தால், இதைப் பார்த்து, “நமக்கு ஏன் இதெல்லாம்? அன்றாட நிரலாக்கத்திற்கு இது எதற்கு?” என்று தோன்றலாம். ஆனால், நீங்கள் ஒரு உண்மையான நிரலாளராக மாறும் போது, இந்தப் புரிதல் உங்கள் திறமையை ஒரு புதிய நிலைக்கு உயர்த்தும். கணினிகள் இயங்கும் அடிப்படையைப் புரிந்துகொள்வதற்கு இந்த எண் முறைகள் மிகவும் அவசியம். சில குறிப்பிட்ட துறைகளில் அவை எப்படிப் பயன்படுகின்றன என்பதைப் பார்ப்போம்:

- **இருநிலை (`Binary`):** நீங்கள் ஒரு மென்பொருளின் ஆழமான ரகசியங்களை, கணினியின் நினைவக வடிவமைப்பு, அல்லது ஒரு சாதனத்தின் மிக அடிப்படையான செயல்பாடுகளைக் கையாளும்போது (`embedded systems`) இருநிலை மொழி மிக அவசியம். உதாரணமாக, ஒரு சிறு **LED விளக்கு** ‘ஆன்’ அல்லது ‘ஆஃப்’ ஆவதைக் கட்டுப்படுத்துகிறீர்கள் என்று வைத்துக்கொள்வோம். `1` என்பது விளக்கை ஆன் செய்யவும், `0` என்பது ஆஃப் செய்யவும் பயன்படும். இப்படி, மிக எளிய ‘ஆம்’ அல்லது ‘இல்லை’ முடிவுகளைக் கையாள இருநிலை எண்கள் உதவுகின்றன.
- **எண்ம (`Octal`): லினக்ஸ்/யுனிக்ஸ் (Linux/Unix)** போன்ற இயக்க முறைமைகளில் கோப்புகளுக்கு அனுமதி கொடுக்கும் மாயாஜாலம் (`chmod 755 = rwxr-xr-x`) எண்ம மூலமே எளிதாகச் செயல்படுகிறது. ஒரு கோப்பை **யார் படிக்கலாம் (read), எழுதலாம் (write), இயக்கலாம் (execute)** போன்ற அனுமதிகளை இந்த எண்ம மதிப்புகள் மூலம் எளிதாகக் குறிப்பிட முடியும். உங்கள் வீட்டில் ஒரு அறைக்கு யார் நுழையலாம், யார் பொருட்களைப் பயன்படுத்தலாம் என்று அனுமதி கொடுப்பதைப் போல, கணினி கோப்புகளுக்கும் இந்த எண்ம எண்கள் மூலம் அனுமதி கொடுக்கப்படுகிறது.
- **பதினாறும (`Hexadecimal`):** இது நிரலாக்க உலகில் மிகவும் பரவலாகப் பயன்படுத்தப்படும் ஒன்று. உங்கள் வெப் பக்கத்திற்குப் பிடித்த நிறத்தைக் கொடுக்கும் மாயாஜாலம் (`#FF5733` என்பது ஒரு பதினாறும குறியீடு!), கணினியின் நினைவக முகவரிகள், மென்பொருள் பிழைக் குறியீடுகள் (`error codes`), வன்பொருள் தொடர்பு (`hardware communication`) எனப் பல இடங்களில் பதினாறும முறை மிக முக்கியம்.

இதோ ஒரு கற்பனை:

நீங்கள் ஒரு **`web designer`** ஆக இருக்கிறீர்கள். ஒரு வெப்சைட் உருவாக்கும்போது, ஒரு குறிப்பிட்ட பட்டன் அல்லது பின்னணிக்கு அழகான **சிவப்பு நிறம்** கொடுக்க விரும்புகிறீர்கள். அந்தச் சிவப்பு நிறத்தை நீங்கள் **`#FF0000`** என்று குறிப்பிடுவீர்கள். இங்கு **`FF0000`** என்பது ஒரு **பதினாறும எண்**. இது **சிவப்பு (FF), பச்சை (00), நீலம் (00)** ஆகிய வண்ணங்களின் கலவையை (RGB) பதினாறும வடிவில் குறிக்கிறது. இப்படி எண்ணற்ற நிறங்களை இந்த பதினாறும எண்கள் மூலம் துல்லியமாகக் குறிப்பிடலாம்.

`Hexadecimal` கொஞ்சம் ஸ்டைலாக இருக்கும்! (ஏனென்றால், 0-9 வரைக்கும் வழக்கமான இலக்கங்கள், 10-15-க்கு `A, B, C, D, E, F` என ஆங்கில எழுத்துகள் பயன்படுத்தப்படுகின்றன!)

கீழே பதினாறும எண்களைப் பைதான் எப்படிப் புரிந்துகொள்கிறது என்பதற்கான எடுத்துக்காட்டுகள்:

Python

```python
print(0xF)   # Output: 15 (F என்பது தசமத்தில் 15-ஐக் குறிக்கிறது)
print(0x1F)  # Output: 31  → இது எப்படி 31 ஆனது?
             # (1 × 16^1) + (F × 16^0) = (1 × 16) + (15 × 1) = 16 + 15 = 31
```

இதை நீங்கள் ஒவ்வொரு நாளும் நேரடியாகப் பயன்படுத்த மாட்டீர்கள். ஆனால், ஒரு நாள் **நிறங்களை வடிவமைக்கும்போது, நினைவகப் பகுதிகளைக் கையாளும்போது**, அல்லது ஒரு **பிழைக் குறியீட்டைப் புரிந்துகொள்ளும்போது**, இந்த `base` சிஸ்டம்கள் உங்கள் வாழ்க்கையை எளிதாக்கும். பைதான் இதை நீங்களே கையாளத் தெரிந்துகொள்ளும்படி வடிவமைத்திருக்கிறது. இது உங்கள் திறமையை அடுத்த கட்டத்திற்கு உயர்த்தும்!

1.2. float – தசம எண்கள் (Numbers with Decimal Points)

நம்முடைய வாழ்க்கையில் **தசம மதிப்புகள் (decimal values)** இல்லாமல் ஒரு நாளும் செல்ல முடியாது. பணப் பரிவர்த்தனைகள், அறிவியல் அளவீடுகள், அன்றாட சராசரிகள் – இந்த உலகில் தசம எண்கள் இல்லாமல் ஒரு நொடியும் நகர முடியாது.

“ஒரு பவுன் தங்கம் இன்று எவ்வளவு? (₹53,425.50)”, “வெப்பநிலை இன்று 36.6°C இருக்கு”, “பெட்ரோல் ₹103.50 ஆகிவிட்டது!” — தசம எண்கள் இல்லாமல் இவை அனைத்தும் அரைச்சொல்லாகத் தோன்றும். இதுபோல தசம எண்கள் நம்முடைய நாள் தோறும் உரையாடல்களில் அடிக்கடி வந்துகொண்டே இருக்கின்றன.

பைதானில் தசம எண்களை `float` **வகை (type)**-ல் பிரதிநிதித்துவப்படுத்துகிறோம்.

Python

```
# ஒருவரின் உயரம்
my_height = 5.8  # 5 அடி 8 இன்ச்

# ஒருவரின் எடை
my_weight = 72.3  # 72.3 கிலோகிராம்

# கணித மாறிலி Pi
pi_value = 3.14159

# ஒரு பங்கின் விலை
stock_price = 1234.56

print(f"My height: {my_height} feet")
print(f"My weight: {my_weight} kg")
print(f"Value of Pi: {pi_value}")
print(f"Current stock price: ${stock_price}")
```

இவை எல்லாம் `float`, அதாவது **`floating-point numbers`** (மிதக்கும் புள்ளி எண்கள்). ஆனால் இங்கு ஒரு சுவாரசியமான சிக்கல் உள்ளது, இது புதிதாகக் கற்கும் பலரைக் குழப்பக்கூடும்:

Python

```
print(0.1 + 0.2)
```

Output:

```
0.30000000000000004
```

ஏன் இப்படியொரு திடுக்கிடும் ஒரு விடையை பைதான் தருகிறது? `0.1 + 0.2` என்றால் `0.3` தானே?

நீங்கள் நிச்சயம் குழப்பமடைந்திருக்கலாம்! `0.1` மற்றும் `0.2`-ஐக் கூட்டினால், கணினி ஏன் `0.30000000000000004` என்று காட்டுகிறது?

உண்மை என்னவென்றால், பைதானில் `float` என்பது கணினியின் வன்பொருளில் உள்ள **இருநிலை மிதக்கும் புள்ளி (binary floating-point)** அமைப்பை அடிப்படையாகக் கொண்டது. நம் கணினியின் மூளை, தசம எண்களை நேரடியாக, நாம் புரிந்து கொள்ளும் விதத்தில் புரிந்துகொள்வதில்லை. அது எல்லாவற்றையும் `0` மற்றும் `1` என இருநிலையாகவே (இரண்டு இலக்க முறையில்) நினைக்கும். `0.1`, `0.2` போன்ற சில தசம எண்களை (எல்லா தசம எண்களையும் அல்ல) இருநிலையில் சரியாக, துல்லியமாகப் பிரதிபலிக்க முடியாது.

இது ஒரு `1/3` என்பதை நாம் `0.333333333333` என முடிவில்லாமல் எழுதுவது போல. கணினிக்கு ஒரு குறிப்பிட்ட (வரையறுக்கப்பட்ட) இடமே இருப்பதால், அது ஒரு கட்டத்தில் அந்த எண்ணை **'சுருக்கி' (truncate)** அல்லது **'வட்டமிட்டு' (round)** சேமிக்க வேண்டியிருக்கிறது. அதனால்தான் இந்தக் கணக்கீட்டில் ஒரு சிறு **'சறுக்கல்' (rounding error)** ஏற்படுகிறது. இது பைதானின் பிரச்சனை அல்ல; இது **கணினியின் வன்பொருளின் அடிப்படை வரம்பு**.

தீர்வு: `decimal` module – கண்டிப்பான கணக்காளர்!

நாள் தோறும் தசம எண்கள் உபயோகிக்கப்படுகின்றன — பொருளாதாரம், வெப்பநிலை, எடை, விலை, அறிவியல் அளவீடுகள் ஆகிய அனைத்திலும். நீங்கள் ஒரு **கணக்காளர் (accountant)**, **செயலி உருவாக்குநர் (app developer)** அல்லது **விஞ்ஞானி (scientist)** என்றால், தசம எண்களை அதிகக் கண்டிப்புடன், மிகச் சரியாகக் கணக்கிட வேண்டி வரும். பணக் கணக்குகள், அறிவியல் ஆய்வுகள் போன்ற இடங்களில் இந்தச் சிறு சறுக்கல்கள் பெரிய ஆபத்தை விளைவிக்கலாம்!

அதனால்தான், பைதான் ஒரு **'கண்டிப்பான கணக்காளரை'** உங்களுக்கு அறிமுகப்படுத்துகிறது: அதுதான் **`decimal` module**. இது தசம எண்களை இருநிலையாக மாற்றாமல், நேரடியாகவே துல்லியமாகக் கணக்கிடுகிறது.

Python

```python
from decimal import Decimal

# Decimal() பயன்படுத்தும் போது, எண்களை quotation marks ('') உள்ளிடவும்
print(Decimal('0.1') + Decimal('0.2'))  # Output: 0.3
print(Decimal('1.05') * Decimal('3.20')) # Output: 3.3600
```

`float`-ல் வரும் சிறிய தவறுகளைத் தவிர்க்க, இந்த `Decimal` Module மிகவும் பயனுள்ளதாக இருக்கும். குறிப்பாக, நிதி சார்ந்த பயன்பாடுகளுக்கு இது அத்தியாவசியமானது. உங்கள் கணக்கீடுகள் துல்லியமாக இருக்க வேண்டும் என்று நீங்கள் விரும்பினால், `decimal` module உங்கள் சிறந்த நண்பன்!

1.3. `complex` – கலப்பு எண்கள் (Real + Imaginary Numbers)

"எண்கள்" என்றால் என்ன நினைப்போம்?

₹10, -5 டிகிரி செல்சியஸ், 0, 3.14 (pi), 100 — இவை எல்லாம் நமக்குத் தெரிந்த மிகவும் சாதாரணமான எண்கள். இவைகளை எல்லாம் **'மெய் எண்கள்' (real numbers)** என்பார்கள். நம் அன்றாட வாழ்வின் பெரும்பாலான கணக்கீடுகளுக்கு இந்த மெய் எண்களே போதுமானவை.

ஆனால் சில சமயங்களில், குறிப்பாகக் கணிதம், இயற்பியல், பொறியியல் (எலெக்ட்ரிக் சர்க்யூட்ஸ், சிக்னல் புராசஸிங், அலைகள், குவாண்டம் மெக்கானிக்ஸ்) போன்ற ஆழ்ந்த துறைகளில், நாம் அறிந்த மெய் எண்களுக்கு அப்பால் ஒரு புதிய வகை எண்கள் தேவைப்படும்.

ஒரு எளிய உதாரணம்:

ஒரு எண்ணை அதே எண்ணால் பெருக்கும்போது, விடை எதிர்மறையாக வர முடியுமா?

`x * x = -1` என்ற ஒரு கணக்கை நீங்கள் தீர்க்க முயற்சித்தால் என்ன செய்வீர்கள்? எந்த மெய் எண்ணை அதனுடனேயே பெருக்கினாலும், நேர்மறை எண்ணாகத்தான் வரும் (உதாரணமாக, `2 * 2 = 4`, `-2 * -2 = 4`). ஆனால், இந்தச் சமன்பாட்டைத் தீர்க்க, `√-1` என்ற ஒரு எண்ணைக் கண்டுபிடிக்க வேண்டியிருக்கிறது. இது மெய் எண்களின் உலகத்தில் சாத்தியமில்லை!

இதற்குத்தான் கணிதத்தில் ஒரு புதிய வகை எண் கண்டுபிடிக்கப்பட்டது: அதுதான் **'கற்பனை எண்' (Imaginary number)**. இதை `i` **(ஐ)** என்று குறிப்பார்கள். அதாவது, `i = √-1`.

இந்தக் கற்பனை எண்களும், மெய் எண்களும் இணைந்த கலவையே **'கலப்பு எண்கள்' (Complex Numbers)** எனப்படுகின்றன.

கற்பனை எண்கள் ஏன் தேவை? ஒரு சிறிய விளக்கம்:

கற்பனை எண்கள் ஏன் உருவாக்கப்பட்டன என்பதைப் புரிந்துகொள்ள, ஒரு **திசைகாட்டி (compass)** போல சிந்திக்கலாம். சாதாரண எண்கள் (மெய் எண்கள்) ஒரு நேர் கோட்டில், அதாவது கிழக்கு-மேற்கு திசையில் உள்ள தூரத்தைக் குறிக்கின்றன. ஆனால் சில சமயங்களில், நமக்கு வட-தெற்கு திசையிலும் ஒரு அளவீடு தேவைப்படும். இந்த `i` அல்லது `j` என்பது, அந்த நேர் கோட்டிற்குச் செங்குத்தாக (90 டிகிரி) ஒரு புதிய திசையைக் குறிக்க உதவுகிறது.

உதாரணமாக, **மின்சார சுற்றுக்களில் (electric circuits)**, மின்சாரம் பாயும்போது அதன் அளவு, அது பாயும் திசை ஆகிய இரண்டு தகவல்களும் முக்கியம். இந்த இரண்டு தகவல்களையும் ஒரே எண்ணில் குறிக்க இந்த `complex` எண்கள் உதவுகின்றன. `complex` எண்கள், மின் பொறியியலாளர்களுக்கு மின்னோட்டம் மற்றும் மின்னழுத்தத்தின் **'கட்டம்' (phase)** போன்றவற்றை விவரிக்க மிகவும் உதவுகின்றன.

பைதானில் இந்தக் கற்பனை எண்ணை `i` என்பதற்குப் பதிலாக `j` என்று எழுதுகிறோம்.

Python

```
# ஒரு எலெக்ட்ரிக் சர்க்யூட்டில் உள்ள மின் எதிர்ப்பை (impedance) குறிக்கப் பயன்படுத்தலாம்
z = 3 + 4j # இது ஒரு கலப்பு எண் (complex number)

print(f"கலப்பு எண்: {z}")
```

இந்தக் கலப்பு எண்ணில்:

- `3` → `real part` **(மெய் பகுதி):** இது வழக்கமான எண், அதாவது ஒரு நேர் கோட்டில் உள்ள தூரம் போன்றது.
- `4j` → `imaginary part` **(கற்பனைப் பகுதி):** இது `j` உடன் வரும் பகுதி, அதாவது நேர் கோட்டிற்குச் செங்குத்தாக உள்ள தூரம் போன்றது.

நீங்கள் இப்படி ஒரு எண்ணைக் கொடுத்தால்:

Python

```
print(f"மெய் பகுதி: {z.real}")   # Output: 3.0
print(f"கற்பனைப் பகுதி: {z.imag}") # Output: 4.0
```

பைதான் இதில் வல்லுநர் (`expert`). இந்தக் கலப்பு எண்களை நேரடியாகவே உருவாக்கி, அவற்றின் மெய் மற்றும் கற்பனைப் பகுதிகளைப் பிரித்து, பகுப்பாய்வு (`analysis`) பண்ண முடியும். C, Java போன்ற பல நிரலாக்க மொழிகளில் இந்தக் `complex` எண்களைக் கையாள, நீங்கள் `external libraries` (வெளியில் இருந்து சிறப்பு மென்பொருள் தொகுப்புகளை) சேர்க்க வேண்டும், இது சிக்கலான `செட்டப்` தேவைப்படலாம். ஆனால், பைதான் இதை இயல்பாகவே, உங்கள் விரல் நுனியில் வைத்திருக்கிறது! நீங்கள் ஒருநாள் இந்த விசித்திர எண்களுடன் வேலை செய்ய நேரிட்டால், பைதானின் இந்தத் திறனை வியந்து பார்ப்பீர்கள்!

1.4. பொதுவான எண் கணிதச் செயல்பாடுகள் (Common Numeric Operations)

எண்களுடன் வேலை செய்ய, அடிப்படைச் செயல்பாடுகள் அவசியம். பைதான், கணிதச் செயல்பாடுகளை எளிதாக்குகிறது. இதை ஒரு சமையல்காரரின் அடிப்படை சமையல் கருவிகள் போலக் கற்பனை செய்து கொள்ளுங்கள். எப்படி ஒரு சமையல்காரருக்குக் கத்தி, பாத்திரம், அடுப்பு போன்ற அடிப்படை உபகரணங்கள் இல்லாமல் சமைக்க முடியாதோ, அதேபோல ஒரு நிரலாளருக்கு இந்த எண் கணிதச் செயல்பாடுகள் இல்லாமல் நிரல் எழுத முடியாது. இவைதான் பைதான் மொழியில் எண்களைக் கையாள உதவும் அன்றாடக் கருவிகள்.

கீழே உள்ள அட்டவணையில், இந்த அடிப்படைச் செயல்பாடுகள் என்ன, அவற்றை எப்படிப் பயன்படுத்துவது மற்றும் அவற்றின் வெளியீடு என்ன என்பதைக் காணலாம்:

செயல்	எடுத்துக்காட்டு	விளக்கம்
கூட்டல் `+`	`5 + 2.0`	`7.0` (இரண்டு எண்களையும் கூட்டுகிறது)
கழித்தல் `-`	`10 - 3`	`7` (ஒரு எண்ணிலிருந்து மற்றொன்றைக் கழிக்கிறது)
பெருக்கல் `*`	`4 * 3`	`12` (இரண்டு எண்களையும் பெருக்குகிறது)
வகுத்தல் `/`	`7 / 2`	`3.5` (எப்போதும் `float` வெளியீடாக வரும், அதாவது தசமப் பகுதியுடன்)
முழு எண் வகுத்தல் `//`	`7 // 2`	`3` (தசமப் பகுதி துண்டிக்கப்பட்டு, முழு எண்ணாக மட்டுமே வரும்)
மீதம் (`Modulus`) `%`	`7 % 2`	`1` (ஒரு எண்ணை மற்றொன்றால் வகுக்கும்போது கிடைக்கும் மீதம்)
அடுக்கு `**`	`2 ** 3`	`8` (2-ன் 3வது அடுக்கு: 222)

இந்த உதாரணத்தை நீங்கள் தவறவிடக் கூடாது; இது பைதானின் ஒரு புத்திசாலித்தனமான அம்சம்:

Python

```python
print(type(5 + 2.0))  # Output: <class 'float'>
```

பைதான் மிக புத்திசாலி! ஒரு கணக்கீட்டில் வெவ்வேறு வகை எண்கள் (`int` + `float`) இருக்கும்போது, அது துல்லியத்தை இழக்காமல் இருக்க, **'பெரிய வகை' (`float`)**-க்குத் தானாகவே மாற்றி, முடிவைத் தரும். இதை **`Type Promotion`** என்று அழைக்கிறார்கள். இது உங்கள் கணக்கீடுகளில் துல்லியத்தை உறுதி செய்யும் ஒரு பாதுகாப்பு அம்சம். இந்த அம்சம், நீங்கள் எதிர்பாராத பிழைகளைத் தவிர்க்க உதவுகிறது, குறிப்பாகக் கலவையான எண் வகைகளைக் கையாளும்போது. பைதான் எப்போதுமே துல்லியத்திற்கு முக்கியத்துவம் கொடுக்கும் என்பதை இது காட்டுகிறது.

1.5. தரவு வகை மாற்றம் (Data Type Conversion) – மாறுவது நம் தன்மை!

சில சமயங்களில், ஒரு தரவு வகையை மற்றொன்றாக மாற்ற வேண்டிய தேவை வரும். உதாரணமாக, பயனர் உள்ளிடும் தகவல்கள் பொதுவாக 'string' வடிவத்தில் இருக்கும். அவற்றை நீங்கள் கணக்கீடுகளுக்குப் பயன்படுத்த, எண் வகையாக மாற்ற வேண்டும். பைதான் அதற்கு எளிதான செயல்பாடுகளை வழங்குகிறது:

Python

```
age_text = "42"          # வயது ஒரு எழுத்து வடிவில் (string)
pi_text = "3.14"         # Pi ஒரு எழுத்து வடிவில்
score_number = 99        # மதிப்பெண் ஒரு எண்ணாக (int)

# String-ஐ முழு எண்ணாக மாற்றுதல்
print(f"String to int: {int(age_text)}")     # Output: 42

# String-ஐ தசம எண்ணாக மாற்றுதல்
print(f"String to float: {float(pi_text)}")  # Output: 3.14

# ஒரு எண்ணை எழுத்து வடிவமாக மாற்றுதல்
print(f"Int to string: {str(score_number)}") # Output: "99"
```

நினைவில் கொள்ளுங்கள்: உள்ளீடு (Input) எப்போதும் string ஆக வரும். நாம்தான் அதை சரியான வகைக்கு (appropriate type) மாற்ற வேண்டும்.

ஆனால் இங்கே ஒரு சிறிய நுணுக்கம் உள்ளது, இது புதியவர்களுக்குச் சற்றுக் குழப்பத்தை ஏற்படுத்தலாம்:

Python

```
# ஒரு தசம எண்ணை நேரடியாக முழு எண்ணாக மாற்ற முயற்சித்தால்
print(int("3.5"))  # ❌ Error: ValueError: invalid literal for int() with base 10: '3.5'
```

`int("3.5")` என்று நீங்கள் ஒரு string-ல் உள்ள தசம எண்ணை நேரடியாக முழு எண்ணாக மாற்ற முயற்சிக்கும்போது, பைதான் பிழை (Error) காட்டும். ஏனெனில் “3.5” என்பது ஒரு தசம வடிவில் உள்ள string. அதை நேரடையாக ஒரு முழு எண்ணாக மாற்ற முடியாது. பைதான் `int()` செயல்பாடு, “3” போன்ற **தூய முழு எண் string-ஐ** (pure integer string) மட்டுமே நேரடியாக மாற்றும்.

சரியான வழி:

Python

```
# முதலில் தசம எண்ணாக மாற்றி, பிறகு முழு எண்ணாக மாற்றுதல்
print(int(float("3.5")))  # ✅ Output: 3
```

“3.5” மாதிரி தசமப் பகுதி இருக்கும்போது, முதலில் `float()` மூலமாக அதை ஒரு தசம எண்ணாக மாற்றி, பிறகு `int()` மூலம் முழு எண்ணாக மாற்ற வேண்டும். அப்போது, அந்த தசமப் பகுதி (.5 portion) துண்டிக்கப்படும். இது **truncation** என்று அழைக்கப்படுகிறது.

நம் புரிதலைச் சோதிப்போம் : எண் கணித விளையாட்டு (Number Guessing Game)

நீங்கள் கற்றதைச் சோதித்துப் பார்க்க, ஒரு சிறந்த வழி அதைச் செயல்படுத்திப் பார்ப்பதுதான். இதோ, நீங்கள் பைதான் மற்றும் எண்களைப் பற்றிப் புரிந்து கொண்டதை உறுதிப்படுத்த ஒரு எளிய ஆனால் புத்திசாலித்தனமான விளையாட்டு: **Number Guessing Game (எண் கணித விளையாட்டு)**.

விளையாட்டின் நோக்கம்: கற்றதை நடைமுறைப்படுத்துதல்

இந்த விளையாட்டை உருவாக்குவதன் மூலம், நீங்கள் எண் வகைகளைப் புரிந்துகொள்வதோடு மட்டுமல்லாமல், பின்வரும் அடிப்படைக் கருத்துக்களையும் வலுப்படுத்துவீர்கள்:

- **எண் வகை மாற்றம் (Type Conversion):** பயனர் உள்ளிடும் தகவலை எப்படி எண்ணாக மாற்றுவது.

- **சீரற்ற எண்கள் (Random Numbers):** ஒரு விளையாட்டிற்கான மர்மமான எண்ணை எப்படி உருவாக்குவது.
- **லூப் (Loop):** ஒரு செயலை மீண்டும் மீண்டும் எப்படிச் செய்வது.
- **நிபந்தனைகள் (Conditionals):** ஒரு முடிவை எப்படி எடுப்பது (`if`, `elif`, `else`).

விளையாட்டின் விதிகள்: ஒரு எளிய ஆனால் சவாலான புதிர்

விளையாட்டின் விதிகள் மிக எளிமையானவை:

1. கணினி 1 முதல் 20 வரையிலான ஒரு ரகசிய எண்ணைத் தானாகவே தேர்ந்தெடுக்கும்.
2. நீங்கள் அந்த எண்ணைக் கண்டுபிடிக்க வேண்டும்.
3. ஒவ்வொரு முறை நீங்கள் ஒரு எண்ணைச் சொல்லும்போதும், கணினி அது அதிகமாக இருக்கிறதா (Too high!) அல்லது குறைவாக இருக்கிறதா (Too low!) என்று சொல்லும்.
4. எண்ணைக் கண்டுபிடிக்கும் வரை நீங்கள் முயற்சிக்கலாம்.
5. நீங்கள் எத்தனை முயற்சிகளில் எண்ணைக் கண்டுபிடித்தீர்கள் என்று கணினி இறுதியில் தெரிவிக்கும்.

இந்த விளையாட்டை பைதான் மூலம் எப்படி உருவாக்குவது என்று படி படியாகப் பார்ப்போம்.

பைதான் குறியீடு: படி படியாக ஒரு விளையாட்டு உருவாக்கம்

படி 1: தேவையான 'விளையாட்டுப் பொருட்கள்' - `random` module

ஒரு சீரற்ற, மர்மமான எண்ணை உருவாக்க, பைதான் ஒரு சிறப்பு 'தொகுப்பை' (module) வழங்குகிறது, அதன் பெயர் `random`. இதை நாம் நம் குறியீட்டிற்குள் கொண்டு வர வேண்டும்.

Python

```python
import random # 'random' தொகுப்பை இறக்குமதி செய்கிறோம்
```

படி 2: மர்மமான எண்ணை உருவாக்குதல்

இப்போது, 1 முதல் 20 வரையிலான ஒரு முழு எண்ணை (int) கணினியைத் தேர்ந்தெடுக்கச் சொல்லலாம்.

Python

```python
secret_number = random.randint(1, 20) # 1 மற்றும் 20 உட்பட ஒரு சீரற்ற முழு எண்
print("நான் ஒரு ரகசிய எண்ணை மனதில் வைத்திருக்கிறேன்... கண்டுபிடி பார்ப்போம்!")
```

படி 3: வீரரின் முதல் முயற்சி - உள்ளீடு பெறுதல்

விளையாட்டின் விதிப்படி, நாம் வீரரிடமிருந்து ஒரு எண்ணை உள்ளீடாகப் பெற வேண்டும். பயனர் கொடுக்கும் உள்ளீடு எப்போதும் ஒரு `string` ஆக (எழுத்து வடிவில்) இருக்கும் என்பதை நினைவில் கொள்ளுங்கள். அதை நாம் `int` ஆக மாற்ற வேண்டும்.

Python

```python
# பயனர் உள்ளீட்டைப் பெறுகிறோம்
guess_str = input("உங்கள் யூகத்தைச் சொல்லுங்கள் (1 முதல் 20 வரை): ")

# உள்ளீட்டை முழு எண்ணாக மாற்றுகிறோம் (இங்குதான் type conversion முக்கியம்!)
guess = int(guess_str)
```

படி 4: மந்திர ‘லூப்’ - `while` Loop-இன் சக்தி

நாம் எண்ணைக் கண்டுபிடிக்கும் வரை, வீரர் தொடர்ந்து முயற்சிக்க வேண்டும். ஒரு குறிப்பிட்ட நிபந்தனை பூர்த்தியாகும் வரை ஒரு செயலைத் திரும்பத் திரும்பச் செய்ய, நாம் `while` loop-ஐப் பயன்படுத்துவோம். இங்கு, “கண்டுபிடித்த எண் ரகசிய எண்ணுக்குச் சமம் இல்லை” என்ற நிபந்தனை இருக்கும் வரை loop தொடர வேண்டும்.

Python

```
attempts = 0 # முயற்சிகளின் எண்ணிக்கையைக் கணக்கிட ஒரு மாறி (variable)
# இங்கு முழு குறியீட்டையும் கொடுக்கவில்லை, அடுத்த படியைப் பார்த்த பிறகு முழு குறியீட்டைக் காணலாம்
```

படி 5: முடிவெடுக்கும் கலை - `if`, `elif`, `else`

ஒவ்வொரு முறை யூகத்தை உள்ளிடும்போதும், அது ரகசிய எண்ணை விட அதிகமாக இருக்கிறதா, குறைவாக இருக்கிறதா, அல்லது சரியாக இருக்கிறதா என்று சொல்ல வேண்டும். இதற்கு நிபந்தனைச் சாய்வுகள் (`if`, `elif`, `else`) தேவை.

Python

```
if guess < secret_number:
    print("உங்கள் யூகம் மிகவும் குறைவு!")
elif guess > secret_number:
    print("உங்கள் யூகம் மிகவும் அதிகம்!")
else:
    print(f"வாழ்த்துகள்! நீங்கள் {attempts} முயற்சிகளில் சரியான எண்ணைக் கண்டுபிடித்துவிட்டீர்கள்!")
```

படி 6: முயற்சிகளைக் கணக்கிடுதல்

ஒவ்வொரு முறை பயனர் ஒரு யூகத்தைச் சொல்லும்போதும், `attempts` என்ற மாறியின் மதிப்பை 1 அதிகரிக்க வேண்டும்.

Python

```
attempts += 1 # attempts = attempts + 1 என்பதற்குச் சமம்
```

படி 7: முழுமையான குறியீடு (Full Code)

இப்போது, மேலே நாம் பார்த்த அனைத்துப் பகுதிகளையும் இணைத்து ஒரு முழுமையான Number Guessing Game குறியீட்டை உருவாக்கலாம்! உங்கள் பைதான் எடிட்டரில் இதை உள்ளிட்டு இயக்கவும்.

Python

```
import random # 'random' தொகுப்பை இறக்குமதி செய்கிறோம்

secret_number = random.randint(1, 20) # 1 மற்றும் 20 உட்பட ஒரு சீரற்ற முழு எண்
attempts = 0 # முயற்சிகளின் எண்ணிக்கையைக் கணக்கிட ஆரம்ப மதிப்பு

print("நான் ஒரு ரகசிய எண்ணை மனதில் வைத்திருக்கிறேன்... கண்டுபிடி பார்ப்போம்!")
print("அது 1 முதல் 20 வரை உள்ள ஒரு எண்.")

# எண்ணைக் கண்டுபிடிக்கும் வரை இந்த லூப் தொடரும்
while True: # 'True' என்றால் லூப் தொடர்ந்து இயங்கும், உள்ளே 'break' தேவை
```

```
    try:
        # பயனர் உள்ளீட்டைப் பெறுகிறோம்
        guess_str = input("உங்கள் யூகத்தைச் சொல்லுங்கள்: ")

        # உள்ளீட்டை முழு எண்ணாக மாற்றுகிறோம் (இங்குதான் int() type conversion முக்கியம்!)
        guess = int(guess_str)

        # ஒவ்வொரு முறை யூகிக்கும்போதும் முயற்சிகளைக் கணக்கிடுகிறோம்
        attempts += 1

        if guess < secret_number:
            print("உங்கள் யூகம் மிகவும் குறைவு! இன்னும் கொஞ்சம் அதிகம் முயற்சி செய்யுங்கள்.")
        elif guess > secret_number:
            print("உங்கள் யூகம் மிகவும் அதிகம்! இன்னும் கொஞ்சம் குறைவாக முயற்சி செய்யுங்கள்.")
        else:
            # சரியான எண்ணைக் கண்டுபிடித்துவிட்டால்
            print(f"\nவாழ்த்துகள்! நீங்கள் {attempts} முயற்சிகளில் சரியான எண்ணைக் கண்டுபிடித்துவிட்டீர்கள்!")
            break # எண்ணைக் கண்டுபிடித்துவிட்டதால் லூப்பை விட்டு வெளியேறுகிறோம்

    except ValueError:
        # பயனர் எண் அல்லாத வேறு ஏதேனும் உள்ளிட்டால் பிழையைக் கையாளுகிறோம்
        print("இது ஒரு சரியான எண் அல்ல. தயவுசெய்து ஒரு முழு எண்ணை உள்ளிடவும்.")
```

ஒரு சிறிய குறிப்பு: மேலே உள்ள குறியீட்டில் `try-except ValueError` பகுதியைப் பார்த்தீர்கள் அல்லவா? நீங்கள் அத்தியாயம் 1-ல் `int("3.5")` பிழை கொடுக்கும் என்பதைப் படித்தீர்கள். அதேபோல, பயனர் “hello” அல்லது வேறு எழுத்துகளை உள்ளிடும்போது `int()` அதை எண்ணாக மாற்ற முடியாது. அந்தச் சமயத்தில் உங்கள் நிரல் உடைந்துவிடாமல், ஒரு பிழையைக் கையாள `try-except` பயன்படுத்தப்படுகிறது. இதைப்பற்றி வரும் அத்தியாயங்களில் விரிவாகக் கற்போம். இப்போதைக்கு, இது நம் விளையாட்டைப் பாதுகாப்பானதாக்குகிறது என்று மட்டும் புரிந்துகொண்டால் போதும்.

விளையாட்டிலிருந்து நாம் கற்றுக்கொண்டது

இந்த எளிய விளையாட்டை உருவாக்குவதன் மூலம், நீங்கள் பைதானின் எண்களைப் பற்றிய புரிதலைக் கோட்பாட்டு ரீதியாகவும், நடைமுறை ரீதியாகவும் வலுப்படுத்தியுள்ளீர்கள். குறிப்பாக:

- **தரவு வகைகள்:** `int` எப்படிப் பயன்படுத்தப்படுகிறது, மற்றும் `input()` மூலம் கிடைக்கும் `string`-ஐ `int()` பயன்படுத்தி எப்படி எண்ணாக மாற்றுவது என்பதைக் கண்டோம்.
- **மாறிகள் (Variables):** `secret_number`, `attempts`, `guess` போன்ற மாறிகளில் எப்படி மதிப்புகளைச் சேமிப்பது என்பதைக் கற்றுக்கொண்டோம்.
- **கணிதச் செயல்பாடுகள்:** மறைமுகமாக, ஒப்பிடும் செயல்பாடுகளைப் (comparison operators: `<`, `>`, `==`) பயன்படுத்தினோம்.
- **கட்டுப்பாட்டு ஓட்டம் (Control Flow):** `while` loop மூலம் ஒரு செயலை எப்படி மீண்டும் மீண்டும் செய்வது என்பதையும், `if-elif-else` மூலம் எப்படி முடிவெடுப்பது என்பதையும் அறிந்தோம்.

எண்களின் இந்த அடிப்படைப் புரிதல், உங்கள் பைதான் பயணத்தின் முதல் ஆனால் மிக உறுதியான படி. இந்தக் கோட்பாடுகள் ஒரு தொடக்கப் புள்ளியே. நீங்கள் இந்த அறிவை வைத்துக்கொண்டு, அடுத்த பகுதியில் மேலும் சுவாரசியமான தரவு வகைகளைப் பற்றி அறிந்துகொள்வீர்கள்.

2. பைதானில் எழுத்துத் தரவு (Text Data - Strings)

எண்களைப் பற்றி நாம் விரிவாகப் பேசிவிட்டோம். ஆனால், நம் உலகம் வெறும் எண்களால் ஆனது அல்ல, இல்லையா? நம் அன்றாட வாழ்வில் நாம் எழுத்துகளையும், வார்த்தைகளையும், வாக்கியங்களையும்தான் அதிகம் பயன்படுத்துகிறோம். உங்கள் பெயர், முகவரி, நண்பர்களுக்கு அனுப்பும் மெசேஜ்கள், நீங்கள் படிக்கும் இந்த புத்தகம் – இவை அனைத்தும் எழுத்துகளின் கோர்வையே!

கணினி உலகில், இந்த எழுத்துகளையும், வார்த்தைகளையும் நாம் **'ஸ்ட்ரிங்ஸ்' (Strings)** என்று அழைக்கிறோம். பைதானில் `str` வகை என்பது இந்த எழுத்துத் தரவுகளுக்காகப் பிரத்யேகமாக வடிவமைக்கப்பட்டது. ஒரு மனிதன் எப்படி மொழி இல்லாமல் பேச முடியாதோ, அதேபோல், பைதான் போன்ற Programming Languages, 'ஸ்ட்ரிங்ஸ்' இல்லாமல் தகவல்களைக் கையாள முடியாது.

நாம் ஒரு வலைத்தளத்தை (website) உருவாக்குகிறோம் என்று எடுத்துக்கொள்ளுங்கள். பயனரின் பெயர், மின்னஞ்சல் முகவரி, அவர் உள்ளிடும் கருத்துகள் – இவை அனைத்தும் ஸ்ட்ரிங்ஸ் தான். ஒரு மின்னஞ்சல் முகவரியில் `@` இருக்கிறதா என்று சோதிப்பதோ, ஒரு வாக்கியத்தில் குறிப்பிட்ட வார்த்தையைத் தேடுவதோ, அல்லது ஒரு பெயரைப் பெரிய எழுத்தில் மாற்றுவதோ – இந்த வேலைகளுக்கெல்லாம் `str` வகை அவசியம்.

எண்களைப் போலவே, ஸ்ட்ரிங்ஸ் என்பதும் பைதானில் மிக சக்திவாய்ந்த ஒரு தரவு வகையாகும். அதன் பல நுட்பமான அம்சங்கள், ஒரு புரோகிராமரின் வாழ்க்கையை மிகவும் எளிதாக்கும். இந்த அத்தியாயத்தில், ஸ்ட்ரிங்ஸ் என்றால் என்ன, அவற்றை எப்படி உருவாக்குவது, எப்படி அதனுடன் விளையாடுவது, அதன் மறைக்கப்பட்ட சில சூப்பர் பவர்கள் என்னென்ன என்று விரிவாகக் கற்கப் போகிறோம்.

உங்கள் பைதான் பயணத்தின் அடுத்த சாகசத்திற்குத் தயாராகுங்கள்!

2.1 ஸ்ட்ரிங்ஸ் என்றால் என்ன? (What are Strings?)

எளிமையாகச் சொன்னால், **ஸ்ட்ரிங் (String)** என்பது வரிசையாக அடுக்கி வைக்கப்பட்ட எழுத்துகளின் தொகுப்பு. இந்த எழுத்துகள் வார்த்தைகளாகவோ, வாக்கியங்களாகவோ, ஏன் வெறும் ஒரு எழுத்தோ, அல்லது ஒரு எண் வரிசையாகவோ கூட இருக்கலாம். பைதானில் ஒரு ஸ்ட்ரிங்கை உருவாக்க, அதை ஒற்றை மேற்கோள்களுக்குள் (`' '`) அல்லது இரட்டை மேற்கோள்களுக்குள் (`" "`) இட வேண்டும்.

Python

```python
# ஒரு ஒற்றை மேற்கோள் ஸ்ட்ரிங்
my_name = 'அறிவு'
print(f"என் பெயர்: {my_name}")

# ஒரு இரட்டை மேற்கோள் ஸ்ட்ரிங்
greeting = "வணக்கம், உலகமே!"
print(f"வாழ்த்து: {greeting}")

# ஒரு நீண்ட வாக்கியம் ஸ்ட்ரிங்
long_sentence = "பைதான் நிரலாக்கம் கற்க மிகவும் எளிமையானது மற்றும் சக்திவாய்ந்தது."
print(f"வாக்கியம்: {long_sentence}")
```

இரண்டுமே ஒரே வேலைதான் செய்யும். ஆனால், உங்கள் ஸ்ட்ரிங்குக்குள் ஒற்றை மேற்கோள் அல்லது இரட்டை மேற்கோள் வந்தால் என்ன செய்வது?

Python

```
# பிழை ஏற்படும்!
# message = 'அவர் சொன்னார், 'பைதான் அருமை!' '

# தீர்வு: இரட்டை மேற்கோள்களைப் பயன்படுத்துதல்
message_double = "அவர் சொன்னார், 'பைதான் அருமை!'"
print(f"சரியான செய்தி: {message_double}")

# அல்லது, ஒற்றை மேற்கோள்களைப் பயன்படுத்தி, உள்ளே உள்ள ஒற்றை மேற்கோளை 'escapes' செய்தல்
message_escape = 'அவர் சொன்னார், \'பைதான் அருமை!\''
print(f"சரியான செய்தி (escaped): {message_escape}")
```

மேலே உள்ள `\'` என்பதை **'escape character'** என்று சொல்வார்கள். `\` (backslash) குறியீடு, அதற்குப் பின் வரும் எழுத்து ஒரு சிறப்பு அர்த்தத்தைக் கொண்டது என்று பைதானுக்குச் சொல்கிறது.

மல்டி-லைன் ஸ்ட்ரிங்ஸ் (Multi-line Strings): கவிதை எழுதலாமா?

சில சமயம், ஒரு நீண்ட பத்தியை, அல்லது ஒரு கவிதையை, பல வரிகளில் எழுத வேண்டியிருக்கும். அதற்கு மூன்று மேற்கோள்களை (`'''` அல்லது `"""`) பயன்படுத்தலாம்.

Python

```
# ஒரு கவிதை
my_poem = """அகர முதல எழுத்தெல்லாம் ஆதி
பகவன் முதற்றே உலகு.
- திருக்குறள்"""
print(f"திருக்குறள்: \n{my_poem}")

# ஒரு நீண்ட விளக்கம்
description = """
இந்த அத்தியாயத்தில், பைதான் மொழியின்
அடிப்படை எண் வகைகளைக் கற்றோம்.
அடுத்ததாக, எழுத்துத் தரவுகளைப் பற்றிப் பார்க்கிறோம்.
"""
print(f"விளக்கம்:\n{description}")
```

இந்த மல்டி-லைன் ஸ்ட்ரிங்ஸ், உங்கள் குறியீட்டைப் படிக்க எளிதாக்குவதுடன், பல வரிகளிலான உள்ளடக்கங்களைச் சேமிக்கவும் உதவுகின்றன.

2.2 ஸ்ட்ரிங்ஸ் என்பவை 'வரிசை'கள் (Strings are Sequences)

எப்படி ஒரு பாமாலையில் முத்துகள் வரிசையாக அடுக்கப்பட்டிருக்குமோ, அதேபோல், பைதானில் ஒரு ஸ்ட்ரிங்கில் உள்ள எழுத்துகள் ஒரு குறிப்பிட்ட வரிசையில் அடுக்கப்பட்டிருக்கும். இந்த வரிசைமுறை மிக முக்கியம்!

இன்டெக்ஸிங் (Indexing): எழுத்துகளைக் கண்டறிதல்

ஒரு ஸ்ட்ரிங்கில் உள்ள ஒவ்வொரு எழுத்துக்கும் ஒரு தனிப்பட்ட 'இடம்' அல்லது **'இன்டெக்ஸ்' (Index)** உண்டு. பைதானில், இந்த இன்டெக்ஸ் **0-ல் இருந்து தொடங்குகிறது**.

உதாரணமாக, `word = "Python"` என்ற ஸ்ட்ரிங்கைப் பார்ப்போம்:

எழுத்து	P	y	t	h	o	n
இன்டெக்ஸ் (நேர்மறை)	0	1	2	3	4	5
இன்டெக்ஸ் (எதிர்மறை)	-6	-5	-4	-3	-2	-1

- `word[0]` என்றால் ‘P’ கிடைக்கும்.
- `word[1]` என்றால் ‘y’ கிடைக்கும்.

Python

```
programming_language = "Python"
print(f"முதல் எழுத்து: {programming_language[0]}")  # Output: P
print(f"மூன்றாவது எழுத்து: {programming_language[2]}") # Output: t
```

எதிர்மறை இன்டெக்ஸிங் (Negative Indexing): பின்னோக்கிப் பார்த்தல்

பைதான் ஒரு கூடுதல் வசதியை வழங்குகிறது: பின்னோக்கி இன்டெக்ஸ் செய்யலாம்!

- `word[-1]` என்றால் கடைசி எழுத்தான ‘n’ கிடைக்கும்.
- `word[-2]` என்றால் கடைசிக்கு முந்தைய எழுத்தான ‘o’ கிடைக்கும்.

Python

```
print(f"கடைசி எழுத்து: {programming_language[-1]}")  # Output: n
print(f"கடைசிக்கு முந்தைய எழுத்து: {programming_language[-2]}") # Output: o
```

இது, ஸ்ட்ரிங்கின் நீளத்தைப் பற்றி கவலைப்படாமல், கடைசி எழுத்துகளை அணுக மிகவும் பயனுள்ளதாக இருக்கும்.

ஸ்ட்ரிங்ஸ் மாற்ற முடியாதவை (Strings are Immutable): ஒருமுறை உருவாக்கப்பட்டால் மாறாது!

இது ஸ்ட்ரிங்ஸ் பற்றிய ஒரு மிக முக்கியமான அம்சம்! பைதானில் ஒரு ஸ்ட்ரிங்கை ஒருமுறை உருவாக்கிவிட்டால், அதன் உள்ளே இருக்கும் எழுத்துகளை நேரடியாக மாற்ற முடியாது.

Python

```
my_string = "Hello"
# my_string[0] = 'J' # ❌ இது பிழை ஏற்படுத்தும்! (TypeError)
```

“அப்படியானால் ஒரு ஸ்ட்ரிங்கைப் புதுப்பிக்கவே முடியாதா?” என்று நீங்கள் கேட்கலாம். நேரடியாக மாற்ற முடியாது என்றாலும், புதிய ஸ்ட்ரிங்கை உருவாக்கலாம்:

Python

```
original_string = "Hello World"
# "World" என்பதை "Python" என மாற்ற வேண்டும்
new_string = original_string.replace("World", "Python")
print(f"Original: {original_string}") # Output: Hello World
print(f"New: {new_string}")           # Output: Hello Python

# அல்லது எழுத்துகளை இணைத்து புதிய ஸ்ட்ரிங் உருவாக்குதல்
another_new_string = "J" + original_string[1:] # முதல் எழுத்தை மட்டும் மாற்றுகிறோம்
print(f"Another new string: {another_new_string}") # Output: Jello World
```

இந்த `immutable` தன்மை, ஸ்ட்ரிங்ஸ் பாதுகாப்பாக இருப்பதையும், அவை எதிர்பாராத விதமாக மாறாமல் இருப்பதையும் உறுதி செய்கிறது.

2.3 ஸ்ட்ரிங் துண்டாக்கும் கலை (String Slicing): ஒரு பகுதியைப் பிரித்தெடுத்தல்

முழு ஸ்ட்ரிங்கையும் அணுகாமல், அதன் ஒரு குறிப்பிட்ட பகுதியை மட்டும் பிரித்தெடுக்க **‘ஸ்லைஸிங்’ (Slicing)** என்ற நுட்பத்தைப் பயன்படுத்துவோம். இது ஒரு நீண்ட சமையல் வீடியோவில், தேவையான பகுதியைக் கத்தரித்து எடுப்பது போல!

ஸ்லைசிங் மூன்று பாகங்களைக் கொண்டது: `[start:end:step]`

- `start`: எங்கிருந்து தொடங்க வேண்டும் (இந்த இன்டெக்ஸ் சேர்க்கப்படும்).
- `end`: எங்கு முடிக்க வேண்டும் (இந்த இன்டெக்ஸ் சேர்க்கப்படாது).
- `step`: எத்தனை இன்டெக்ஸ் தாண்டி அடுத்த எழுத்தைக் கணக்கில் கொள்ள வேண்டும் (விரும்பினால் பயன்படுத்தலாம், இயல்பாக 1).

Python

```
my_sentence = "Python Programming is fun"

# முதல் 6 எழுத்துகள்
part1 = my_sentence[0:6]
print(f"முதல் பகுதி: {part1}") # Output: Python

# ஒரு குறிப்பிட்ட பகுதியிலிருந்து இறுதி வரை
part2 = my_sentence[11:] # 11வது இன்டெக்ஸில் இருந்து இறுதி வரை
print(f"இரண்டாம் பகுதி: {part2}") # Output: Programming is fun

# ஆரம்பத்திலிருந்து ஒரு குறிப்பிட்ட பகுதி வரை
part3 = my_sentence[:6]
print(f"மூன்றாம் பகுதி: {part3}") # Output: Python

# முழு ஸ்ட்ரிங் (இதற்குப் பெரும்பாலும் தேவையில்லை)
full_string = my_sentence[:]
print(f"முழு ஸ்ட்ரிங்: {full_string}") # Output: Python Programming is fun

# எதிர்மறை இன்டெக்ஸிங் உடன் ஸ்லைசிங்
last_word = my_sentence[-3:] # கடைசி 3 எழுத்துகள்
print(f"கடைசி சொல்: {last_word}") # Output: fun
```

`step` பயன்பாடு:

`step` என்பது நீங்கள் எத்தனை எழுத்துகள் தாண்டி அடுத்த எழுத்தை எடுக்க வேண்டும் என்று குறிக்கும்.

Python

```
text = "HelloWorld"
every_other = text[0:10:2] # முதல் எழுத்தில் தொடங்கி, இரண்டுக்கு ஒன்று விட்டு
print(f"ஒன்றுவிட்ட எழுத்துகள்: {every_other}") # Output: Hlool
```

ஸ்ட்ரிங்கை தலைகீழாக மாற்றுதல் (Reverse a String): ஒரு மாயாஜால ட்ரிக்!

ஸ்லைசிங் பயன்படுத்தி ஒரு ஸ்ட்ரிங்கை மிக எளிதாகத் தலைகீழாக மாற்றலாம்.

Python

```
original = "madam"
reversed_text = original[::-1] # -1 step, அதாவது பின்னோக்கி
print(f"தலைகீழ்: {reversed_text}") # Output: madam
```

இது ஸ்ட்ரிங்ஸ் பற்றிய உங்கள் புரிதலை மேலும் வலுப்படுத்தும். அடுத்ததாக, ஸ்ட்ரிங்ஸ் உடன் வேலை செய்ய பைதான் வழங்கும் சில சக்திவாய்ந்த செயல்பாடுகளைப் பார்ப்போம்!

3. பைதானில் பட்டியல்கள் (Lists)

எண்களைப் பற்றிப் பேசினோம், எழுத்துகளுடன் (strings) எப்படி விளையாடுவது என்று பார்த்தோம். இப்போது, நாம் சேகரிக்கும் தகவல்கள் ஒரே ஒரு எண் அல்லது ஒரு வார்த்தையாக மட்டும் இருப்பதில்லை, இல்லையா? பல சமயங்களில், நாம் பல தகவல்களை ஒன்றாகச் சேமிக்க வேண்டியிருக்கும்.

உதாரணமாக:

- உங்கள் தினசரி செய்ய வேண்டிய பணிகளின் பட்டியல் (To-do list).
- ஒரு வாரத்தின் மளிகைப் பொருட்கள் பட்டியல் (Grocery list).
- ஒரு குழுவில் உள்ள மாணவர்களின் மதிப்பெண்களின் வரிசை.
- உங்கள் பிடித்த திரைப்படங்களின் வரிசை.

இந்த எல்லா இடங்களிலும் நாம் பல விஷயங்களை ஒரு **வரிசையாக**, ஒரு **தொகுப்பாக** வைத்திருக்கிறோம். கணினி உலகில், இந்த வகையான தொகுப்புகளைச் சேமிக்கப் பைதான் ஒரு சூப்பர் பவரை வழங்குகிறது – அதுதான் **பட்டியல்கள் (Lists)**.

ஒரு பட்டியல் என்பது பைதானில் மிக முக்கியமான மற்றும் பல்துறை சார்ந்த (versatile) ஒரு தரவு வகையாகும். இது பலதரப்பட்ட தகவல்களை ஒருங்கே சேமிக்க உதவுகிறது. இந்த அத்தியாயத்தில், பட்டியல்கள் என்றால் என்ன, அவற்றை எப்படி உருவாக்குவது, எப்படி மாற்றுவது, அதனுடன் விளையாடும் நுணுக்கங்கள் என்னென்ன என்று விரிவாகக் கற்கப் போகிறோம்.

உங்கள் பைதான் பயணத்தின் அடுத்த சாகசத்திற்குத் தயாராகுங்கள்!

3.1 பட்டியல்கள் என்றால் என்ன? (What are Lists?)

எளிமையாகச் சொன்னால், **பட்டியல் (List)** என்பது பலதரப்பட்ட மதிப்புகளின் (values) ஒரு வரிசைப்படுத்தப்பட்ட தொகுப்பு. இந்த மதிப்புகள் எண்களாக இருக்கலாம், எழுத்துகளாக இருக்கலாம், ஏன் மற்ற பட்டியல்களாகக்கூட இருக்கலாம்! பைதானில் ஒரு பட்டியலை உருவாக்க, மதிப்புகளைச் சதுர அடைப்புக்குறிகளுக்குள் (`[ ]`) இட்டு, ஒவ்வொரு மதிப்புக்கும் இடையில் காற்புள்ளி (`,`) இட வேண்டும்.

Python

```python
# எண்களின் பட்டியல்
numbers = [1, 2, 3, 4, 5]
print(f"எண்களின் பட்டியல்: {numbers}")

# எழுத்துகளின் பட்டியல்
fruits = ["apple", "banana", "cherry"]
print(f"பழங்களின் பட்டியல்: {fruits}")

# கலப்புத் தரவு வகைகளின் பட்டியல் (எண்கள், எழுத்துகள் கலந்து)
mixed_list = [10, "Python", 3.14, True]
print(f"கலப்புப் பட்டியல்: {mixed_list}")

# பட்டியலுக்குள் ஒரு பட்டியல் (Nested List)
matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
print(f"அடுக்கு பட்டியல்: {matrix}")
```

பட்டியல்கள் பலதரப்பட்ட தரவுகளை ஒன்றாகச் சேமிக்க உதவுகின்றன. இது பைதானின் நெகிழ்வுத்தன்மைக்கு ஒரு சிறந்த எடுத்துக்காட்டு.

3.2 பட்டியல்களை அணுகுதல் (Accessing List Elements): ஒரு பொருளைக் கண்டறிதல்

ஒரு ஸ்ட்ரிங்கைப் போலவே, பட்டியலிலும் உள்ள ஒவ்வொரு உறுப்புக்கும் ஒரு தனிப்பட்ட 'இடம்' அல்லது **'இன்டெக்ஸ்' (Index)** உண்டு. பைதானில், இந்த இன்டெக்ஸ் **0-ல் இருந்து தொடங்குகிறது**.

உதாரணமாக, `my_list = ["a", "b", "c", "d"]` என்ற பட்டியலைப் பார்ப்போம்:

உறுப்பு	“a”	“b”	“c”	“d”
இன்டெக்ஸ் (நேர்மறை)	0	1	2	3
இன்டெக்ஸ் (எதிர்மறை)	-4	-3	-2	-1

- `my_list[0]` என்றால் “a” கிடைக்கும்.
- `my_list[1]` என்றால் “b” கிடைக்கும்.

Python

```python
programming_languages = ["Python", "Java", "C++", "JavaScript"]

print(f"முதல் மொழி: {programming_languages[0]}")  # Output: Python
print(f"மூன்றாவது மொழி: {programming_languages[2]}") # Output: C++
```

எதிர்மறை இன்டெக்ஸிங் (Negative Indexing): பின்னோக்கிப் பார்த்தல்

ஸ்ட்ரிங்குகளைப் போலவே, பட்டியல்களிலும் பின்னோக்கி இன்டெக்ஸ் செய்யலாம்.

- `my_list[-1]` என்றால் கடைசி உறுப்பு கிடைக்கும்.
- `my_list[-2]` என்றால் கடைசிக்கு முந்தைய உறுப்பு கிடைக்கும்.

Python

```
print(f"கடைசி மொழி: {programming_languages[-1]}")  # Output: JavaScript
print(f"கடைசிக்கு முந்தைய மொழி: {programming_languages[-2]}") # Output: C++
```

3.3 பட்டியல்களை மாற்றுதல் (Modifying Lists): ஒரு புதிய பட்டியல் உருவாக்குதல்

ஸ்ட்ரிங்குகளைப் போலல்லாமல், பட்டியல்கள் **மாற்றக்கூடியவை (mutable)**. அதாவது, ஒரு பட்டியலை உருவாக்கிய பிறகு, அதன் உள்ளே உள்ள உறுப்புகளை நாம் நேரடியாக மாற்றலாம், புதிய உறுப்புகளைச் சேர்க்கலாம், அல்லது ஏற்கனவே உள்ளவற்றை நீக்கலாம். இது பட்டியல்களின் ஒரு மிக முக்கியமான சக்தி!

3.3.1 ஒரு உறுப்பை மாற்றுதல் (Changing an Element)

ஒரு குறிப்பிட்ட இன்டெக்ஸில் உள்ள உறுப்பின் மதிப்பை நேரடியாக மாற்றலாம்.

Python

```
colors = ["red", "green", "blue"]
print(f"அசல் பட்டியல்: {colors}") # Output: ['red', 'green', 'blue']

colors[1] = "yellow" # இரண்டாவது உறுப்பை மாற்றுகிறோம் (green -> yellow)
print(f"மாற்றப்பட்ட பட்டியல்: {colors}") # Output: ['red', 'yellow', 'blue']
```

3.3.2 உறுப்புகளைச் சேர்த்தல் (Adding Elements)

பட்டியல்களுடன் புதிய உறுப்புகளைச் சேர்க்க மூன்று முக்கிய methods உள்ளன:

- `append(item)`: பட்டியலின் **கடைசியில்** ஒரு உறுப்பைச் சேர்க்கும்.
- `insert(index, item)`: ஒரு குறிப்பிட்ட **இன்டெக்ஸில்** ஒரு உறுப்பைச் சேர்க்கும்.
- `extend(iterable)`: ஒரு பட்டியலுடன் மற்றொரு பட்டியலின் உறுப்புகளைச் சேர்க்கும் (ஒன்றுக்கும் மேற்பட்ட உறுப்புகளைச் சேர்க்க).

Python

```
my_list = ["apple", "banana"]

# append(): கடைசியில் சேர்த்தல்
my_list.append("cherry")
print(f"append செய்த பின்: {my_list}") # Output: ['apple', 'banana', 'cherry']

# insert(): குறிப்பிட்ட இடத்தில் சேர்த்தல்
my_list.insert(1, "orange") # index 1-ல் orange-ஐ சேர்க்கிறோம்
```

```
print(f"insert செய்த பின்: {my_list}") # Output: ['apple', 'orange', 'banana', 'cherry']

# extend(): பல உறுப்புகளைச் சேர்த்தல்
more_fruits = ["grape", "kiwi"]
my_list.extend(more_fruits)
print(f"extend செய்த பின்: {my_list}") # Output: ['apple', 'orange', 'banana', 'cherry', 'grape', 'kiwi']

# '+' ஆப்பரேட்டர் மூலமாகவும் பட்டியல்களை இணைக்கலாம் (இது ஒரு புதிய பட்டியலை உருவாக்கும்)
combined_list = my_list + ["mango", "peach"]
print(f"கூட்டப்பட்ட பட்டியல்: {combined_list}")
```

3.3.3 உறுப்புகளை நீக்குதல் (Removing Elements)

பட்டியல்களில் இருந்து உறுப்புகளை நீக்கவும் பல வழிகள் உள்ளன:

- `remove(value)`: பட்டியலிலிருந்து ஒரு குறிப்பிட்ட **மதிப்பை** (value) நீக்கும். (முதல் தோற்றத்தை மட்டுமே நீக்கும்).
- `pop(index)`: ஒரு குறிப்பிட்ட **இன்டெக்ஸில்** உள்ள உறுப்பை நீக்கி, நீக்கப்பட்ட உறுப்பை வழங்கும். இன்டெக்ஸ் கொடுக்கப்படாவிட்டால், கடைசி உறுப்பை நீக்கும்.
- `del statement`: ஒரு குறிப்பிட்ட இன்டெக்ஸ் அல்லது ஸ்லைஸை (slice) நீக்கும்.
- `clear()`: பட்டியலின் அனைத்து உறுப்புகளையும் நீக்கும், ஆனால் பட்டியலை அழிக்காது.

Python

```
shopping_list = ["milk", "bread", "eggs", "milk", "cheese"]
print(f"அசல் பட்டியல்: {shopping_list}")

# remove(): மதிப்பின் மூலம் நீக்குதல்
shopping_list.remove("milk") # முதல் 'milk' நீக்கப்படும்
print(f"remove செய்த பின்: {shopping_list}") # Output: ['bread', 'eggs', 'milk', 'cheese']

# pop(): இன்டெக்ஸ் மூலம் நீக்குதல் (அல்லது கடைசி உறுப்பு)
removed_item = shopping_list.pop(1) # index 1 (eggs) நீக்கப்படும்
print(f"pop செய்த பின்: {shopping_list} (நீக்கப்பட்டது: {removed_item})") # Output: ['bread', 'milk', 'cheese'] (நீக்கப்பட்டது: eggs)

# del statement: இன்டெக்ஸ் அல்லது ஸ்லைஸ் மூலம் நீக்குதல்
del shopping_list[0] # index 0 (bread) நீக்கப்படும்
print(f"del செய்த பின்: {shopping_list}") # Output: ['milk', 'cheese']

# clear(): அனைத்து உறுப்புகளையும் நீக்குதல்
shopping_list.clear()
print(f"clear செய்த பின்: {shopping_list}") # Output: []
```

3.4 பட்டியல்களின் நீளம் மற்றும் பிற செயல்பாடுகள் (List Length and Other Operations)

3.4.1 பட்டியலின் நீளம் (List Length)

ஒரு பட்டியலில் எத்தனை உறுப்புகள் உள்ளன என்பதைக் கண்டறிய `len()` செயல்பாட்டைப் பயன்படுத்தலாம்.

Python

```
my_data = [10, 20, 30, 40, 50]
print(f"பட்டியலின் நீளம்: {len(my_data)}") # Output: 5
```

3.4.2 உறுப்பு உள்ளதா எனச் சரிபார்த்தல் (Checking for an Element)

ஒரு குறிப்பிட்ட உறுப்பு பட்டியலில் உள்ளதா என்று `in` ஆப்பரேட்டரைப் பயன்படுத்திச் சரிபார்க்கலாம். இது `True` அல்லது `False` என்ற Boolean மதிப்பை வழங்கும்.

Python

```
fruits = ["apple", "banana", "cherry"]

print(f"'banana' பட்டியலில் உள்ளதா? {"banana" in fruits}") # Output: True
print(f"'grape' பட்டியலில் உள்ளதா? {"grape" in fruits}")   # Output: False
```

3.4.3 பட்டியலைக் கண்டுபிடித்தல் (Finding an Element’s Index)

ஒரு குறிப்பிட்ட மதிப்பின் இன்டெக்ஸைக் கண்டறிய `index()` method பயன்படுத்தலாம். மதிப்பு பட்டியலில் இல்லாவிட்டால் பிழை ஏற்படும்.

Python

```
planets = ["Mercury", "Venus", "Earth", "Mars"]
earth_index = planets.index("Earth")
print(f"'Earth' இன்டெக்ஸ்: {earth_index}") # Output: 2

# print(planets.index("Jupiter")) # ❌ இது பிழை ஏற்படுத்தும், ஏனெனில் Jupiter பட்டியலில் இல்லை
```

3.4.4 பட்டியலை வரிசைப்படுத்துதல் (Sorting a List)

ஒரு பட்டியலை ஏறுவரிசையில் (ascending) அல்லது இறங்குவரிசையில் (descending) வரிசைப்படுத்தலாம்.

- `sort()`: பட்டியலை நேரடியாக மாற்றும் (in-place sort).
- `sorted()`: பட்டியலை மாற்றாமல், வரிசைப்படுத்தப்பட்ட ஒரு **புதிய பட்டியலை** வழங்கும்.

Python

```
numbers = [3, 1, 4, 1, 5, 9, 2]

# sort() - அசல் பட்டியலை மாற்றும்
numbers.sort()
print(f"sort செய்த பின்: {numbers}") # Output: [1, 1, 2, 3, 4, 5, 9]

# reverse=True பயன்படுத்தி இறங்கு வரிசையில் வரிசைப்படுத்துதல்
numbers.sort(reverse=True)
print(f"இறங்கு வரிசையில்: {numbers}") # Output: [9, 5, 4, 3, 2, 1, 1]
```

```
# sorted() - புதிய பட்டியலை உருவாக்கும்
unsorted_numbers = [5, 2, 8, 1]
new_sorted_list = sorted(unsorted_numbers)
print(f"அசல் பட்டியல்: {unsorted_numbers}") # Output: [5, 2, 8, 1]
print(f"புதிய வரிசைப்படுத்தப்பட்ட பட்டியல்: {new_sorted_list}") # Output: [1, 2, 5, 8]
```

பட்டியல்கள் (Lists) என்பது பைதான் புரோகிராமிங்கின் மிக முக்கியமான மற்றும் அன்றாட வாழ்க்கைக்குப் பயன்படும் ஒரு தரவு வகையாகும். இவை, நீங்கள் சேகரிக்கும் தகவல்களை ஒழுங்கமைக்கவும், கையாளவும் ஒரு சக்திவாய்ந்த கருவியாகச் செயல்படுகின்றன.

பட்டியல்கள் பற்றிய இந்த விரிவான புரிதல், உங்கள் பைதான் பயணத்தில் ஒரு வலிமையான அடியை எடுத்து வைக்க உதவும். அடுத்த பகுதியில், பைதானின் மற்றொரு முக்கியமான தரவு வகையைப் பற்றிப் பார்ப்போம், அது பட்டியல்களுடன் தொடர்புடையது ஆனால் தனித்துவமானது!

4. பைதானில் டியூபிள்ஸ் (Tuples)

எண்கள், எழுத்துகள் (strings) மற்றும் பட்டியல்கள் (lists) பற்றிப் பேசினோம். பட்டியல்கள் என்பவை ஒரு பெரிய பெட்டகம் போல. அதில் பொருட்களைச் சேர்க்கலாம், எடுக்கலாம், வரிசைப்படுத்தலாம், மாற்றி அமைக்கலாம். ஒரு வீட்டு மளிகைப் பொருட்கள் பட்டியலை (grocery list) போல – நாம் எப்போது வேண்டுமானாலும் பொருட்களைச் சேர்க்கலாம் அல்லது நீக்கலாம்.

ஆனால், சில சமயங்களில், நாம் ஒருமுறை உருவாக்கினால், **மாற்றவே முடியாத** ஒரு பட்டியல் நமக்குத் தேவைப்படும், இல்லையா? உதாரணமாக:

- ஒரு கிரிக்கெட் அணியின் நிரந்தர வீரர்களின் பட்டியல்.
- ஒரு வாரத்தின் நாட்களின் வரிசை (திங்கள், செவ்வாய்...).
- ஒரு நட்சத்திரத்தின் ஆயத்தொலைவுகள் (coordinates) – இவை எப்போதும் மாறாது.
- ஒரு மாதத்தின் தேதிகள் – நிரந்தரமானவை.

இந்த மாதிரியான சூழ்நிலைகளில், பைதான் ஒரு சிறப்புத் தரவு வகையை வழங்குகிறது – அதுதான் **டியூபிள்ஸ் (Tuples)**. இது பட்டியல்களைப் போலவே உறுப்புகளை வரிசையாகச் சேமிக்கும், ஆனால் ஒருமுறை உருவாக்கிய பின், அதன் உள்ளே உள்ள உறுப்புகளை நேரடியாக மாற்ற முடியாது. இந்த அத்தியாயத்தில், டியூபிள்ஸ் என்றால் என்ன, பட்டியல்களிலிருந்து அவை எப்படி வேறுபடுகின்றன, அவற்றின் முக்கியத்துவம் என்ன என்பதை விரிவாகக் கற்கப் போகிறோம்.

உங்கள் பைதான் பயணத்தின் அடுத்த உறுதியான அடியை எடுத்து வைப்போம்!

4.1 டியூபிள்ஸ் என்றால் என்ன? (What are Tuples?)

எளிமையாகச் சொன்னால், **டியூபிள் (Tuple)** என்பது பலதரப்பட்ட மதிப்புகளின் (values) ஒரு **வரிசைப்படுத்தப்பட்ட (ordered)** மற்றும் **மாற்ற முடியாத (immutable)** தொகுப்பு. பட்டியல்களைப் போலவே, டியூபிள்களிலும் எண்கள், எழுத்துகள், ஏன் மற்ற டியூபிள்கள்கூட இருக்கலாம். பைதானில் ஒரு டியூபிளை உருவாக்க, மதிப்புகளைச் **சாதாரண அடைப்புக்குறிகளுக்குள் (`()`)** இட்டு, ஒவ்வொரு மதிப்புக்கும் இடையில் காற்புள்ளி (`,`) இட வேண்டும்.

Python

```
# எண்களின் டியூபிள்
coordinates = (10, 20, 30)
print(f"ஆயத்தொலைவுகள்: {coordinates}") # Output: (10, 20, 30)

# எழுத்துகளின் டியூபிள்
days_of_week = ("Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday",
"Sunday")
print(f"வார நாட்கள்: {days_of_week}") # Output: ('Monday', 'Tuesday', ..., 'Sunday')

# கலப்புத் தரவு வகைகளின் டியூபிள்
student_info = ("அறிவு", 101, 3.85, True)
print(f"மாணவர் தகவல்: {student_info}") # Output: ('அறிவு', 101, 3.85, True)
```

ஒரு உறுப்பு கொண்ட டியூபிள்: ஒரு சிறிய நுணுக்கம்!

நீங்கள் ஒரு உறுப்பை மட்டும் கொண்ட ஒரு டியூபிளை உருவாக்க விரும்பினால், ஒரு சிறிய காற்புள்ளி (`,`) ஐச் சேர்க்க மறக்காதீர்கள். இது பைதானுக்கு நீங்கள் ஒரு டியூபிளை உருவாக்குகிறீர்கள் என்று புரிய வைக்கும். இல்லையென்றால், அது ஒரு சாதாரண எண்ணாகவோ, எழுத்துகளாகவோ கருதப்படும்.

Python

```
single_item_tuple = (10,) # ஒரு உறுப்புடன் ஒரு காற்புள்ளி அவசியம்
print(f"ஒரு உறுப்பு டியூபிள்: {single_item_tuple}") # Output: (10,)
print(f"வகை: {type(single_item_tuple)}")        # Output: <class 'tuple'>

not_a_tuple = (10) # காற்புள்ளி இல்லை, இது ஒரு சாதாரண எண்!
print(f"டியூபிள் அல்ல: {not_a_tuple}")         # Output: 10
print(f"வகை: {type(not_a_tuple)}")           # Output: <class 'int'>
```

4.2 டியூபிள் உறுப்புகளை அணுகுதல் (Accessing Tuple Elements)

பட்டியல்களைப் போலவே, டியூபிளிலும் உள்ள ஒவ்வொரு உறுப்புக்கும் ஒரு தனிப்பட்ட 'இடம்' அல்லது **'இன்டெக்ஸ்' (Index)** உண்டு. பைதானில், இந்த இன்டெக்ஸ் **0-ல் இருந்து தொடங்குகிறது**.

உதாரணமாக, `colors = ("red", "green", "blue")` என்ற டியூபிளைப் பார்ப்போம்:

உறுப்பு	"red"	"green"	"blue"
இன்டெக்ஸ் (நேர்மறை)	0	1	2
இன்டெக்ஸ் (எதிர்மறை)	-3	-2	-1

- `colors[0]` என்றால் "red" கிடைக்கும்.
- `colors[1]` என்றால் "green" கிடைக்கும்.

Python

```
top_cities = ("Chennai", "Mumbai", "Delhi", "Bengaluru")

print(f"முதல் நகரம்: {top_cities[0]}")  # Output: Chennai
print(f"மூன்றாவது நகரம்: {top_cities[2]}") # Output: Delhi
```

எதிர்மறை இன்டெக்ஸிங் (Negative Indexing): பின்னோக்கிப் பார்த்தல்

ஸ்ட்ரிங்குகள் மற்றும் பட்டியல்களைப் போலவே, டியூபிள்களிலும் பின்னோக்கி இன்டெக்ஸ் செய்யலாம்.

- `top_cities[-1]` என்றால் கடைசி உறுப்பு (“Bengaluru”) கிடைக்கும்.
- `top_cities[-2]` என்றால் கடைசிக்கு முந்தைய உறுப்பு (“Delhi”) கிடைக்கும்.

Python

```
print(f"கடைசி நகரம்: {top_cities[-1]}")  # Output: Bengaluru
print(f"கடைசிக்கு முந்தைய நகரம்: {top_cities[-2]}") # Output: Delhi
```

4.3 டியூபிள்ஸ் ஏன் ‘மாறாதவை’ (Immutable)? - உறுதியான பாதுகாப்புக் கவசம்!

இது டியூபிள்ஸ் பற்றிய மிக முக்கியமான அம்சம், மற்றும் பட்டியல்களிலிருந்து (Lists) இவை வேறுபடும் முக்கியக் காரணம்! பைதானில் ஒரு டியூபிளை ஒருமுறை உருவாக்கிவிட்டால், அதன் உள்ளே இருக்கும் உறுப்புகளை நேரடியாக மாற்றவோ, நீக்கவோ, புதிதாகச் சேர்க்கவோ **முடியாது**.

Python

```
my_tuple = ("apple", "banana", "cherry")

# my_tuple[0] = "orange" # இது பிழை ஏற்படுத்தும்! (TypeError)
# my_tuple.append("grape") # இதுவும் பிழை ஏற்படுத்தும்! (AttributeError)
# del my_tuple[1]          # இதுவும் பிழை ஏற்படுத்தும்! (TypeError)
```

“அப்படியானால் டியூபிளைப் புதுப்பிக்கவே முடியாதா?” என்று நீங்கள் கேட்கலாம். ஒரு டியூபிளை நேரடியாக மாற்ற முடியாது என்றாலும், ஏற்கனவே உள்ள டியூபிள்களின் உறுப்புகளைப் பயன்படுத்தி ஒரு **புதிய டியூபிளை** உருவாக்கலாம்.

Python

```
original_tuple = (1, 2, 3, 4)
# 3-ஐ 5 ஆக மாற்ற வேண்டும் என வைத்துக் கொள்வோம்
# நேரடியாக மாற்ற முடியாது, ஆனால் ஒரு புதிய டியூபிளை உருவாக்கலாம்
new_tuple = original_tuple[:2] + (5,) + original_tuple[3:]
print(f"Original tuple: {original_tuple}") # Output: (1, 2, 3, 4)
print(f"New tuple: {new_tuple}")           # Output: (1, 2, 5, 4)
```

‘மாறாதவை’ என்பதன் முக்கியத்துவம் என்ன?

இந்த `immutable` தன்மை, வெறும் ஒரு வரம்பு மட்டுமல்ல; இது டியூபிள்ஸ் வழங்கும் ஒரு பெரிய நன்மை! இது, சில குறிப்பிட்ட சூழ்நிலைகளில் பட்டியல்களை விட டியூபிள்கள் ஏன் சிறந்தவை என்பதை விளக்குகிறது:

- **தரவு ஒருமைப்பாடு (Data Integrity):** நீங்கள் மாற்றப்படவே கூடாத தகவல்களை (எ.கா: பிறந்த தேதி, ஒரு

பொருளின் ஐடி எண்) சேமிக்கும்போது, டியூபிள்ஸ் மிகச் சிறந்த தேர்வு. இது எதிர்பாராத மாற்றங்களிலிருந்து தரவைப் பாதுகாக்கும் ஒரு ‘உறுதியான பாதுகாப்புக் கவசம்’ போல செயல்படுகிறது.

- **செயல்பாடுகளுக்கான தரவு பாதுகாப்பு (Safe Function Arguments):** ஒரு function-க்கு (செயல்பாட்டிற்கு) ஒரு டியூபிளை உள்ளீடாகக் கொடுக்கும்போது, அந்த function அந்த டியூபிளை மாற்றாது என்று நீங்கள் உறுதியாக நம்பலாம். இது குறியீட்டை (code) மிகவும் நம்பகத்தன்மை உடையதாக மாற்றுகிறது.
- **அகராதி விசைகளாகப் பயன்படுத்தல் (Usable as Dictionary Keys):** பைதானில், அகராதிகளின் (Dictionaries) விசைகள் (keys) ‘மாறாத’ தரவு வகைகளாக மட்டுமே இருக்க முடியும். ஒரு `list` மாறக்கூடியது என்பதால் அதை Dictionary Key ஆகப் பயன்படுத்த முடியாது. ஆனால், `tuple` மாறாதது என்பதால், அதை Dictionary Key ஆகப் பயன்படுத்தலாம்! இது ஒரு முக்கியமான திறன்.
- **வேகம் (Speed - ஒரு சிறிய நன்மை):** சில சமயங்களில், டியூபிள்கள் பட்டியல்களை விடச் சற்றே வேகமாகச் செயல்படலாம், குறிப்பாகப் பெரிய தரவுகளைப் படிக்கும்போது. ஆனால், இது முதன்மையான காரணம் அல்ல; முக்கிய நன்மை அதன் ‘மாறாத’ தன்மைதான்.

4.4 டியூபிள்ஸ் உடன் செய்யக்கூடிய செயல்பாடுகள் (Tuple Operations)

டியூபிள்ஸ் ‘மாறாதவை’ என்றாலும், நீங்கள் அவற்றின் உறுப்புகளை நேரடியாக மாற்ற முடியாது என்றாலும், அவற்றைக் கொண்டு பல பயனுள்ள செயல்பாடுகளைச் செய்ய முடியும்:

- **ஸ்லைசிங் (Slicing):** பட்டியல்களைப் போலவே, டியூபிள்களிலும் ஸ்லைசிங் மூலம் ஒரு குறிப்பிட்ட பகுதியைப் பிரித்தெடுக்கலாம். இது ஒரு புதிய டியூபிளை உருவாக்கும்.

  Python

  ```
  my_tuple = (10, 20, 30, 40, 50)
  subset = my_tuple[1:4] # Index 1-ல் இருந்து 4 வரை (4 சேர்க்கப்படாது)
  print(f"ஸ்லைஸ் செய்யப்பட்ட டியூபிள்: {subset}") # Output: (20, 30, 40)
  ```

- **இணைத்தல் (Concatenation - `+`):** இரண்டு அல்லது அதற்கு மேற்பட்ட டியூபிள்களை `+` ஆப்பரேட்டரைப் பயன்படுத்தி இணைத்து ஒரு புதிய டியூபிளை உருவாக்கலாம்.

  Python

  ```
  tuple1 = (1, 2)
  tuple2 = (3, 4)
  combined_tuple = tuple1 + tuple2
  print(f"இணைக்கப்பட்ட டியூபிள்: {combined_tuple}") # Output: (1, 2, 3, 4)
  ```

- **மீண்டும் மீண்டும் பயன்படுத்துதல் (Repetition - `\*`):** ஒரு டியூபிளை பலமுறை மீண்டும் பயன்படுத்த `*` ஆப்பரேட்டரைப் பயன்படுத்தலாம்.

  Python

  ```
  repeated_tuple = ("hi",) * 3
  print(f"மீண்டும் பயன்படுத்தப்பட்டது: {repeated_tuple}") # Output: ('hi', 'hi', 'hi')
  ```

- **நீளம் (Length - `len()`):** ஒரு டியூபிளில் எத்தனை உறுப்புகள் உள்ளன என்பதைக் கண்டறிய `len()` செயல்பாட்டைப் பயன்படுத்தலாம்.

  Python

```
my_tuple = (1, 2, 3, 4, 5)
print(f"டியூபிளின் நீளம்: {len(my_tuple)}") # Output: 5
```

- **உறுப்பு உள்ளதா எனச் சரிபார்த்தல் (Checking for an Element - `in`):** ஒரு குறிப்பிட்ட உறுப்பு டியூபிளில் உள்ளதா என்று `in` ஆப்பரேட்டரைப் பயன்படுத்திச் சரிபார்க்கலாம். இது `True` அல்லது `False` என்ற Boolean மதிப்பை வழங்கும்.

  Python

  ```
  fruits = ("apple", "banana", "cherry")
  print(f"'banana' டியூபிளில் உள்ளதா? {"banana" in fruits}") # Output: True
  ```

- **உறுப்பு எத்தனை முறை உள்ளது (Count - `count()`):** ஒரு குறிப்பிட்ட உறுப்பு டியூபிளில் எத்தனை முறை தோன்றுகிறது என்று `count()` method மூலம் எண்ணிச் சொல்லும்.

  Python

  ```
  numbers = (1, 2, 2, 3, 2)
  print(f"2 எத்தனை முறை உள்ளது? {numbers.count(2)}") # Output: 3
  ```

- **உறுப்பின் இன்டெக்ஸ் (Index - `index()`):** ஒரு குறிப்பிட்ட மதிப்பின் இன்டெக்ஸைக் கண்டறிய `index()` method பயன்படுத்தலாம். மதிப்பு பட்டியலில் இல்லாவிட்டால் பிழை ஏற்படும்.

  Python

  ```
  planets = ("Mercury", "Venus", "Earth", "Mars")
  earth_index = planets.index("Earth")
  print(f"'Earth' இன்டெக்ஸ்: {earth_index}") # Output: 2
  ```

4.5 டியூபிள் அன்பேக்கிங் (Tuple Unpacking): எளிமையான பிரித்தெடுத்தல்

டியூபிள் அன்பேக்கிங் என்பது பைதான் வழங்கும் ஒரு அழகான வசதி. ஒரு டியூபிளில் உள்ள உறுப்புகளை ஒரே நேரத்தில் பல மாறிகளுக்கு (variables) பிரித்து ஒதுக்குவது இதன் மூலம் சாத்தியம். இது குறியீட்டை மிகவும் படிக்க எளிதாக்குகிறது.

Python

```
# ஒரு டியூபிளில் உள்ள பெயரை, வயதை, நகரத்தைப் பிரித்தெடுக்கிறோம்
person_details = ("அறிவு", 32, "சென்னை")

name, age, city = person_details # டியூபிள் அன்பேக்கிங்!

print(f"பெயர்: {name}")  # Output: பெயர்: அறிவு
print(f"வயது: {age}")     # Output: வயது: 32
print(f"நகரம்: {city}")  # Output: நகரம்: சென்னை
```

இது, ஒரே வரிசையில் பல மாறிகளுக்கு மதிப்புகளை ஒதுக்குவதற்கும் பயன்படுகிறது:

Python

```
x, y = 10, 20 # இதுவும் ஒரு மறைமுக டியூபிள் அன்பேக்கிங் தான்!
print(f"x: {x}, y: {y}") # Output: x: 10, y: 20
```

டியூபிள்ஸ், அதன் ‘மாறாத’ தன்மையால், தரவு ஒருமைப்பாடு மற்றும் குறியீட்டின் நம்பகத்தன்மையை உறுதி செய்கின்றன. இது பட்டியல்களைப் போலவே முக்கியமான ஒரு தரவு வகையாகும்.

பட்டியல்கள் மற்றும் டியூபிள்ஸ் பற்றிய இந்த விரிவான புரிதல், பைதானில் தரவுகளைச் சேமித்து, நிர்வகிக்கும் உங்கள் திறனை மேம்படுத்தும். அடுத்த பகுதியில், பைதானின் மற்றொரு முக்கியமான தரவு வகையைப் பற்றிப் பார்ப்போம் – அது ஒரே மாதிரியான உறுப்புகளை மட்டும் கொண்ட ஒரு ‘தொகுப்பு’ ஆகும்.

5. பைதானில் தொகுப்புகள் (Sets)

எண்களைக் கண்டோம், எழுத்துகளுடன் விளையாடினோம், வரிசையான பட்டியல்களையும் (Lists), மாறாத டியூபிள்களையும் (Tuples) உருவாக்கினோம். இப்போது, நாம் சேகரிக்கும் தரவுகளில், **ஒரே மதிப்பை மீண்டும் மீண்டும் சேர்க்க விரும்பாத** சூழ்நிலைகள் வரலாம், இல்லையா?

உதாரணமாக:

- ஒரு நிகழ்வில் கலந்துகொண்டவர்களின் பெயர் பட்டியல் – ஒரே நபர் இரண்டு முறை வந்தாலும் ஒரு முறைதானே கணக்கில் கொள்ள வேண்டும்?
- நீங்கள் கற்றுக்கொண்ட பைதான் பாடங்களின் பட்டியல் – ஒரு பாடத்தை ஒரு முறை கற்றுக்கொண்டால் போதும்.
- ஒரு வலைத்தளத்திற்கு வந்த தனிப்பட்ட பயனர்களின் IP முகவரிகள் – ஒரே முகவரி மீண்டும் வந்தாலும், ஒரு தனிப்பட்ட பயனராகத்தான் கணக்கிட வேண்டும்.
- இரண்டு வெவ்வேறு தயாரிப்புப் பட்டியலில் உள்ள பொதுவான தயாரிப்புகளைக் கண்டறிதல்.

இந்தச் சூழ்நிலைகளில், பைதான் ஒரு சிறப்புத் தரவு வகையை வழங்குகிறது – அதுதான் **தொகுப்புகள் (Sets)**. ஒரு தொகுப்பு என்பது, ஒரு பள்ளியின் மாணவர்கள் சங்கத்தைப் போல, ஒவ்வொரு உறுப்பினரும் தனித்துவமானவர்களாக இருக்க வேண்டும், யாரும் திரும்ப வரக்கூடாது! இந்த அத்தியாயத்தில், தொகுப்புகள் என்றால் என்ன, அவற்றின் தனித்துவமான பண்புகள், எப்படி உறுப்புகளைச் சேர்ப்பது/நீக்குவது, மற்றும் அவற்றின் சக்திவாய்ந்த கணிதச் செயல்பாடுகளைப் பற்றி விரிவாகக் கற்கப் போகிறோம்.

உங்கள் பைதான் பயணத்தின் அடுத்த தனித்துவமான அடியை எடுத்து வைப்போம்!

5.1 தொகுப்புகள் என்றால் என்ன? (What are Sets?)

எளிமையாகச் சொன்னால், **தொகுப்பு (Set)** என்பது பலதரப்பட்ட **தனித்துவமான (unique)** மதிப்புகளின் **வரிசைப்படுத்தப்படாத (unordered)** மற்றும் **மாற்றக்கூடிய (mutable)** ஒரு தொகுப்பு. பட்டியல்கள் மற்றும் டியூபிள்களைப் போலல்லாமல், தொகுப்புகள் உறுப்புகளின் வரிசையை நினைவில் வைத்துக் கொள்வதில்லை. மேலும், மிக முக்கியமாக, ஒரு தொகுப்பில் ஒரே மதிப்பு இரண்டு முறை இருக்கவே முடியாது! நீங்கள் ஒரு மதிப்பை இரண்டு முறை சேர்த்தாலும், அது ஒரு முறையாக மட்டுமே சேமிக்கப்படும்.

பைதானில் ஒரு தொகுப்பை உருவாக்க, மதிப்புகளைச் **சுருள் அடைப்புக்குறிகளுக்குள் (`{}`)** இட்டு, ஒவ்வொரு மதிப்புக்கும் இடையில் காற்புள்ளி (`,`) இட வேண்டும்.

Python

```
# எண்களின் தொகுப்பு - கவனியுங்கள், 2 என்ற எண் இரண்டு முறை சேர்த்தாலும் ஒரு முறைதான் சேமிக்கப்படும்
unique_numbers = {1, 2, 3, 2, 4, 5}
print(f"தனித்துவ எண்களின் தொகுப்பு: {unique_numbers}") # Output: {1, 2, 3, 4, 5} (வரிசை மாறலாம்)

# எழுத்துகளின் தொகுப்பு
unique_colors = {"red", "green", "blue", "red"}
print(f"தனித்துவ வண்ணங்களின் தொகுப்பு: {unique_colors}") # Output: {'red', 'green', 'blue'} (வரிசை மாறலாம்)

# கலப்புத் தரவு வகைகளின் தொகுப்பு (ஆனால், உறுப்புகள் மாறாதவையாக இருக்க வேண்டும்!)
mixed_set = {10, "Python", 3.14}
print(f"கலப்புத் தொகுப்பு: {mixed_set}") # Output: {3.14, 10, 'Python'} (வரிசை மாறலாம்)
```

ஒரு வெற்றுத் தொகுப்பை உருவாக்குதல்: ஒரு சிறிய நுணுக்கம்!

ஒரு **வெற்றுத் தொகுப்பை (empty set)** உருவாக்க, `set()` என்ற செயல்பாட்டைப் பயன்படுத்த வேண்டும். வெறுமனே `{}` என்று பயன்படுத்தினால், பைதான் அதை ஒரு வெற்று அகராதியாக (empty dictionary) கருதும்! இது ஒரு முக்கியமான வேறுபாடு.

Python

```
empty_set = set() # இதுதான் ஒரு வெற்றுத் தொகுப்பை உருவாக்கும் சரியான வழி
print(f"வெற்றுத் தொகுப்பு: {empty_set}") # Output: set()
print(f"வகை: {type(empty_set)}")       # Output: <class 'set'>

empty_dict = {} # இது ஒரு வெற்று அகராதியை உருவாக்கும்
print(f"வெற்று அகராதி: {empty_dict}") # Output: {}
print(f"வகை: {type(empty_dict)}")      # Output: <class 'dict'>
```

5.2 தொகுப்புகளின் முக்கியப் பண்புகள் (Key Characteristics of Sets)

தொகுப்புகள், பட்டியல்கள் மற்றும் டியூபிள்களிலிருந்து சில தனித்துவமான பண்புகளைக் கொண்டுள்ளன:

- **வரிசைப்படுத்தப்படாதவை (Unordered):** தொகுப்புகளுக்கு எந்தவிதமான உள் வரிசையும் கிடையாது. நீங்கள் ஒரு தொகுப்பை அச்சிடும்போது, அதன் உறுப்புகள் நீங்கள் சேர்த்த வரிசையில் வராமல் போகலாம். இதனால், பட்டியல்கள் மற்றும் டியூபிள்களைப் போல இன்டெக்ஸிங் (`my_set[0]`) அல்லது ஸ்லைசிங் (`my_set[1:3]`) மூலம் உறுப்புகளை அணுக முடியாது.
- **தனித்துவமான உறுப்புகள் (Unique Elements Only):** ஒரு தொகுப்பின் மிக முக்கியமான பண்பு இது. நீங்கள் எத்தனை முறை ஒரே மதிப்பைச் சேர்க்க முயற்சி செய்தாலும், தொகுப்பு அதை ஒரு முறையாக மட்டுமே சேமிக்கும். இது 'டூப்ளிகேட்' (duplicate) மதிப்புகளை வடிகட்ட மிகச் சிறந்த வழியாகும்.
- **மாற்றக்கூடியவை (Mutable - தொகுப்பை மாற்றலாம்):** ஒரு தொகுப்பை உருவாக்கிய பிறகு, அதில் புதிய உறுப்புகளைச் சேர்க்கலாம் (`add()`) அல்லது ஏற்கனவே உள்ளவற்றை நீக்கலாம் (`remove()`, `discard()`).
- **மாறாத உறுப்புகள் (Immutable Elements - உறுப்புகளின் தன்மை):** ஒரு தொகுப்பிற்குள் நீங்கள் சேமிக்கும் உறுப்புகள் **மாறாதவையாக (immutable)** இருக்க வேண்டும். அதாவது, எண்கள், எழுத்துகள், டியூபிள்கள் போன்றவற்றைச் சேமிக்கலாம். ஆனால், பட்டியல்கள் அல்லது வேறு தொகுப்புகளை நேரடியாக ஒரு தொகுப்பிற்குள் சேமிக்க முடியாது, ஏனெனில் அவை மாறக்கூடியவை.

5.3 தொகுப்புகளுடன் உறுப்புகளைச் சேர்த்தல் மற்றும் நீக்குதல் (Adding and Removing Elements)

தொகுப்புகள் மாற்றக்கூடியவை என்பதால், நாம் உறுப்புகளைச் சேர்க்கலாம் மற்றும் நீக்கலாம்:

உறுப்புகளைச் சேர்த்தல் (Adding Elements)

- `add(item)`: தொகுப்பில் ஒரு புதிய உறுப்பைச் சேர்க்கும். உறுப்பு ஏற்கனவே தொகுப்பில் இருந்தால், எந்த மாற்றமும் நிகழாது.

Python

```python
my_courses = {"Python Basics", "Data Science"}
print(f"அசல் பாடங்கள்: {my_courses}")

my_courses.add("Web Development") # புதிய பாடம் சேர்த்தல்
print(f"add செய்த பின்: {my_courses}") # Output: {'Python Basics', 'Data Science', 'Web Development'} (வரிசை மாறலாம்)

my_courses.add("Python Basics") # ஏற்கனவே உள்ளதை மீண்டும் சேர்த்தல் - எந்த மாற்றமும் இருக்காது
print(f"மீண்டும் add செய்த பின்: {my_courses}") # Output: {'Python Basics', 'Data Science', 'Web Development'} (வரிசை மாறாது)
```

உறுப்புகளை நீக்குதல் (Removing Elements)

- `remove(value)`: தொகுப்பிலிருந்து ஒரு குறிப்பிட்ட மதிப்பை நீக்கும். அந்த மதிப்பு தொகுப்பில் இல்லாவிட்டால், `KeyError` என்ற பிழையை ஏற்படுத்தும்.
- `discard(value)`: தொகுப்பிலிருந்து ஒரு குறிப்பிட்ட மதிப்பை நீக்கும். `remove()` போலவே, ஆனால் அந்த மதிப்பு தொகுப்பில் இல்லாவிட்டால் எந்தப் பிழையையும் ஏற்படுத்தாது. இது `remove()` ஐ விடப் பாதுகாப்பானது.
- `pop()`: தொகுப்பிலிருந்து ஒரு சீரற்ற உறுப்பை நீக்கி, நீக்கப்பட்ட உறுப்பை வழங்கும். தொகுப்புகளுக்கு வரிசை இல்லை என்பதால், எந்த உறுப்பு நீக்கப்படும் என்று கணிக்க முடியாது.
- `clear()`: தொகுப்பின் அனைத்து உறுப்புகளையும் நீக்கும், ஆனால் தொகுப்பை அழிக்காது.

Python

```python
registered_users = {"அறிவு", "மதி", "கனி", "அறிவு"} # 'அறிவு' ஒருமுறைதான் இருக்கும்
print(f"அசல் பயனர்கள்: {registered_users}") # Output: {'அறிவு', 'கனி', 'மதி'} (வரிசை மாறலாம்)

# remove(): மதிப்பின் மூலம் நீக்குதல்
registered_users.remove("மதி")
print(f"remove செய்த பின்: {registered_users}") # Output: {'அறிவு', 'கனி'} (வரிசை மாறலாம்)

# registered_users.remove("புதியவர்") # ❌ இது KeyError ஏற்படுத்தும், ஏனெனில் 'புதியவர்' இல்லை

# discard(): பிழை இல்லாமல் நீக்குதல்
registered_users.discard("கனி")
print(f"discard செய்த பின்: {registered_users}") # Output: {'அறிவு'}
```

```
registered_users.discard("இல்லாதவர்") # இது பிழை ஏற்படுத்தாது
print(f"இல்லாதவரை discard செய்த பின்: {registered_users}") # Output: {'அறிவு'}

# pop(): சீரற்ற உறுப்பை நீக்குதல்
random_user = registered_users.pop() # 'அறிவு' நீக்கப்படலாம்
print(f"pop செய்த பின்: {registered_users} (நீக்கப்பட்டது: {random_user})") # Output: set() (நீக்கப்பட்டது: அறிவு)

# clear(): அனைத்து உறுப்புகளையும் நீக்குதல்
my_set = {1, 2, 3}
my_set.clear()
print(f"clear செய்த பின்: {my_set}") # Output: set()
```

5.4 தொகுப்பு கணிதச் செயல்பாடுகள் (Set Mathematical Operations): தனித்துவமான சக்தி

தொகுப்புகள், கணிதத்தில் நாம் பார்க்கும் 'செட் தியரி' (Set Theory) செயல்பாடுகளை நேரடியாகச் செய்ய அனுமதிக்கின்றன. இது தரவு பகுப்பாய்வு மற்றும் வடிகட்டலுக்கு மிகவும் சக்திவாய்ந்த கருவியாகும்.

- **யூனியன் (Union) - `|` அல்லது `union()`:** இரண்டு அல்லது அதற்கு மேற்பட்ட தொகுப்புகளில் உள்ள அனைத்துத் தனித்துவமான உறுப்புகளையும் ஒன்றிணைக்கும். இது இரண்டு குழுக்களின் மொத்த உறுப்பினர்களைக் கண்டறிவது போல.

  Python

  ```
  students_in_class_A = {"அரவிந்த்", "பாலா", "சித்ரா"}
  students_in_class_B = {"சித்ரா", "தீபா", "எழில்"}

  all_students_union = students_in_class_A.union(students_in_class_B)
  # அல்லது all_students_union = students_in_class_A | students_in_class_B
  print(f"மொத்த மாணவர்கள் (யூனியன்): {all_students_union}")
  # Output: {'அரவிந்த்', 'பாலா', 'சித்ரா', 'தீபா', 'எழில்'} (வரிசை மாறலாம்)
  ```

- **இன்டர்செக்ஷன் (Intersection) - `&` அல்லது `intersection()`:** இரண்டு அல்லது அதற்கு மேற்பட்ட தொகுப்புகளில் உள்ள **பொதுவான** உறுப்புகளை மட்டும் வழங்கும். இது இரண்டு குழுக்களிலும் உள்ள பொதுவான உறுப்பினர்களைக் கண்டறிவது போல.

  Python

  ```
  common_students_intersection = students_in_class_A.intersection(students_in_class_B)
  # அல்லது common_students_intersection = students_in_class_A & students_in_class_B
  print(f"பொதுவான மாணவர்கள் (இன்டர்செக்ஷன்): {common_students_intersection}")
  # Output: {'சித்ரா'}
  ```

- **டிஃபரன்ஸ் (Difference) - `-` அல்லது `difference()`:** முதல் தொகுப்பில் மட்டும் இருக்கும் உறுப்புகளை, இரண்டாவது தொகுப்பில் இல்லாத உறுப்புகளை வழங்கும். இது ஒரு குழுவில் இருந்து, மற்றொரு குழுவில் உள்ளவர்களை நீக்கிய பின் எஞ்சியவர்களைக் கண்டறிவது போல.

  Python

```
students_only_in_A = students_in_class_A.difference(students_in_class_B)
# அல்லது students_only_in_A = students_in_class_A - students_in_class_B
print(f"A-ல் மட்டும் உள்ள மாணவர்கள் (டிஃபரன்ஸ்): {students_only_in_A}")
# Output: {'அரவிந்த்', 'பாலா'}

students_only_in_B = students_in_class_B.difference(students_in_class_A)
print(f"B-ல் மட்டும் உள்ள மாணவர்கள் (டிஃபரன்ஸ்): {students_only_in_B}")
# Output: {'தீபா', 'எழில்'}
```

- **சிமெட்ரிக் டிஃபரன்ஸ் (Symmetric Difference) - `^` அல்லது `symmetric_difference()`:** இரண்டு தொகுப்புகளிலும் **பொதுவாக இல்லாத** (ஒன்றில் மட்டுமே இருக்கும்) அனைத்து உறுப்புகளையும் வழங்கும். இது இரண்டு குழுக்களிலும் உள்ள தனிப்பட்ட உறுப்பினர்களைக் கண்டறிவது போல.

  Python

  ```
  unique_to_either_class =
  students_in_class_A.symmetric_difference(students_in_class_B)
  # அல்லது unique_to_either_class = students_in_class_A ^ students_in_class_B
  print(f"இரண்டிலும் தனித்துவமான மாணவர்கள் (சிமெட்ரிக் டிஃபரன்ஸ்): {unique_to_either_class}")
  # Output: {'அரவிந்த்', 'பாலா', 'தீபா', 'எழில்'}
  ```

- **சப்செட்/சூப்பர்செட் (Subset/Superset): `issubset()`, `issuperset()`:** ஒரு தொகுப்பு மற்றொரு தொகுப்பின் பகுதியா (subset) அல்லது மற்றொரு தொகுப்பு ஒரு பெரிய பகுதியா (superset) என்று சரிபார்க்கும்.

  Python

  ```
  set1 = {1, 2}
  set2 = {1, 2, 3, 4}

  print(f"{set1} என்பது {set2}-இன் சப்செட்டா? {set1.issubset(set2)}") # Output: True
  print(f"{set2} என்பது {set1}-இன் சூப்பர்செட்டா? {set2.issuperset(set1)}") # Output: True
  ```

5.5 தொகுப்புகளின் பயன்பாடுகள் (Use Cases of Sets)

தொகுப்புகள் சில குறிப்பிட்ட சூழ்நிலைகளில் மிக சக்திவாய்ந்தவை:

- **டூப்ளிகேட் மதிப்புகளை நீக்குதல் (Removing Duplicates):** ஒரு பட்டியலில் உள்ள டூப்ளிகேட் மதிப்புகளை மிக எளிதாகவும், விரைவாகவும் நீக்க, பட்டியலை ஒரு தொகுப்பாக மாற்றி, மீண்டும் பட்டியலாக மாற்றலாம்.

  Python

  ```
  numbers_with_duplicates = [1, 2, 2, 3, 4, 4, 5]
  unique_numbers_list = list(set(numbers_with_duplicates))
  print(f"டூப்ளிகேட் நீக்கப்பட்ட பட்டியல்: {unique_numbers_list}") # Output: [1, 2, 3, 4, 5]
  (வரிசை மாறலாம்)
  ```

- **உறுப்பினர் சரிபார்ப்பு (Membership Testing):** ஒரு பெரிய தரவுத் தொகுப்பில் ஒரு குறிப்பிட்ட உறுப்பு உள்ளதா என்று மிக விரைவாகச் சரிபார்க்க தொகுப்புகள் மிகவும் திறமையானவை.

  Python

```
registered_ids = {101, 105, 120, 130}
check_id = 105
if check_id in registered_ids:
    print(f"ID {check_id} பதிவு செய்யப்பட்டுள்ளது.")
else:
    print(f"ID {check_id} பதிவு செய்யப்படவில்லை.")
```

- **கணிதச் செட் செயல்பாடுகள்:** இரண்டு தரவுத் தொகுதிகளுக்கு இடையில் பொதுவான பகுதிகள், தனித்துவமான பகுதிகள், அல்லது மொத்த உறுப்புகள் போன்றவற்றை கண்டறிய. (உதாரணமாக, ஒரு இணையதளத்திற்குச் சென்ற பயனர்கள், மற்றும் ஒரு பொருளை வாங்கிய பயனர்களுக்கு இடையே உள்ள பொதுவான பயனர்களைக் கண்டறிதல்).

தொகுப்புகள் (Sets) என்பவை பைதான் புரோகிராமிங்கில் தனித்துவமான மற்றும் சக்திவாய்ந்த ஒரு தரவு வகையாகும். அவற்றின் 'தனித்துவமான உறுப்புகள்' என்ற பண்பு மற்றும் கணிதச் செயல்பாடுகள், தரவுகளை வடிகட்டவும், பகுப்பாய்வு செய்யவும், திறமையாக நிர்வகிக்கவும் உதவுகின்றன.

தொகுப்புகள் பற்றிய இந்த விரிவான புரிதல், உங்கள் பைதான் திறமைகளை மேலும் ஒரு படி உயர்த்தும். அடுத்த பகுதியில், பைதானின் மற்றொரு அத்தியாவசியத் தரவு வகையைப் பற்றிப் பார்ப்போம் – அது 'விசை-மதிப்பு' ஜோடிகளைக் கொண்ட ஒரு 'அகராதி' ஆகும்!

6. பைதானில் அகராதிகள் (Dictionaries)

நாம் இதுவரை எண்களை (integers, floats, complex), எழுத்துகளை (strings), மற்றும் வரிசைப்படுத்தப்பட்ட தொகுப்புகளான பட்டியல்கள் (lists) மற்றும் மாறாத டியூபிள்களை (tuples) கண்டோம். ஆனால், சில சமயங்களில், நமக்குத் தகவல்களை ஒரு குறிப்பிட்ட வரிசையில் சேமிப்பதுடன், **பெயர் சொல்லி அழைக்கக்கூடிய ஒரு அடையாளத்துடன்** சேமிக்க வேண்டியிருக்கும், இல்லையா?

உதாரணமாக:

- உங்கள் நண்பரின் தகவல்கள்: பெயர், வயது, தொலைபேசி எண், மின்னஞ்சல் – இவை ஒவ்வொன்றையும் ஒரு 'பெயர்' அல்லது 'அடையாளம்' கொண்டு அழைக்க வேண்டும்.
- ஒரு பொருளின் விவரங்கள்: தயாரிப்பு பெயர், விலை, இருப்பு, நிறம் – இவை ஒவ்வொன்றும் தனித்தனி தலைப்புகள்.
- ஒரு நகரத்தின் மக்கள் தொகை, பரப்பளவு, மாநிலம் போன்ற விவரங்கள்.

இந்த எல்லா இடங்களிலும் நாம் ஒரு **'விசை' (key)** மற்றும் அந்த விசைக்குரிய **'மதிப்பு' (value)** என்ற ஜோடியாகத் தகவல்களைச் சேமிக்கிறோம். கணினி உலகில், இந்த வகையான 'பெயர் சொல்லி அழைக்கும்' தகவல் பெட்டகங்களைச் சேமிக்கப் பைதான் ஒரு சூப்பர் பவரை வழங்குகிறது – அதுதான் **அகராதிகள் (Dictionaries)**.

ஒரு அகராதி என்பது பைதான் புரோகிராமிங்கில் மிக முக்கியமான மற்றும் அன்றாடப் பயன்பாட்டில் உள்ள ஒரு தரவு வகையாகும். இது தகவல்களை மிகவும் கட்டமைக்கப்பட்ட, எளிதில் அணுகக்கூடிய வழியில் சேமிக்க உதவுகிறது. இந்த அத்தியாயத்தில், அகராதிகள் என்றால் என்ன, அவற்றின் தனித்துவமான பண்புகள், எப்படி தகவல்களைச் சேர்ப்பது/நீக்குவது/மாற்றுவது, மற்றும் அதன் சக்திவாய்ந்த பயன்பாடுகளைப் பற்றி விரிவாகக் கற்கப் போகிறோம்.

உங்கள் பைதான் பயணத்தின் அடுத்த, மிகவும் நடைமுறைக்கு உகந்த அடியை எடுத்து வைப்போம்!

6.1 அகராதிகள் என்றால் என்ன? (What are Dictionaries?)

எளிமையாகச் சொன்னால், **அகராதி (Dictionary)** என்பது பல **‘விசை-மதிப்பு ஜோடிகளின்’ (key-value pairs)** ஒரு வரிசைப்படுத்தப்படாத (unordered) மற்றும் மாற்றக்கூடிய (mutable) தொகுப்பு. இதை நாம் பயன்படுத்தும் ஒரு சாதாரண அகராதியைப் (dictionary - புத்தகம்) போலவே கற்பனை செய்து கொள்ளலாம். ஒரு வார்த்தையை (key) நாம் தேடினால், அதற்கான அர்த்தம் (value) நமக்குக் கிடைக்கும்.

பைதானில் ஒரு அகராதியை உருவாக்க, **சுருள் அடைப்புக்குறிகளுக்குள் (`{}`)** `key: value` ஜோடிகளை இட்டு, ஒவ்வொரு ஜோடிக்கும் இடையில் காற்புள்ளி (`,`) இட வேண்டும்.

Python

```
# ஒரு நபரின் தகவல்களைக் கொண்ட அகராதி
person = {
    "name": "அறிவு",
    "age": 32,
    "city": "சென்னை"
}
print(f"நபரின் தகவல்: {person}")
# Output: {'name': 'அறிவு', 'age': 32, 'city': 'சென்னை'}

# ஒரு பொருளின் தகவல்களைக் கொண்ட அகராதி
product = {
    "product_id": "P001",
    "name": "Python Book",
    "price": 499.99,
    "in_stock": True
}
print(f"பொருளின் தகவல்: {product}")
# Output: {'product_id': 'P001', 'name': 'Python Book', 'price': 499.99, 'in_stock': True}
```

அகராதிகள், பெயர் மூலம் தகவல்களை உடனடியாக அணுக ஒரு சிறந்த வழியாகும்.

6.2 அகராதிகளின் முக்கியப் பண்புகள் (Key Characteristics of Dictionaries)

அகராதிகள், நாம் இதுவரை பார்த்த பட்டியல்கள், டியூபிள்கள் மற்றும் தொகுப்புகளிலிருந்து சில தனித்துவமான பண்புகளைக் கொண்டுள்ளன:

- **விசை-மதிப்பு ஜோடிகள் (Key-Value Pairs):** அகராதியின் ஒவ்வொரு உறுப்பும் ஒரு ‘விசை’ (key) மற்றும் அந்த விசைக்குரிய ‘மதிப்பு’ (value) என்ற இரட்டையாகும்.
- **வரிசைப்படுத்தப்படாதவை (Unordered - ஆனால், Python 3.7+ இல் மாறுபட்டது):** பைதான் 3.7 மற்றும் அதற்குப் பிந்தைய பதிப்புகளில், அகராதிகள் **உள்ளிடப்பட்ட வரிசையை (insertion order)** பராமரிக்கின்றன. அதாவது, நீங்கள் எந்த வரிசையில் உறுப்புகளைச் சேர்த்தீர்களோ, அதே வரிசையிலேயே அவை இருக்கும். இருப்பினும், பட்டியல்களைப் போல இன்டெக்ஸ் மூலம் (`my_dict[0]`) அணுக முடியாது, ஏனெனில் அவை இன்னும் ‘வரிசைப்படுத்தப்பட்ட தொகுப்புகள்’ அல்ல.
- **மாற்றக்கூடியவை (Mutable):** அகராதியை உருவாக்கிய பிறகு, அதில் புதிய விசை-மதிப்பு ஜோடிகளைச் சேர்க்கலாம், ஏற்கனவே உள்ள மதிப்புகளை மாற்றலாம், அல்லது ஜோடிகளை நீக்கலாம். இது பட்டியல்களைப் போலவே நெகிழ்வானது.
- **விசைகள் தனித்துவமானவை (Keys Must Be Unique):** ஒரு அகராதியில் ஒரே விசையை இரண்டு முறை பயன்படுத்த முடியாது. நீங்கள் ஒரு விசையை மீண்டும் பயன்படுத்த முயற்சித்தால், அது ஏற்கனவே உள்ள

விசையின் மதிப்பைOverwrite (மேல் எழுதும்).

Python

```
my_dict = {"name": "அறிவு", "age": 32, "name": "கனி"}
print(my_dict) # Output: {'name': 'கனி', 'age': 32} – 'அறிவு' என்பதற்குப் பதிலாக 'கனி' மாற்றப்பட்டுவிட்டது.
```

- **விசைகள் மாறாதவையாக இருக்க வேண்டும் (Keys Must Be Immutable):** அகராதியின் விசைகளாகப் பயன்படுத்தப்படும் மதிப்புகள் **மாறாதவையாக (immutable)** இருக்க வேண்டும். அதாவது, எழுத்துகள் (strings), எண்கள் (numbers), டியூபிள்கள் (tuples) போன்றவற்றை விசைகளாகப் பயன்படுத்தலாம். ஆனால், பட்டியல்கள் (lists), தொகுப்புகள் (sets) அல்லது மற்ற அகராதிகள் போன்ற மாறக்கூடிய (mutable) தரவு வகைகளை விசைகளாகப் பயன்படுத்த முடியாது. இது ஒரு முக்கியமான கட்டுப்பாடு.

  Python

  ```
  # ஒரு டியூபிளை விசை ஆகப் பயன்படுத்தலாம்
  coordinates_dict = {(10, 20): "Point A", (30, 40): "Point B"}
  print(coordinates_dict)

  # list-ஐ விசை ஆகப் பயன்படுத்த முடியாது (பிழை ஏற்படும்)
  # invalid_dict = {[1, 2]: "List as key"} # ❌ TypeError
  ```

- **மதிப்புகள் எதுவாகவும் இருக்கலாம் (Values Can Be Anything):** அகராதியின் மதிப்புகளாக எந்தத் தரவு வகையையும் சேமிக்கலாம் – எண்கள், எழுத்துகள், பட்டியல்கள், டியூபிள்கள், தொகுப்புகள், ஏன் மற்ற அகராதிகள்கூட!

6.3 அகராதி மதிப்புகளை அணுகுதல் (Accessing Dictionary Values)

அகராதியிலிருந்து ஒரு மதிப்பை எடுக்க, அதன் 'விசை'யைப் பயன்படுத்துவோம். இது ஒரு நூலகத்தில் ஒரு புத்தகத்தை அதன் தலைப்பைக் கொண்டு கண்டுபிடிப்பது போல.

6.3.1 சதுர அடைப்புக்குறிகள் `[]` மூலம் அணுகுதல்

விசையைச் சதுர அடைப்புக்குறிகளுக்குள் (`[]`) இட்டு மதிப்பை அணுகலாம்.

Python

```
student = {
    "name": "ஆகாஷ்",
    "grade": "A",
    "major": "கணினி அறிவியல்"
}

print(f"மாணவர் பெயர்: {student['name']}")  # Output: மாணவர் பெயர்: ஆகாஷ்
print(f"பிரிவு: {student['major']}")      # Output: பிரிவு: கணினி அறிவியல்

# இல்லாத விசையை அணுக முயற்சித்தால்
# print(student['age']) # ❌ KeyError: 'age' – பிழை ஏற்படும்
```

6.3.2 `get()` method மூலம் அணுகுதல் - பாதுகாப்பான அணுகல்

ஒரு விசை அகராதியில் இருக்கிறதா இல்லையா என்று உங்களுக்குத் தெரியாவிட்டால், `get()` method-ஐப் பயன்படுத்துவது மிகவும் பாதுகாப்பானது. விசை இல்லாவிட்டால், `get()` method எந்தப் பிழையையும் ஏற்படுத்தாமல் `None` என்ற மதிப்பைத் திருப்பிக் கொடுக்கும். நீங்கள் ஒரு 'default' மதிப்பையும் கொடுக்கலாம்.

Python

```
print(f"மாணவர் வயது (get method): {student.get('age')}") # Output: None (age என்ற விசை
இல்லை)

# default மதிப்புடன் get()
print(f"மாணவர் வயது (default உடன்): {student.get('age', 'கிடைக்கவில்லை')}") # Output:
கிடைக்கவில்லை
print(f"மாணவர் பெயர் (default உடன்): {student.get('name', 'அறியப்படாதது')}") # Output:
ஆகாஷ்
```

6.4 அகராதியில் உள்ளீடு செய்தல் மற்றும் மாற்றுதல் (Adding and Modifying Key-Value Pairs)

அகராதிகள் மாற்றக்கூடியவை என்பதால், நாம் எளிதாக புதிய விசை-மதிப்பு ஜோடிகளைச் சேர்க்கலாம் அல்லது ஏற்கனவே உள்ளவற்றின் மதிப்பை மாற்றலாம்.

புதிய ஜோடிகளைச் சேர்த்தல் (Adding New Pairs)

புதிய விசையைப் பயன்படுத்தி நேரடியாக மதிப்பை ஒதுக்குவதன் மூலம் புதிய ஜோடிகளைச் சேர்க்கலாம்.

Python

```
my_profile = {} # ஒரு வெற்று அகராதி
print(f"வெற்றுப் ப்ரொஃபைல்: {my_profile}") # Output: {}

my_profile["name"] = "பிரியா"
my_profile["city"] = "பெங்களூரு"
print(f"புதிய விவரங்கள் சேர்த்த பின்: {my_profile}") # Output: {'name': 'பிரியா', 'city':
'பெங்களூரு'}
```

உள்ள மதிப்புகளை மாற்றுதல் (Modifying Existing Values)

ஏற்கனவே உள்ள விசையைப் பயன்படுத்தி புதிய மதிப்பை ஒதுக்குவதன் மூலம், அதன் மதிப்பை மாற்றலாம்.

Python

```
my_profile["city"] = "சென்னை" # 'city'யின் மதிப்பை மாற்றுகிறோம்
print(f"நகரத்தை மாற்றிய பின்: {my_profile}") # Output: {'name': 'பிரியா', 'city': 'சென்னை'}
```

6.5 அகராதியிலிருந்து நீக்குதல் (Removing Key-Value Pairs)

அகராதியிலிருந்து விசை-மதிப்பு ஜோடிகளை நீக்கவும் பல வழிகள் உள்ளன:

- `del statement`: ஒரு குறிப்பிட்ட விசையைப் பயன்படுத்தி ஜோடியை நீக்கும். விசை இல்லாவிட்டால்

  `KeyError` ஏற்படுத்தும்.
- `pop(key)`: ஒரு குறிப்பிட்ட விசையைப் பயன்படுத்தி ஜோடியை நீக்கும் மற்றும் நீக்கப்பட்ட மதிப்பை (value) வழங்கும். விசை இல்லாவிட்டால் பிழை ஏற்படுத்தும். நீங்கள் ஒரு 'default' மதிப்பையும் கொடுக்கலாம்.
- `popitem()`: அகராதியிலிருந்து கடைசியாகச் சேர்க்கப்பட்ட (அல்லது சீரற்ற) விசை-மதிப்பு ஜோடியை நீக்கி, அதை ஒரு டியூபிளாக வழங்கும்.
- `clear()`: அகராதியின் அனைத்து ஜோடிகளையும் நீக்கும், அகராதியை காலியாக்கும்.

Python

```
user_settings = {
    "theme": "dark",
    "notifications": True,
    "language": "தமிழ்",
    "auto_save": False
}
print(f"அசல் அமைப்புகள்: {user_settings}")

# del statement: ஒரு விசையை நீக்குதல்
del user_settings["auto_save"]
print(f"auto_save நீக்கப்பட்ட பின்: {user_settings}") # Output: {'theme': 'dark', 'notifications': True, 'language': 'தமிழ்'}

# pop(key): ஒரு விசையை நீக்கி, அதன் மதிப்பை எடுத்தல்
removed_lang = user_settings.pop("language")
print(f"language நீக்கப்பட்ட பின்: {user_settings} (நீக்கப்பட்டது: {removed_lang})") # Output: {'theme': 'dark', 'notifications': True} (நீக்கப்பட்டது: தமிழ்)

# popitem(): ஒரு சீரற்ற/கடைசி ஜோடியை நீக்குதல்
removed_pair = user_settings.popitem()
print(f"popitem செய்த பின்: {user_settings} (நீக்கப்பட்டது: {removed_pair})") # Output: {'theme': 'dark'} (நீக்கப்பட்டது: ('notifications', True))

# clear(): அனைத்தையும் நீக்குதல்
user_settings.clear()
print(f"clear செய்த பின்: {user_settings}") # Output: {}
```

6.6 அகராதியைச் சுற்றி வலம் வருதல் (Iterating Through Dictionaries)

அகராதியில் உள்ள தகவல்களை அணுகிப் பயன்படுத்த `for` loop-ஐப் பயன்படுத்தலாம்.

- **விசைகளை மட்டும் வலம் வருதல் (Looping through keys):** அகராதியை நேரடியாக `for` loop-ல் பயன்படுத்தினால், அது அதன் விசைகளை (keys) மட்டுமே வழங்கும்.

  Python

```
for key in product:
    print(f"விசை: {key}")
# Output:
# விசை: product_id
# விசை: name
# விசை: price
# விசை: in_stock
```

- **மதிப்புகளை மட்டும் வலம் வருதல் (Looping through values):** `values()` method-ஐப் பயன்படுத்தி மதிப்புகளை மட்டும் அணுகலாம்.

  Python

  ```
  for value in product.values():
      print(f"மதிப்பு: {value}")
  # Output:
  # மதிப்பு: P001
  # மதிப்பு: Python Book
  # மதிப்பு: 499.99
  # மதிப்பு: True
  ```

- **விசை மற்றும் மதிப்பு இரண்டையும் வலம் வருதல் (Looping through key-value pairs):** `items()` method-ஐப் பயன்படுத்தி விசை மற்றும் மதிப்பு இரண்டையும் ஒரே நேரத்தில் அணுகலாம். இது மிகவும் பொதுவான பயன்பாடு.

  Python

  ```
  for key, value in product.items():
      print(f"விசை: {key}, மதிப்பு: {value}")
  # Output:
  # விசை: product_id, மதிப்பு: P001
  # விசை: name, மதிப்பு: Python Book
  # விசை: price, மதிப்பு: 499.99
  # விசை: in_stock, மதிப்பு: True
  ```

6.7 பிற பயனுள்ள அகராதி செயல்பாடுகள் (Other Useful Dictionary Methods)

- `len(dictionary)`: அகராதியில் உள்ள விசை-மதிப்பு ஜோடிகளின் எண்ணிக்கையை வழங்கும்.

  Python

  ```
  print(f"Product அகராதியின் நீளம்: {len(product)}") # Output: 4
  ```

- `key in dictionary`: ஒரு குறிப்பிட்ட விசை அகராதியில் உள்ளதா என்று சரிபார்க்க `in` ஆப்பரேட்டரைப் பயன்படுத்தலாம்.

  Python

```
print(f"'name' என்ற விசை அகராதியில் உள்ளதா? {"name" in product}")    # Output: True
print(f"'stock' என்ற விசை அகராதியில் உள்ளதா? {"stock" in product}") # Output: False
```

- `copy()`: ஒரு அகராதியின் நகலை (shallow copy) உருவாக்கும். நேரடி assignment செய்தால், இரண்டு மாறிகளும் ஒரே அகராதியைப் பார்க்கும், ஆனால் `copy()` ஒரு புதிய அகராதியை உருவாக்கும்.

  Python

  ```
  original_dict = {"a": 1, "b": 2}
  copied_dict = original_dict.copy()
  copied_dict["a"] = 100 # நகலை மாற்றுகிறோம்
  print(f"Original: {original_dict}") # Output: Original: {'a': 1, 'b': 2}
  print(f"Copied: {copied_dict}")     # Output: Copied: {'a': 100, 'b': 2}
  ```

- `fromkeys(iterable, value)`: ஒரு லிஸ்ட் அல்லது டியூபிளில் உள்ள உறுப்புகளைக் கொண்டு புதிய அகராதியின் விசைகளை உருவாக்கி, அனைத்து விசைகளுக்கும் ஒரே மதிப்பை ஒதுக்கும்.

  Python

  ```
  default_value = 0
  new_scores = dict.fromkeys(["math", "science", "history"], default_value)
  print(f"புதிய மதிப்பெண்கள்: {new_scores}") # Output: {'math': 0, 'science': 0,
  'history': 0}
  ```

6.8 அகராதிகளின் பயன்பாடுகள் (Use Cases of Dictionaries)

அகராதிகள், பைதான் புரோகிராமிங்கில் நம்பமுடியாத அளவுக்குப் பலதரப்பட்ட பயன்பாடுகளைக் கொண்டுள்ளன:

- **பதிவுகளைப் பிரதிநிதித்துவப்படுத்துதல் (Representing Records):** ஒரு நபர், ஒரு புத்தகம், ஒரு தயாரிப்பு போன்ற நிஜ உலகப் பொருட்களின் கட்டமைக்கப்பட்ட தகவல்களைச் சேமிக்க மிகச் சிறந்த வழி. (இது JSON தரவு வடிவத்திற்கு மிகவும் நெருக்கமானது).
- **அதிர்வெண்ணைக் கணக்கிடுதல் (Counting Frequencies):** ஒரு வாக்கியத்தில் ஒவ்வொரு வார்த்தையும் எத்தனை முறை வருகிறது, அல்லது ஒரு தரவுத் தொகுப்பில் ஒவ்வொரு உறுப்பும் எத்தனை முறை வருகிறது என்பதைக் கண்டறிய.
- **மேப்பிங் உறவுகள் (Mapping Relationships):** ஒரு தனிப்பட்ட அடையாளங்காட்டிக்கும் (ID) அதனுடன் தொடர்புடைய தகவல்களுக்கும் இடையில் ஒரு உறவை உருவாக்க.
- **அமைப்புகளைச் சேமித்தல் (Storing Configurations):** ஒரு மென்பொருளின் பல்வேறு அமைப்புகள் (settings) அல்லது விருப்பத் தேர்வுகள் (options) போன்றவற்றை அகராதியாகச் சேமிப்பது எளிது.

அகராதிகள் (Dictionaries), பைதான் புரோகிராமிங்கின் மிக முக்கியமான, சக்திவாய்ந்த மற்றும் அன்றாடப் பயன்பாட்டிற்கு அத்தியாவசியமான ஒரு தரவு வகையாகும். 'விசை-மதிப்பு' ஜோடிகளின் மூலம் தகவல்களை ஒழுங்கமைக்கும் அதன் திறன், நீங்கள் சிக்கலான தரவுகளை திறமையாக நிர்வகிக்க உதவுகிறது.

அகராதிகள் பற்றிய இந்த விரிவான புரிதல், உங்கள் பைதான் திறமைகளை மேலும் ஒரு படி உயர்த்தும். அடுத்த பகுதியில், பைதானின் மற்றொரு முக்கியமான தரவு வகையைப் பற்றிப் பார்ப்போம் – அது 'சரி' அல்லது 'தவறு' என்ற இரண்டு மதிப்புகளை மட்டுமே கொண்ட ஒரு அடிப்படை வகை ஆகும்!

7. பைதானில் பூலியன் (Boolean Data Type)

நாம் இதுவரை எண்கள், எழுத்துகள், பட்டியல்கள், டியூபிள்கள், தொகுப்புகள் மற்றும் அகராதிகள் எனப் பலதரப்பட்ட தகவல்களைச் சேமிக்கும் வகைகளைப் பார்த்தோம். ஆனால், கணினியின் உலகம் ஒரு முக்கியமான கேள்விக்கு எப்போதும் பதில் தேடும்: **‘இது உண்மையா அல்லது பொய்யா?’ (True or False?)**

- “பயனர் கடவுச்சொல்லைச் சரியாக உள்ளிட்டுள்ளாரா? ஆம்/இல்லை.”
- “ஒரு பொருளின் இருப்பு உள்ளதா? ஆம்/இல்லை.”
- “இன்று மழை பெய்கிறதா? ஆம்/இல்லை.”
- “நான் வயது வந்தவனா? ஆம்/இல்லை.”

இந்த ‘ஆம்’ அல்லது ‘இல்லை’ என்ற கேள்விகள்தான், கணினி முடிவுகளை எடுக்க அடிப்படையாக அமைகின்றன. ஒரு கணினி நிரலின் ஓட்டத்தைக் கட்டுப்படுத்தவும், நிபந்தனைகளைச் சரிபார்க்கவும், தர்க்கரீதியான செயல்பாடுகளைச் செய்யவும் இந்த ‘உண்மை/பொய்’ என்ற கருத்து மிக முக்கியம். பைதானில், இந்த ‘உண்மை/பொய்’ மதிப்புகளைச் சேமிக்கப் பயன்படும் தரவு வகைதான் **பூலியன் (Boolean)**.

இது பைதானின் மிக அடிப்படையான, ஆனால் சக்திவாய்ந்த தரவு வகைகளில் ஒன்றாகும். ஒரு கணினியின் ‘மூளை’ எப்படி முடிவெடுக்கிறது என்பதைப் புரிந்துகொள்ள, பூலியன் வகையைப் பற்றிய தெளிவான புரிதல் அவசியம். இந்த அத்தியாயத்தில், பூலியன் என்றால் என்ன, அதன் மதிப்புகள் என்ன, எப்படி பூலியன் செயல்பாடுகளைப் பயன்படுத்துவது என்று விரிவாகக் கற்கப் போகிறோம்.

உங்கள் பைதான் பயணத்தின் அடுத்த தர்க்கரீதியான அடியை எடுத்து வைப்போம்!

7.1 பூலியன் என்றால் என்ன? (What are Booleans?)

எளிமையாகச் சொன்னால், **பூலியன் (Boolean)** என்பது இரண்டு சாத்தியமான மதிப்புகளை மட்டுமே கொண்ட ஒரு தரவு வகை: **`True` (உண்மை)** அல்லது **`False` (பொய்)**. இந்த மதிப்புகள் எப்போதும் முதல் எழுத்து பெரியதாக (Capital) இருக்க வேண்டும் என்பதை நினைவில் கொள்ளுங்கள். இவை ‘ஆம்’ அல்லது ‘இல்லை’ என்ற ஒரு கேள்விக்கான திட்டவட்டமான பதில்கள்.

Python

```python
# ஒரு பூலியன் மதிப்பு
is_raining = True
print(f"மழை பெய்கிறதா? {is_raining}") # Output: மழை பெய்கிறதா? True
print(f"வகை: {type(is_raining)}")     # Output: <class 'bool'>

# மற்றொரு பூலியன் மதிப்பு
is_adult = False
print(f"நீங்கள் வயது வந்தவரா? {is_adult}") # Output: நீங்கள் வயது வந்தவரா? False
print(f"வகை: {type(is_adult)}")           # Output: <class 'bool'>
```

`True` மற்றும் `False` என்பவை பைதான் மொழியின் சிறப்புச் சொற்கள். இவற்றை மாறிகளாகவோ அல்லது வேறு எந்த வகையிலோ பயன்படுத்தக் கூடாது.

7.2 ஒப்பீட்டுச் செயல்பாடுகள் (Comparison Operators): கேள்விகள் கேட்பது!

பூலியன் மதிப்புகளை உருவாக்க நாம் பெரும்பாலும் **ஒப்பீட்டுச் செயல்பாடுகளை (Comparison Operators)** பயன்படுத்துவோம். இந்தச் செயல்பாடுகள் இரண்டு மதிப்புகளை ஒப்பிட்டு, அவை உண்மை அல்லது பொய்யா என்று சொல்லும். இது ஒரு கேள்வி கேட்டு ‘ஆம்’ அல்லது ‘இல்லை’ என்று பதில் பெறுவது போல.

செயல்பாடு	விளக்கம்	எடுத்துக்காட்டு	வெளியீடு
`==`	சமமா? (இரண்டு மதிப்புகள் சமமா)	`5 == 5`	`True`
		`'hello' == 'Hello'`	`False`
`!=`	சமம் இல்லையா? (இரண்டு மதிப்புகள் சமம் இல்லையா)	`5 != 10`	`True`
		`'apple' != 'apple'`	`False`
`>`	பெரியதா? (முதல் மதிப்பு இரண்டாவது மதிப்பை விடப் பெரியதா)	`10 > 5`	`True`
`<`	சிறியதா? (முதல் மதிப்பு இரண்டாவது மதிப்பை விடச் சிறியதா)	`5 < 10`	`True`
`>=`	பெரியதா அல்லது சமமா?	`10 >= 10`	`True`
`<=`	சிறியதா அல்லது சமமா?	`5 <= 5`	`True`

Python

```
x = 10
y = 20
name1 = "Alice"
name2 = "alice"

print(f"x == 10: {x == 10}")         # Output: True
print(f"y < x: {y < x}")             # Output: False
print(f"name1 == name2: {name1 == name2}") # Output: False (கேஸ் சென்சிட்டிவ்)
print(f"x != y: {x != y}")           # Output: Output: True
```

7.3 தர்க்கரீதியான செயல்பாடுகள் (Logical Operators): முடிவுகளை இணைத்தல்

பல பூலியன் மதிப்புகளையும், நிபந்தனைகளையும் இணைக்க நாம் **தர்க்கரீதியான செயல்பாடுகளை (Logical Operators)** பயன்படுத்துவோம். இது பல கேள்விகளை இணைத்து ஒரு பெரிய முடிவை எடுப்பது போல.

செயல்பாடு	விளக்கம்	எடுத்துக்காட்டு	வெளியீடு
`and`	இரண்டும் `True` ஆக இருந்தால் மட்டுமே `True`	`True and False`	`False`
		`(5 > 3) and (10 < 20)`	`True`
`or`	ஏதேனும் ஒன்று `True` ஆக இருந்தால் `True`	`True or False`	`True`
		`(5 == 5) or (10 > 20)`	`True`
`not`	பூலியன் மதிப்பின் எதிர்ப்பதத்தை வழங்கும் (True என்றால் False, False என்றால் True)	`not True`	`False`
		`not (5 < 3)`	`True`

Python

```python
has_license = True
has_car = False
is_sunny = True

# ஒருவருக்கு கார் ஓட்ட லைசென்ஸ் இருந்து கார் இருந்தால் மட்டுமே ஓட்ட முடியும்
can_drive = has_license and has_car
print(f"ஓட்ட முடியுமா? {can_drive}") # Output: False (கார் இல்லை)

# வெயில் அடித்தால் அல்லது லைசென்ஸ் இருந்தால் வெளியே செல்லலாம்
go_outside = is_sunny or has_license
print(f"வெளியே செல்லலாமா? {go_outside}") # Output: True (வெயில் அடிக்கிறது)

# மழை இல்லை
not_raining = not is_raining # is_raining என்பது True என்றால், not is_raining False ஆக இருக்கும்
print(f"மழை இல்லை: {not not_raining}") # Output: True (மழை பெய்கிறது)
```

கவனிக்க: `and` மற்றும் `or` செயல்பாடுகளில் நிபந்தனைகள் அடைப்புக்குறிக்குள் இருந்தால் குழப்பம் இல்லாமல் இருக்கும்.

7.4 பூலியன் பின்னணியில் (Booleans in Context): நிஜ உலகப் பயன்பாடு

பூலியன் மதிப்புகள் பெரும்பாலும் நிபந்தனைச் சாய்வுகள் (`if-elif-else`) மற்றும் லூப்களில் (`while`) முடிவுகளை எடுக்கப் பயன்படுத்தப்படுகின்றன. நாம் 'Number Guessing Game'-இல் இதை மறைமுகமாகப் பயன்படுத்தினோம் என்பதை நினைவில் கொள்ளுங்கள்!

Python

```python
# பயனர் வயது 18 அல்லது அதற்கு மேல் இருந்தால், 'வாக்களிக்கலாம்' என்று அச்சிடுக
age = 20
if age >= 18:
    print("வாக்களிக்கத் தகுதியுடையவர்.")
```

```
else:
    print("வாக்களிக்கத் தகுதியற்றவர்.")

# ஒரு பட்டியல் காலியாக இருக்கிறதா என்று சரிபார்க்க
my_list = []
if not my_list: # if my_list என்பது False ஆக இருந்தால்
    print("பட்டியல் காலியாக உள்ளது.")

# ஒரு அகராதியில் ஒரு விசை உள்ளதா என்று சரிபார்க்க
user_data = {"name": "Arjun", "email": "arjun@example.com"}
if "email" in user_data:
    print("மின்னஞ்சல் தகவல் உள்ளது.")
```

7.5 `True` மற்றும் `False` இன் மறைமுக அர்த்தங்கள் (Truthy and Falsy Values)

பைதானில், `True` மற்றும் `False` என்ற இரண்டு பூலியன் மதிப்புகள் மட்டுமல்லாமல், மற்ற தரவு வகைகளுக்கும் ஒரு ‘பூலியன் அர்த்தம்’ உண்டு. இதை **Truthy** (உண்மையைப் போன்றது) மற்றும் **Falsy** (பொய்க்குச் சமமானது) மதிப்புகள் என்று அழைக்கிறார்கள். இது ஒரு நிபந்தனையைச் சரிபார்க்கும்போது மிகவும் பயனுள்ளதாக இருக்கும்.

பொதுவாக, பின்வரும் மதிப்புகள் **`Falsy`** என கருதப்படுகின்றன:

- `False` (சுய பூலியன் பொய்)
- `None` (எதுவும் இல்லை)
- `0` (எண் பூஜ்ஜியம்)
- `0.0` (தசம பூஜ்ஜியம்)
- `""` (வெற்று ஸ்ட்ரிங்)
- `[]` (வெற்று பட்டியல்)
- `()` (வெற்று டியூபிள்)
- `{}` (வெற்று அகராதி)
- `set()` (வெற்றுத் தொகுப்பு)

இந்த `Falsy` மதிப்புகள் தவிர மற்ற அனைத்து மதிப்புகளும் (எ.கா: எந்த எண்ணும் 0 தவிர, எந்த எழுத்தும் வெற்று ஸ்ட்ரிங் தவிர, எந்த பட்டியலும் வெற்றுப் பட்டியல் தவிர) **`Truthy`** எனக் கருதப்படுகின்றன.

Python

```
# Falsy மதிப்புகளுக்கான எடுத்துக்காட்டுகள்:
if 0:
    print("இது அச்சிடப்படாது")
if "":
    print("இதுவும் அச்சிடப்படாது")
if []:
    print("இதுவும் அச்சிடப்படாது")
```

```
# Truthy மதிப்புகளுக்கான எடுத்துக்காட்டுகள்:
if 1:
    print("1 என்பது Truthy") # Output: 1 என்பது Truthy
if "Hello":
    print("வெற்று இல்லாத ஸ்ட்ரிங் Truthy") # Output: வெற்று இல்லாத ஸ்ட்ரிங் Truthy
if [1, 2]:
    print("வெற்று இல்லாத பட்டியல் Truthy") # Output: வெற்று இல்லாத பட்டியல் Truthy
```

இந்த `Truthy` மற்றும் `Falsy` கருத்து, உங்கள் குறியீட்டைச் சுருக்கமாகவும், படிக்க எளிதாகவும் மாற்ற உதவும்.

பூலியன் தரவு வகை, ஒப்பீட்டுச் செயல்பாடுகள், மற்றும் தர்க்கரீதியான செயல்பாடுகள் ஆகியவை பைதான் நிரலாக்கத்தின் இதயமாகும். கணினி முடிவெடுக்கும் ஒவ்வொரு படிக்கும் இதுவே அடிப்படையாக அமைகிறது.

பூலியன் பற்றி இந்த விரிவான புரிதல், உங்கள் பைதான் திறமைகளை மேலும் ஒரு படி உயர்த்தும். அடுத்த பகுதியில், நாம் இதுவரை கற்றுக்கொண்ட அனைத்து அடிப்படைத் தரவு வகைகளையும் (எண்கள், ஸ்ட்ரிங்ஸ், லிஸ்ட், டியூபிள், செட், டிக்ஷனரி, பூலியன்) ஒருங்கே வைத்துப் பார்க்கும் ஒரு சுருக்கத்தையும், உங்கள் அடுத்த பயணத்திற்கான வழிகாட்டுதலையும் காண்போம்.

8. பைதானில் முடிவெடுக்கும் கலை – கட்டுப்பாட்டு ஓட்டம் (Control Flow)

நாம் இதுவரை பைதானின் அடிப்படைக் கட்டுமானத் தொகுதிகளான தரவு வகைகளைப் பற்றி விரிவாகக் கற்றுக் கொண்டோம். எண்கள், எழுத்துகள், பட்டியல்கள், அகராதிகள் – இவை அனைத்தும் தகவல்களைச் சேமிக்கவும், நிர்வகிக்கவும் உதவுகின்றன. ஆனால், வெறும் தகவல்களைச் சேமிப்பதால் மட்டும் ஒரு நிரல் (program) உயிருள்ளதாக மாறாது. ஒரு நிரல் உண்மையிலேயே புத்திசாலித்தனமாகச் செயல்பட வேண்டுமானால், அது **முடிவுகளை எடுக்க வேண்டும்**; சில நிபந்தனைகளின் அடிப்படையில் வெவ்வேறு பாதைகளில் செல்ல வேண்டும்; அல்லது ஒரு குறிப்பிட்ட வேலையை மீண்டும் மீண்டும் செய்ய வேண்டும்.

நம் அன்றாட வாழ்க்கையில்கூட, நாம் தொடர்ந்து முடிவுகளை எடுத்துக்கொண்டே இருக்கிறோம்:

- “காலை 8 மணிக்குள் மழை பெய்தால், குடையை எடுத்துக்கொண்டு செல்ல வேண்டும். இல்லையென்றால், குடை தேவையில்லை.”
- “வங்கிக் கணக்கில் போதுமான பணம் இருந்தால், ஆன்லைனில் பொருட்களை வாங்கலாம். இல்லை என்றால், வேறு வழியைக் கண்டுபிடிக்க வேண்டும்.”
- “ஒரு பட்டியலிலுள்ள ஒவ்வொரு பெயரையும் படித்து, ஒவ்வொருவருக்கும் தனிப்பட்ட வாழ்த்துச் செய்தி அனுப்ப வேண்டும்.”

இந்த ’முடிவெடுக்கும் கலை’யும், ’மீண்டும் மீண்டும் செய்யும் செயல்பாடு’ம்தான் நிரலாக்கத்தின் இதயம். பைதான் மொழியில், இந்த முடிவெடுக்கும் மற்றும் செயல்முறைப்படுத்தும் திறனைப் பெற நாம் **கட்டுப்பாட்டு ஓட்டம் (Control Flow)** என்ற கருத்தைப் பயன்படுத்துகிறோம். இது, உங்கள் நிரல் எந்த வரிசையில் இயங்க வேண்டும், எந்தச் சூழ்நிலையில் என்ன செய்ய வேண்டும் என்பதை வழிநடத்தும் ஒரு ‘வரைபடம்’ (roadmap) போன்றது.

இந்த அத்தியாயத்தில், பைதான் நிரல்கள் எப்படி முடிவெடுக்கின்றன என்பதைப் பற்றி விரிவாகக் கற்கப் போகிறோம்.

8.1 நிபந்தனைச் சாய்வுகள் (`if`, `elif`, `else`): முடிவு எடுக்கும் வழிமுறை

ஒரு குறிப்பிட்ட நிபந்தனை (condition) உண்மையா அல்லது பொய்யா என்பதைச் சரிபார்த்து, அதற்கேற்ப ஒரு குறிப்பிட்ட குறியீட்டுப் பகுதியை இயக்க (`execute`) நாம் **நிபந்தனைச் சாய்வுகளை (Conditional Statements)** பயன்படுத்துகிறோம். பைதானில், இதற்கு `if`, `elif` (else if-ன் சுருக்கம்), மற்றும் `else` என்ற சிறப்புச் சொற்கள் பயன்படுகின்றன.

8.1.1 `if` அறிக்கை: ஒரு எளிய நிபந்தனை

மிகவும் அடிப்படையான முடிவு எடுக்கும் வடிவம் இது. ஒரு நிபந்தனை `True` ஆக இருந்தால் மட்டுமே, ஒரு குறிப்பிட்ட குறியீடு இயங்கும்.

கட்டமைப்பு:

Python

```
if நிபந்தனை:
    # நிபந்தனை True ஆக இருந்தால் இயக்கப்படும் குறியீடு
    # (இங்குள்ள indent - இடைவெளி - மிக முக்கியம்!)
```

உதாரணம்:

Python

```
temperature = 28 # வெப்பநிலை

if temperature > 25:
    print("வெப்பநிலை அதிகம், குளிர்பானம் குடிக்கவும்.")
```

இங்கு `temperature > 25` என்ற நிபந்தனை `True` ஆக இருந்தால் மட்டுமே, “வெப்பநிலை அதிகம், குளிர்பானம் குடிக்கவும்.” என்ற செய்தி அச்சிடப்படும்.

8.1.2 `if-else` அறிக்கை: இரு வாய்ப்புகள், ஒரு முடிவு

ஒரு நிபந்தனை `True` ஆக இருந்தால் ஒரு வேலையும், `False` ஆக இருந்தால் வேறு ஒரு வேலையும் செய்ய வேண்டியிருக்கும்போது `if-else` அறிக்கை பயன்படும்.

கட்டமைப்பு:

Python

```
if நிபந்தனை:
    # நிபந்தனை True ஆக இருந்தால் இயக்கப்படும் குறியீடு
else:
    # நிபந்தனை False ஆக இருந்தால் இயக்கப்படும் குறியீடு
```

உதாரணம்:

Python

```
age = 17

if age >= 18:
    print("வாக்களிக்கத் தகுதியுடையவர்.")
else:
    print("வாக்களிக்கத் தகுதியற்றவர்.")
```

இங்கு `age >= 18` என்ற நிபந்தனை `True` ஆக இருந்தால் முதல் பகுதி இயங்கும். `False` ஆக இருந்தால், `else` பகுதி இயங்கும்.

8.1.3 `if-elif-else` அறிக்கை: பல வாய்ப்புகள், ஒரு முடிவு

இரண்டுக்கும் மேற்பட்ட நிபந்தனைகளைச் சரிபார்க்க வேண்டியிருக்கும்போது `if-elif-else` அறிக்கை பயன்படுத்தப்படுகிறது. இது பல சாத்தியக்கூறுகளில் இருந்து ஒரு முடிவைத் தேர்ந்தெடுக்க உதவுகிறது.

கட்டமைப்பு:

Python

```
if முதல்_நிபந்தனை:
    # முதல்_நிபந்தனை True ஆக இருந்தால்
elif இரண்டாம்_நிபந்தனை:
    # முதல்_நிபந்தனை False ஆகவும், இரண்டாம்_நிபந்தனை True ஆகவும் இருந்தால்
elif மூன்றாம்_நிபந்தனை:
    # முதல் மற்றும் இரண்டாம் நிபந்தனைகள் False ஆகவும், மூன்றாம்_நிபந்தனை True ஆகவும் இருந்தால்
else:
    # எந்த நிபந்தனையும் True ஆக இல்லாவிட்டால்
```

உதாரணம்:

Python

```
score = 85

if score >= 90:
    print("Grade: A")
elif score >= 80: # 90-ஐ விடக் குறைவு, ஆனால் 80 அல்லது அதற்கு மேல்
    print("Grade: B")
elif score >= 70: # 80-ஐ விடக் குறைவு, ஆனால் 70 அல்லது அதற்கு மேல்
    print("Grade: C")
else: # 70-ஐ விடக் குறைவு
    print("Grade: D")
```

இங்கு, நிரல் ஒவ்வொரு நிபந்தனையையும் வரிசையாகச் சரிபார்க்கும். எந்த நிபந்தனை முதலில் `True` ஆகிறதோ, அதனுடன் தொடர்புடைய குறியீடு இயங்கும், மற்ற பகுதிகள் தவிர்க்கப்படும். எந்த நிபந்தனையும் `True` ஆகாவிட்டால், `else` பகுதி இயங்கும்.

முக்கியக் குறிப்பு: இன்டென்டேஷன் (Indentation) - பைதானின் கட்டாய இடைவெளி!

`if`, `elif`, `else` அறிக்கைகளில், நிபந்தனைக்குப் பிறகு வரும் குறியீடு ஒரு குறிப்பிட்ட **இடைவெளியுடன் (indentation)** தொடங்க வேண்டும். பைதான் இந்த இடைவெளியை, ஒரு குறியீட்டுத் தொகுதி (block of code) எந்த நிபந்தனைக்குச் சொந்தமானது என்பதைக் குறிக்கப் பயன்படுத்துகிறது. வழக்கமாக, நான்கு இடைவெளிகள் (spaces) அல்லது ஒரு `Tab` பயன்படுத்தப்படுகிறது.

Python

```
# சரியான இன்டென்டேஷன்
if True:
    print("இந்த வரி if க்கு சொந்தமானது")
    print("இந்த வரிகளும் if க்கு சொந்தமானது")
print("இந்த வரி if க்கு வெளியே உள்ளது") # இதற்கு இன்டென்ட் இல்லை

# தவறான இன்டென்டேஷன் (பிழை ஏற்படுத்தும்)
# if True:
# print("இந்த வரி தவறானது")
```

சரியான இன்டென்டேஷன் இல்லாமல் பைதான் நிரல்களை இயக்கினால் **`IndentationError`** என்ற பிழை ஏற்படும். இது பைதானின் மிக முக்கியமான இலக்கண விதிகளில் ஒன்று.

8.2 பல நிபந்தனைகள்: `and`, `or`, `not` பயன்பாடு

நாம் ‘பூலியன்’ அத்தியாயத்தில் பார்த்த தர்க்கரீதியான செயல்பாடுகளான (`and`, `or`, `not`) இங்கு, பல நிபந்தனைகளை இணைத்து மிகவும் சிக்கலான முடிவுகளை எடுக்கப் பயன்படுகின்றன.

- **`and` (மற்றும்):** இரண்டு நிபந்தனைகளும் `True` ஆக இருந்தால் மட்டுமே மொத்த நிபந்தனையும் `True` ஆகும்.
- **`or` (அல்லது):** இரண்டு நிபந்தனைகளில் ஏதேனும் ஒன்று `True` ஆக இருந்தால், மொத்த நிபந்தனையும் `True` ஆகும்.
- **`not` (இல்லை):** ஒரு நிபந்தனையின் எதிர்மறையை (opposites) வழங்கும். (`True` என்றால் `False`, `False` என்றால் `True`).

உதாரணம்:

Python

```
has_passport = True
has_visa = True
has_ticket = False
age = 25

# வெளிநாடு செல்ல பாஸ்போர்ட் மற்றும் விசா இரண்டும் தேவை
if has_passport and has_visa:
    print("நீங்கள் வெளிநாடு செல்லத் தயார்.")
else:
    print("பாஸ்போர்ட் அல்லது விசா இல்லை.")

# பயணிக்க டிக்கெட் இருந்தால் அல்லது வயது 18-க்கு மேல் இருந்தால்
if has_ticket or age >= 18:
    print("பயணிக்கலாம்.") # டிக்கெட் இல்லாவிட்டாலும், வயது 18க்கு மேல் என்பதால் 'True'
else:
    print("பயணிக்க முடியாது.")

# மழை பெய்யவில்லை என்றால் (is_raining = False)
is_raining = False
if not is_raining:
    print("குடை தேவையில்லை!") # Output: குடை தேவையில்லை!
```

இந்த `if`, `elif`, `else` அறிக்கைகள், `and`, `or`, `not` போன்ற தர்க்கரீதியான செயல்பாடுகளுடன் இணைந்து, உங்கள் நிரல்களை நிஜ உலகச் சூழ்நிலைகளுக்கு ஏற்ப புத்திசாலித்தனமாகச் செயல்பட வைக்க உதவுகின்றன. ஒரு நிரல் புத்திசாலித்தனமாகச் செயல்பட, இந்த நிபந்தனைச் சாய்வுகள் தான் அதன் அடிப்படை ‘மூளை’!

அடுத்த பகுதியில், ஒரு குறிப்பிட்ட செயலை மீண்டும் மீண்டும் எப்படிச் செய்வது என்பதைப் பார்ப்போம். அதுதான் **‘லூப்கள்’ (Loops)**!

8.3 லூப்கள் (Loops): மீண்டும் மீண்டும் செய்யும் வேலைகளை எளிதாக்குதல்

நம் அன்றாட வாழ்க்கையில் பல வேலைகளை நாம் மீண்டும் மீண்டும் செய்ய வேண்டியிருக்கும். உதாரணமாக, ஒவ்வொரு நாளும் பல் துலக்குவது, ஒவ்வொரு வாடிக்கையாளருக்கும் மின்னஞ்சல் அனுப்புவது, ஒரு பட்டியலில் உள்ள ஒவ்வொரு பொருளையும் சரிபார்ப்பது. ஒரு கணினி நிரலிலும் இதுதான்! ஒரே வேலையை நூற்றுக்கணக்கான, ஆயிரக்கணக்கான, ஏன் மில்லியன் கணக்கான முறை செய்ய வேண்டியிருக்கும்.

ஒவ்வொரு முறையும் குறியீட்டை மீண்டும் மீண்டும் எழுதுவது மிகவும் சலிப்பை ஏற்படுத்தும், அது நடைமுறைக்கும் சாத்தியமில்லை. இந்த ‘மீண்டும் மீண்டும் செய்யும் வேலைகளை’ (repetitive tasks) எளிதாக்கவே பைதான் **‘லூப்கள்’ (Loops)** என்ற கருத்தை வழங்குகிறது. லூப்கள் உங்கள் நிரல்கள் புத்திசாலித்தனமாக, தானாகவே வேலைகளைத் திரும்பத் திரும்பச் செய்ய உதவுகின்றன.

பைதான் இரண்டு முக்கிய லூப்களைக் கொண்டுள்ளது:

1. **`while` லூப்:** ஒரு நிபந்தனை `True` ஆக இருக்கும் வரை இயங்கும்.
2. **`for` லூப்:** ஒரு வரிசையில் (sequence - list, string, tuple போன்றவை) உள்ள ஒவ்வொரு உறுப்பிற்காகவும் ஒருமுறை இயங்கும்.

8.3.1 `while` லூப்: நிபந்தனை இருக்கும் வரை தொடரலாம்!

`while` லூப் ஒரு கதவு போல. ஒரு குறிப்பிட்ட நிபந்தனை `True` ஆக இருக்கும் வரை அந்தக் கதவு திறந்திருக்கும், அதன் உள்ளே உள்ள குறியீடுகள் இயங்கிக் கொண்டே இருக்கும். நிபந்தனை `False` ஆனவுடன், கதவு மூடப்பட்டு, லூப் நின்றுவிடும்.

கட்டமைப்பு:

Python

```
while நிபந்தனை:
    # நிபந்தனை True ஆக இருக்கும் வரை இயக்கப்படும் குறியீடு
    # (இங்குள்ள indent - இடைவெளி - மிக முக்கியம்!)
    # லூப்பை நிறுத்த நிபந்தனையை மாற்றும் ஒரு வழிமுறையும் இங்கு இருக்க வேண்டும்!
```

உதாரணம் 1: ஒரு எளிய கவுண்டர்

Python

```
count = 0

while count < 5: # 'count' 5-ஐ விடக் குறைவாக இருக்கும் வரை
    print(f"எண்ணிக்கை: {count}")
    count += 1 # ஒவ்வொரு முறையும் count-ஐ 1 அதிகரிக்கிறோம் (count = count + 1)

print("லூப் முடிந்தது!")
```

விளக்கம்:

- `count` 0 ஆக ஆரம்பிக்கிறது.
- `count < 5` (0 < 5) `True` என்பதால், “எண்ணிக்கை: 0” அச்சிடப்படும். `count` 1 ஆக மாறும்.
- மீண்டும் `count < 5` (1 < 5) `True` என்பதால், “எண்ணிக்கை: 1” அச்சிடப்படும். `count` 2 ஆக மாறும்.
- …இப்படியே `count` 4 ஆக இருக்கும் வரை தொடரும்.
- `count` 5 ஆக மாறும்போது, `count < 5` (5 < 5) `False` ஆகிவிடும். லூப் நின்றுவிடும்.

உதாரணம் 2: ஒரு குறிப்பிட்ட உள்ளீட்டைப் பெறும் வரை காத்திருத்தல்

நாம் முன்பே பார்த்த ‘Number Guessing Game’-இல், பயனர் சரியான எண்ணை உள்ளிடும் வரை `while True` லூப்பை நாம் பயன்படுத்தினோம்.

Python

```
password = ""
while password != "secret":
    password = input("கடவுச்சொல்லை உள்ளிடவும்: ")
    if password != "secret":
        print("தவறான கடவுச்சொல்! மீண்டும் முயற்சிக்கவும்.")
print("கடவுச்சொல் சரி! உள்ளே வாருங்கள்.")
```

விளக்கம்:

- `password` என்பது முதலில் காலியாக உள்ளது.
- `password != "secret"` `True` என்பதால், கடவுச்சொல் கேட்கப்படும்.
- சரியான கடவுச்சொல் உள்ளிடப்படும் வரை லூப் தொடரும்.

எச்சரிக்கை: முடிவற்ற லூப்கள் (Infinite Loops)!

`while` லூப்களில் ஒரு முக்கியமான ஆபத்து உள்ளது: நிபந்தனை எப்போதும் `True` ஆகவே இருந்தால், லூப் ஒருபோதும் நிற்காது! உங்கள் நிரல் முடிவில்லாமல் இயங்கிக் கொண்டே இருக்கும், கணினியை முடக்கிவிடும்.

Python

```
# இது ஒரு முடிவற்ற லூப்! இதை இயக்க வேண்டாம்!
while True:
      print("நான் முடிவில்லாமல் இயங்குகிறேன்!")
```

தீர்வு: `while` லூப் பயன்படுத்தும்போது, நிபந்தனை ஒரு கட்டத்தில் `False` ஆக மாறுகிறது என்பதை உறுதிப்படுத்திக் கொள்ள வேண்டும். (உதாரணமாக, `count += 1` போல ஒரு மதிப்பை மாற்றுவதன் மூலம்).

8.3.2 `for` லூப்: ஒவ்வொரு உறுப்பையும் சுற்றி வருவோம்!

`for` லூப், ஒரு ‘வரிசையில்’ (sequence) உள்ள ஒவ்வொரு உறுப்பிற்காகவும் ஒருமுறை இயங்க வடிவமைக்கப்பட்டது. இந்த ‘வரிசை’ என்பது ஒரு `list`, `string`, `tuple`, `set` அல்லது `dictionary` ஆக இருக்கலாம். இதை ஒரு குழுவில் உள்ள ஒவ்வொரு உறுப்பினரையும் தனித்தனியாக அழைத்து பேசுவது போல கற்பனை செய்து கொள்ளலாம்.

கட்டமைப்பு:

Python

```
for உறுப்பு_மாறி in வரிசை:
    # வரிசையில் உள்ள ஒவ்வொரு உறுப்பிற்காகவும் இயக்கப்படும் குறியீடு
```

உதாரணம் 1: பட்டியலின் உறுப்புகளை அச்சிடுதல்

Python

```
fruits = ["apple", "banana", "cherry"]

for fruit in fruits:
    print(f"எனக்கு பிடித்த பழம்: {fruit}")

# Output:
# எனக்கு பிடித்த பழம்: apple
# எனக்கு பிடித்த பழம்: banana
# எனக்கு பிடித்த பழம்: cherry
```

உதாரணம் 2: ஸ்ட்ரிங்கில் உள்ள ஒவ்வொரு எழுத்தையும் அச்சிடுதல்

Python

```
word = "Python"

for letter in word:
    print(f"எழுத்து: {letter}")

# Output:
# எழுத்து: P
# எழுத்து: y
# எழுத்து: t
# ...
```

உதாரணம் 3: அகராதியின் விசைகள் மற்றும் மதிப்புகளை அச்சிடுதல்

Python

```
student_grades = {"Math": 90, "Science": 85, "History": 78}

for subject, grade in student_grades.items():
    print(f"{subject} மதிப்பெண்: {grade}")

# Output:
# Math மதிப்பெண்: 90
# Science மதிப்பெண்: 85
# History மதிப்பெண்: 78
```

`range()` செயல்பாட்டுடன் `for` லூப்: எண்களை எண்ணுவோம்!

`for` லூப்பை எண்களின் வரிசையில் இயக்க, நாம் `range()` என்ற சிறப்புச் செயல்பாட்டைப் பயன்படுத்தலாம். இது ஒரு குறிப்பிட்ட எண் வரம்பை உருவாக்கும்.

- `range(stop)`: 0 இல் தொடங்கி, `stop` எண் வரை (ஆனால் `stop` சேர்க்கப்படாது) எண்களை உருவாக்கும்.
- `range(start, stop)`: `start` இல் தொடங்கி, `stop` எண் வரை எண்களை உருவாக்கும்.
- `range(start, stop, step)`: `start` இல் தொடங்கி, `stop` எண் வரை, `step` அளவு தாண்டி எண்களை உருவாக்கும்.

உதாரணம்:

Python

```python
# 0 முதல் 4 வரை (5 சேர்க்கப்படாது)
for i in range(5):
    print(f"எண் (range 5): {i}")

# Output: 0, 1, 2, 3, 4

# 5 முதல் 10 வரை (10 சேர்க்கப்படாது)
for i in range(5, 10):
    print(f"எண் (range 5, 10): {i}")

# Output: 5, 6, 7, 8, 9

# 0 முதல் 10 வரை, 2 ஆகத் தாண்டி
for i in range(0, 10, 2):
    print(f"எண் (range 0, 10, 2): {i}")

# Output: 0, 2, 4, 6, 8
```

`range()` மூலம், ஒரு குறிப்பிட்ட எண்ணிக்கையிலான முறை லூப்பை இயக்கலாம், அல்லது ஒரு எண் வரிசையை உருவாக்கலாம்.

8.3.3 லூப் கட்டுப்பாட்டு அறிக்கைகள் (Loop Control Statements): பயணத்தின் திசையை மாற்றுதல்

சில சமயங்களில், ஒரு லூப் இயங்கிக் கொண்டிருக்கும்போது, அதன் இயல்பான ஓட்டத்தை மாற்ற வேண்டியிருக்கும். இதைச் செய்ய பைதான் இரண்டு முக்கியக் கட்டுப்பாட்டு அறிக்கைகளை வழங்குகிறது:

`break` அறிக்கை: லூப்பை விட்டு வெளியேறுதல்

`break` அறிக்கை, ஒரு லூப்பை (அது `for` லூப்பாக இருந்தாலும் சரி, `while` லூப்பாக இருந்தாலும் சரி) உடனடியாக நிறுத்தி, லூப்க்கு வெளியே உள்ள குறியீட்டுக்குச் செல்லச் சொல்லும். இது ஒரு அவசர வெளியேறும் கதவு போல.

உதாரணம்: ஒரு குறிப்பிட்ட எண்ணைக் கண்டறிந்தவுடன் லூப்பை நிறுத்துதல்.

Python

```
numbers = [10, 25, 5, 30, 15]
target = 5

for num in numbers:
    if num == target:
        print(f"இலக்கு {target} கண்டுபிடிக்கப்பட்டது!")
        break # எண்ணைக் கண்டுபிடித்துவிட்டதால் லூப்பை நிறுத்துகிறோம்
    print(f"தற்போதைய எண்: {num}")

print("தேடுதல் முடிந்தது.")
```

விளக்கம்: `num` 5 ஆக இருக்கும்போது, `if` நிபந்தனை `True` ஆகி, `break` இயங்கும். லூப் உடனடியாக நின்றுவிடும், “தேடுதல் முடிந்தது.” என்று அச்சிடப்படும். 15 என்ற எண் அச்சிடப்படாது.

`continue` அறிக்கை: நடப்புச் சுழற்சியைத் தவிர்த்தல்

`continue` அறிக்கை, லூப்பின் நடப்புச் சுழற்சியை (current iteration) நிறுத்தி, அடுத்த சுழற்சிக்குச் செல்லச் சொல்லும். இது ஒரு குறிப்பிட்ட நிலையை மட்டும் தவிர் என்று சொல்வது போல.

உதாரணம்: ஒற்றைப்படை எண்களை மட்டும் அச்சிடுதல் (இரட்டைப்படை எண்களைத் தவிர்த்தல்).

Python

```
for i in range(1, 11): # 1 முதல் 10 வரை
    if i % 2 == 0: # எண் இரட்டைப்படையாக இருந்தால் (மீதம் 0 ஆக இருந்தால்)
        continue   # இந்தச் சுழற்சியைத் தவிர்த்து, அடுத்த சுழற்சிக்குச் செல்
    print(f"ஒற்றைப்படை எண்: {i}")

# Output:
# ஒற்றைப்படை எண்: 1
# ஒற்றைப்படை எண்: 3
# ...
# ஒற்றைப்படை எண்: 9
```

விளக்கம்: `i` ஒரு இரட்டைப்படை எண்ணாக (2, 4, 6, 8, 10) இருக்கும்போது, `continue` இயங்கும். `print()` அறிக்கை தவிர்க்கப்பட்டு, லூப் அடுத்த எண்ணுக்குச் செல்லும்.

8.3.4 Nested Loops (அடுக்கு லூப்கள்): சிக்கலான வடிவங்களை உருவாக்குதல்

ஒரு லூப்பிற்குள் மற்றொரு லூப் இருந்தால், அதை **‘Nested Loop’ (அடுக்கு லூப்)** என்று அழைக்கிறோம். இது மிகவும் சிக்கலான வடிவங்கள் மற்றும் தரவுகளைக் கையாளப் பயன்படும். உதாரணமாக, ஒரு அட்டவணையின் (table) வரிசைகள் மற்றும் நெடுவரிசைகளில் (rows and columns) வேலை செய்வது போல.

உதாரணம்: நட்சத்திர வடிவங்களை அச்சிடுதல்

Python

```
# 3x3 நட்சத்திர கட்டம் அச்சிடுதல்
for row in range(3): # வெளிப்புற லூப் – ஒவ்வொரு வரிசைக்கும்
    for col in range(3): # உள் லூப் – ஒவ்வொரு நெடுவரிசைக்கும்
        print("*", end=" ") # நட்சத்திரத்தை அச்சிட்டு, அதே வரியில் இடைவெளி சேர்க்கவும்
    print() # உள் லூப் முடிந்ததும் ஒரு புதிய வரிக்குச் செல்லவும்

# Output:
# * * *
# * * *
# * * *
```

விளக்கம்: வெளிப்புற லூப் மூன்று முறை இயங்கும் (row 0, 1, 2). ஒவ்வொரு முறையும் வெளிப்புற லூப் இயங்கும்போது, உள் லூப் மூன்று முறை இயங்கும் (col 0, 1, 2), '*' ஐ அச்சிடும். உள் லூப் முடிந்ததும், `print()` ஒரு புதிய வரிக்குச் செல்லும்.

லூப்கள் (Loops) மற்றும் கட்டுப்பாட்டு ஓட்டம் (Control Flow) ஆகியவை பைதான் நிரலாக்கத்தின் இதயம். இவை, உங்கள் நிரல்களை வெறும் தகவல்களைச் சேமிக்கும் கருவிகளாக இல்லாமல், **புத்திசாலித்தனமான, முடிவெடுக்கும், மற்றும் வேலைகளைத் தானாகச் செய்யும் சக்திவாய்ந்த இயந்திரங்களாக** மாற்றுகின்றன.

நாம் இதுவரை, ஒரு நிரலை எப்படி முடிவெடுக்க வைப்பது (`if-elif-else`) என்பதையும், ஒரு குறிப்பிட்ட செயலை எப்படி மீண்டும் மீண்டும் செய்வது (`for` மற்றும் `while` லூப்கள்) என்பதையும் கண்டோம். ஆனால், ஒரே மாதிரியான குறியீட்டுப் பகுதிகளை உங்கள் நிரலின் பல இடங்களில் மீண்டும் மீண்டும் எழுத வேண்டிய சூழ்நிலைகள் வந்தால் என்ன செய்வது? உதாரணமாக, ஒரு பயனரை வாழ்த்தும் குறியீடு, அல்லது ஒரு கணக்கீட்டைச் செய்யும் குறியீடு, உங்கள் நிரலில் பல இடங்களில் தேவைப்படலாம்.

ஒரே சமையல் குறிப்பை ஒவ்வொரு முறையும் முழுமையாக எழுதுவதற்குப் பதிலாக, அதன் பெயரை மட்டும் சொல்லி, "சம்பார் செய்!" என்று சொல்வது எவ்வளவு எளிதோ, அதுபோலத்தான் நிரலாக்கத்திலும். உங்கள் குறியீட்டை ஒழுங்கமைத்து, மீண்டும் மீண்டும் பயன்படுத்தக் கூடிய சிறிய, தனிப்பட்ட பகுதிகளாகப் பிரிக்க பைதான் ஒரு சக்திவாய்ந்த கருவியை வழங்குகிறது – அதுதான் **செயல்பாடுகள் (Functions)**.

ஒரு செயல்பாடு என்பது உங்கள் புரோகிராமிங் கருவிப் பெட்டியில் உள்ள ஒரு சிறப்பு கருவி போல. உங்களுக்கு ஒரு குறிப்பிட்ட வேலை செய்ய வேண்டுமானால், அந்தக் கருவியை எடுத்துப் பயன்படுத்தினால் போதும், அதன் உள்வேலையைப் பற்றி நீங்கள் கவலைப்பட வேண்டியதில்லை. இந்த அத்தியாயத்தில், செயல்பாடுகள் என்றால் என்ன, அவற்றை எப்படி உருவாக்குவது, எப்படிப் பயன்படுத்துவது என்று விரிவாகக் கற்கப் போகிறோம்.

8.4 செயல்பாடுகள் (Functions): குறியீட்டை ஒழுங்கமைக்கும் மந்திரக் கருவிகள்

செயல்பாடுகள், உங்கள் குறியீட்டை மிகவும் ஒழுங்கமைக்கப்பட்டதாகவும், மறுபயன்பாட்டுக்கு (reusable) ஏற்றதாகவும், படிக்க எளிதாகவும் மாற்ற உதவுகின்றன. இது ஒரு பெரிய புத்தகத்தை அத்தியாயங்களாகப் பிரிப்பது போல, அல்லது ஒரு பெரிய தொழிற்சாலையைச் சிறிய அலகுகளாகப் பிரிப்பது போல.

8.4.1 ஒரு செயல்பாடு என்றால் என்ன? (What is a Function?)

எளிமையாகச் சொன்னால், ஒரு **செயல்பாடு (Function)** என்பது ஒரு குறிப்பிட்ட வேலையைச் செய்ய வடிவமைக்கப்பட்ட, ஒழுங்கமைக்கப்பட்ட, மறுபயன்பாட்டுக்கு ஏற்ற ஒரு குறியீட்டுத் தொகுதி. நீங்கள் அதை ஒருமுறை எழுதினால் போதும், உங்கள் நிரலின் எந்தப் பகுதியிலிருந்தும் அதை எத்தனை முறை வேண்டுமானாலும் 'அழைக்கலாம்' (call).

கட்டமைப்பு:

Python

```
def செயல்பாட்டின்_பெயர்(அளவுருக்கள்):
    # செயல்பாட்டின் குறியீடு இங்கு வரும்
    # (இங்குள்ள indent - இடைவெளி - மிக முக்கியம்!)
    # விருப்பப்பட்டால், ஒரு மதிப்பை 'return' செய்யலாம்
```

- `def`: ஒரு செயல்பாட்டை வரையறுக்கப் பயன்படும் சிறப்புச் சொல்.
- `செயல்பாட்டின்_பெயர்`: நீங்கள் செயல்பாட்டிற்குக் கொடுக்கும் தனிப்பட்ட பெயர் (எ.கா: `greet`, `add_numbers`).
- `அளவுருக்கள் (Parameters)`: செயல்பாட்டிற்கு வெளியே இருந்து தகவல்களை உள்ளே அனுப்பப் பயன்படும் மாறிகள். இவை அடைப்புக்குறிகளுக்குள் வரும், விருப்பமானவை (optional).
- `:` (கோலன்): செயல்பாட்டுத் தலைப்பிற்குப் பிறகு கட்டாயமாக வர வேண்டும்.
- `குறியீடு`: கோலனுக்குப் பிறகு உள்ளே தள்ளி (indent) எழுதப்படும் பகுதியே செயல்பாட்டின் உள்ளடக்கமாகும்.

எடுத்துக்காட்டு: ஒரு எளிய வாழ்த்துச் செயல்பாடு

Python

```
# greet() என்ற செயல்பாட்டை வரையறுக்கிறோம்
def greet():
    print("வணக்கம், பைதான் உலகிற்கு உங்களை வரவேற்கிறோம்!")

# செயல்பாட்டை அழைக்கிறோம் (execute செய்கிறோம்)
greet()
greet() # நீங்கள் இதை எத்தனை முறை வேண்டுமானாலும் அழைக்கலாம்
```

விளக்கம்:

greet() என்ற செயல்பாட்டை ஒருமுறை வரையறுத்துவிட்டு, அதை எத்தனை முறை வேண்டுமானாலும் அழைத்து, ஒரே மாதிரியான வாழ்த்துச் செய்தியைப் பெற முடியும். இது குறியீட்டை மீண்டும் மீண்டும் எழுதுவதைத் தவிர்க்கிறது.

8.4.2 செயல்பாட்டு அளவுருக்கள் (Function Parameters): தகவல்களை உள்ளே அனுப்புதல்

பல சமயங்களில், ஒரு செயல்பாடு, வெளியிலிருந்து சில தகவல்களைப் பெற்று, அதன் அடிப்படையில் வேலை செய்ய வேண்டியிருக்கும். இந்தத் தகவல்களைப் பெற நாம் **அளவுருக்களை (Parameters)** பயன்படுத்துகிறோம். இவை ஒரு சமையல் குறிப்பிற்குத் தேவையான ‘பொருட்கள்’ போல.

Python

```
def greet_user(name): # 'name' என்பது ஒரு அளவுரு
    print(f"வணக்கம், {name}! பைதான் உலகிற்கு உங்களை வரவேற்கிறோம்!")

# செயல்பாட்டை அளவுகளுடன் அழைக்கிறோம் (arguments)
greet_user("அறிவு")    # Output: வணக்கம், அறிவு! பைதான் உலகிற்கு உங்களை வரவேற்கிறோம்!
greet_user("கனிமொழி") # Output: வணக்கம், கனிமொழி! பைதான் உலகிற்கு உங்களை வரவேற்கிறோம்!
```

விளக்கம்:

இங்கு name என்பது greet_user செயல்பாட்டின் ஒரு அளவுரு. நாம் செயல்பாட்டை அழைக்கும்போது (greet_user("அறிவு")), "அறிவு" என்ற மதிப்பை name என்ற அளவுருவுக்கு அனுப்புகிறோம். இந்த மதிப்புகள் 'ஆர்கியூமென்ட்ஸ்' (Arguments) என அழைக்கப்படுகின்றன.

8.4.3 வெளியீடு மதிப்புகள் (Return Values): முடிவுகளைத் திருப்பிக் கொடுத்தல்

ஒரு செயல்பாடு ஒரு வேலையைச் செய்த பிறகு, ஒரு முடிவை அல்லது ஒரு மதிப்பைப் 'பின்னால்' அனுப்ப வேண்டியிருக்கும். இந்த முடிவை அனுப்ப நாம் `return` என்ற சிறப்புச் சொல்லைப் பயன்படுத்துகிறோம். இது ஒரு கால்குலேட்டர் கணக்கீட்டைச் செய்து, நமக்கு விடையைத் திருப்பிக் கொடுப்பது போல.

Python

```
def add_numbers(a, b): # 'a' மற்றும் 'b' அளவுருக்கள்
    sum_result = a + b
    return sum_result # கணக்கிடப்பட்ட மதிப்பைத் திருப்பிக் கொடுக்கிறோம்

# செயல்பாட்டை அழைக்கிறோம், அதன் வெளியீட்டை ஒரு மாறியில் சேமிக்கிறோம்
result1 = add_numbers(5, 3)
print(f"கூடுதல்: {result1}") # Output: கூடுதல்: 8

result2 = add_numbers(10, 20)
print(f"கூடுதல்: {result2}") # Output: கூடுதல்: 30

# return செய்யாத செயல்பாடுகள்:
def say_hello():
    print("ஹலோ!")

output = say_hello()
print(f"say_hello செயல்பாட்டின் வெளியீடு: {output}") # Output: ஹலோ! \n say_hello செயல்பாட்டின் வெளியீடு: None
```

விளக்கம்:

add_numbers செயல்பாடு a மற்றும் b என்ற இரண்டு எண்களை உள்ளீடாகப் பெற்று, அவற்றைக் கூட்டி, அந்த முடிவை return செய்கிறது. return செய்யப்படாத செயல்பாடுகள், இயல்பாகவே None (பைதான் மொழியில் 'எதுவும் இல்லை' என்பதற்கான மதிப்பு) என்ற மதிப்பைத் திருப்பிக் கொடுக்கும்.

8.4.4 அளவுருக்களின் வகைகள் (Types of Arguments): நுட்பமான தகவல்கள்

பைதான் செயல்பாடுகளுக்கு அளவுருக்களை அனுப்பப் பல வழிகளை வழங்குகிறது, இது உங்கள் குறியீட்டை மேலும் நெகிழ்வாகவும், படிக்க எளிதாகவும் மாற்றும்:

- **நிலைப் அளவுருக்கள் (Positional Arguments):** நீங்கள் செயல்பாட்டை அழைக்கும்போது, அளவுருக்களின் வரிசை மிக முக்கியம். பைதான், நீங்கள் அனுப்பும் மதிப்புகளை, வரையறுக்கப்பட்ட அளவுருக்களின் வரிசையின் அடிப்படையில் ஒதுக்கும்.

  Python

  ```
  def describe_person(name, age):
      print(f"பெயர்: {name}, வயது: {age}")
  describe_person("ஆதித்யா", 30) # 'ஆதித்யா' name-க்கும், 30 age-க்கும் செல்லும்
  ```

- **விசைச்சொல் அளவுருக்கள் (Keyword Arguments):** நீங்கள் அளவுருவின் பெயரைத் தெளிவாகக் குறிப்பிட்டு மதிப்பை அனுப்பலாம். இதனால் வரிசை முக்கியமில்லை, குறியீடு தெளிவாகப் புரியும்.

  Python

  ```
  describe_person(age=30, name="ஆதித்யா") # வரிசை மாறினாலும், பெயரைச் சொன்னதால் குழப்பமில்லை
  ```

- **இயல்புநிலை அளவுருக்கள் (Default Arguments):** ஒரு அளவுருவுக்கு நீங்கள் ஒரு இயல்புநிலை மதிப்பை (default value) ஒதுக்கலாம். அப்படிச் செய்தால், அந்த அளவுருவுக்கு ஒரு மதிப்பை நீங்கள் அனுப்பாதபோது, இயல்புநிலை மதிப்பு பயன்படுத்தப்படும்.

  Python

  ```
  def greet_with_default(name, message="நல்வரவு!"):
      print(f"{name}, {message}")
  greet_with_default("கவிதா")           # Output: கவிதா, நல்வரவு!
  greet_with_default("ரவி", "மகிழ்ச்சி!") # Output: ரவி, மகிழ்ச்சி!
  ```

(மேம்பட்ட தலைப்புகளான `*args` மற்றும் `**kwargs` பற்றி, அவை பல அளவுருக்களைக் கையாளப் பயன்படும் சிறப்பு நுட்பங்கள் என்பதை மட்டும் இப்போதைக்குப் புரிந்துகொண்டால் போதும். அவற்றை வரும் அத்தியாயங்களில் விரிவாகக் காணலாம்.)

8.4.5 மாறிகளின் நோக்கம் (Scope of Variables): எல்லைகளைப் புரிந்துகொள்ளுதல்

ஒரு மாறிகை (variable) நீங்கள் எங்கு வரையறுக்கிறீர்கள் என்பதைப் பொறுத்து, அதை உங்கள் நிரலின் எந்தெந்தப் பகுதிகளில் அணுகலாம் என்பது மாறுபடும். இதை **‘மாறிகளின் நோக்கம்’ (Scope of Variables)** என்று அழைக்கிறோம்.

- **உள்ளூர் மாறிகள் (Local Variables):** ஒரு செயல்பாட்டின் உள்ளே வரையறுக்கப்பட்ட மாறிகள், அந்தச் செயல்பாட்டிற்கு உள்ளே மட்டுமே அணுக முடியும். செயல்பாட்டிற்கு வெளியே அவை இருப்பதில்லை.

  Python

  ```
  def my_function():
      local_var = "நான் ஒரு உள்ளூர் மாறி" # local_var என்பது உள்ளூர் மாறி
      print(local_var)

  my_function()
  # print(local_var) # இது பிழை ஏற்படுத்தும்! (NameError)
  ```

- **உலகளாவிய மாறிகள் (Global Variables):** எந்தச் செயல்பாட்டிற்கும் வெளியே, ஒரு நிரலின் மிக உயர்ந்த மட்டத்தில் வரையறுக்கப்பட்ட மாறிகள், நிரலின் எந்தப் பகுதியிலிருந்தும் அணுக முடியும்.

  Python

```
global_var = "நான் ஒரு உலகளாவிய மாறி" # global_var என்பது உலகளாவிய மாறி

def another_function():
    print(global_var) # செயல்பாட்டிற்கு உள்ளேயும் அணுகலாம்

another_function()
print(global_var) # செயல்பாட்டிற்கு வெளியேயும் அணுகலாம்
```

முக்கியக் குறிப்பு: உலகளாவிய மாறிகளைச் செயல்பாடுகளுக்கு உள்ளே மாற்றுவதைத் (modifying) தவிர்ப்பது நல்லது. இது உங்கள் குறியீட்டைக் குழப்பமானதாகவும், பிழைகளைக் கண்டறிவதைக் கடினமாக்கும். முடிந்தவரை, தகவல்களை அளவுருக்கள் மூலம் செயல்பாடுகளுக்கு அனுப்பவும், `return` மூலம் முடிவுகளைத் திருப்பப் பெறவும் பழகுங்கள்.

8.4.6 Docstrings: செயல்பாடுகளுக்கு விளக்கம் எழுதுதல்

உங்கள் செயல்பாடுகள் என்ன வேலை செய்கின்றன, என்ன உள்ளீடுகளைப் பெறுகின்றன, என்ன வெளியீடுகளைத் தருகின்றன என்பதை மற்ற புரோகிராமர்கள் (அல்லது எதிர்காலத்தில் நீங்கள்!) புரிந்துகொள்ள, **Docstrings** (Documentation Strings) பயன்படுத்தலாம். இவை செயல்பாட்டின் வரையறைக்குக் கீழே, மூன்று மேற்கோள்களுக்குள் (`"""Docstring"""`) எழுதப்படும்.

Python

```
def multiply_numbers(x, y):
    """
    இந்தச் செயல்பாடு இரண்டு எண்களைப் பெருக்கி, அதன் முடிவைத் திருப்பிக் கொடுக்கும்.

    அளவுருக்கள்:
        x (int/float): முதல் எண்.
        y (int/float): இரண்டாவது எண்.

    வெளியீடு:
        int/float: x மற்றும் y-இன் பெருக்கற்பலன்.
    """
    return x * y

# help() செயல்பாட்டின் மூலம் docstring-ஐப் பார்க்கலாம்
help(multiply_numbers)
```

8.4.7 செயல்பாடுகளின் நன்மைகள்: ஏன் பயன்படுத்த வேண்டும்?

- **குறியீட்டு மறுபயன்பாடு (Code Reusability):** ஒருமுறை எழுதப்பட்ட குறியீட்டைப் பலமுறை பயன்படுத்தலாம்.
- **ஒழுங்கமைப்பு (Modularity/Organization):** பெரிய நிரல்களைச் சிறிய, நிர்வகிக்கக்கூடிய பகுதிகளாகப் பிரிக்கலாம். இது குறியீட்டைப் படிக்கவும், புரிந்துகொள்ளவும் எளிதாக்குகிறது.
- **படிக்க எளிது (Readability):** செயல்பாடுகள் உங்கள் குறியீட்டிற்கு ஒரு தெளிவான கட்டமைப்பைக் கொடுக்கும்.
- **பிழைகளைக் கண்டறிதல் எளிது (Easier Debugging):** ஒரு பிழை ஏற்பட்டால், அது எந்தச் செயல்பாட்டில் ஏற்பட்டது என்பதைக் கண்டறிவது எளிதாக இருக்கும்.
- **குறைவான பிழைகள் (Fewer Bugs):** ஒரே குறியீட்டை மீண்டும் மீண்டும் எழுதுவதைத் தவிர்ப்பதன் மூலம், எழுத்துப்பிழைகள் அல்லது தர்க்கப் பிழைகள் ஏற்படும் வாய்ப்புகள் குறைகின்றன.

செயல்பாடுகள் (Functions) என்பவை பைதான் நிரலாக்கத்தின் முதுகெலும்பாகச் செயல்படுகின்றன. இவை, உங்கள் குறியீட்டை வெறும் கட்டளைகளின் தொகுப்பாக இல்லாமல், **தெளிவான, ஒழுங்கமைக்கப்பட்ட, மறுபயன்பாட்டுக்கு ஏற்ற ஒரு கட்டமைப்பாக** மாற்றுகின்றன. ஒரு புரோகிராமராக, செயல்பாடுகளை திறம்படப் பயன்படுத்துவது உங்கள் குறியீட்டின் தரத்தையும், உங்கள் உற்பத்தித்திறனையும் கணிசமாக உயர்த்தும்.

அடுத்த பகுதியில், நிரல்கள் இயங்கும்போது ஏற்படும் எதிர்பாராத சிக்கல்களை எப்படிப் புரிந்துகொள்வது, கண்டறிவது மற்றும் கையாள்வது என்பதைப் பற்றிப் பார்ப்போம் – அதுதான் **‘பிழைகள் மற்றும் விதிவிலக்குகள்’ (Errors and Exceptions)!**

நாம் இதுவரை, ஒரு நிரலை எப்படி முடிவெடுக்க வைப்பது (`if-elif-else`) என்பதையும், ஒரு குறிப்பிட்ட செயலை எப்படி மீண்டும் மீண்டும் செய்வது (`for` மற்றும் `while` லூப்கள்) என்பதையும் கண்டோம். குறியீட்டை ஒழுங்கமைக்க செயல்பாடுகளை (Functions) எப்படிப் பயன்படுத்தலாம் என்றும் பார்த்தோம். இப்போது, நாம் ஒரு முக்கியமான நிஜ உலக யதார்த்தத்தைப் பற்றிப் பேசப் போகிறோம்: **எல்லாம் எப்போதும் சரியாக நடப்பதில்லை!**

உங்கள் கைப்பேசியில் ஒரு செயலியைப் பயன்படுத்தும்போது, திடீரென்று ‘App crashed’ அல்லது ‘An unexpected error occurred’ என்று செய்தி வருவதைப் பார்த்திருப்பீர்கள். சில சமயங்களில், ஒரு வலைத்தளத்தைப் பயன்படுத்தும்போது ‘Page not found’ என்று வரும். இந்த ‘எதிர்பாராத திருப்பங்கள்’ ஏன் நிகழ்கின்றன? ஒரு புரோகிராமராக, உங்கள் நிரல் இப்படித் திடீரென்று நின்றுவிடாமல், இந்தச் சிக்கல்களை எப்படி புத்திசாலித்தனமாகச் சமாளிக்க வேண்டும்?

இங்குதான் **‘பிழைகள் மற்றும் விதிவிலக்குகள்’ (Errors and Exceptions)** என்ற கருத்துப் படிகிறது. ஒரு நிரல் உண்மையிலேயே நம்பகமானதாகவும், பயனர் நட்புடனும் செயல்பட வேண்டுமானால், இந்த எதிர்பாராத சிக்கல்களை எப்படிப் புரிந்துகொள்வது, கண்டறிவது மற்றும் கையாள்வது என்பது அத்தியாவசியம். இந்த அத்தியாயத்தில், பைதான் நிரல்கள் ஏன் ‘தவறாகப் போகலாம்’, மற்றும் அந்தச் சமயங்களில் அவற்றை எப்படி ‘சீர்செய்யலாம்’ என்று விரிவாகக் கற்கப் போகிறோம்.

8.5 பிழைகள் மற்றும் விதிவிலக்குகள் (Errors and Exceptions): நிரலின் எதிர்பாராத திருப்பங்கள்

ஒரு நிரல் இயங்கும்போது ஏற்படும் சிக்கல்களை நாம் பொதுவாக இரண்டு வகைகளாகப் பிரிக்கலாம்: **பிழைகள் (Errors)** மற்றும் **விதிவிலக்குகள் (Exceptions)**.

8.5.1 பிழைகள் என்றால் என்ன? (What are Errors?)

ஒரு நிரலின் குறியீட்டில் உள்ள 'தவறுகள்'தான் பிழைகள். இவை நிரல் சரியாக இயங்குவதைத் தடுக்கின்றன.

அ. இலக்கணப் பிழைகள் (Syntax Errors):

- **என்றால் என்ன?** இவை பைதான் மொழியின் ‘இலக்கண விதிகளை’ நீங்கள் மீறும்போது ஏற்படும் பிழைகள். நீங்கள் ஒரு வாக்கியத்தில் இலக்கணப் பிழை செய்தால், அதை ஒருவர் புரிந்துகொள்ள முடியாதது போல.
- **எப்போது ஏற்படும்?** நீங்கள் நிரலை இயக்கும் முன்பே, பைதான் மொழிபெயர்ப்பாளர் (interpreter) இந்த இலக்கணப் பிழைகளைக் கண்டறிந்துவிடும். நிரல் ஒரு வரியையும் இயக்காது.
- **உதாரணம்:** நீங்கள் ஒரு அடைப்புக்குறியை மூடுவதை மறந்துவிட்டால், அல்லது `if` அறிக்கைக்குப் பிறகு `:` (கோலன்) வைக்க மறந்துவிட்டால்.

  Python

  ```
  # ஒரு இலக்கணப் பிழை உதாரணம்
  print("Hello world" # ❌ இங்கு அடைப்புக்குறி மூடப்படவில்லை!
  SyntaxError: unexpected EOF while parsing
  ```

இந்த வகையான பிழைகளைச் சரிசெய்வது எளிது, ஏனெனில் பைதான் அவை எங்குள்ளன என்பதைத் தெளிவாகக் காட்டிவிடும்.

ஆ. விதிவிலக்குகள் (Exceptions / Runtime Errors):

- **என்றால் என்ன?** இவை நிரல் இயங்கிக் கொண்டிருக்கும்போது, அதாவது 'Runtime'-ல் ஏற்படும் எதிர்பாராத சிக்கல்கள். இலக்கண ரீதியாகக் குறியீடு சரியாக இருக்கலாம், ஆனால், ஒரு குறிப்பிட்ட சூழ்நிலையில், ஒரு செயலைச் செய்யும்போது பிரச்சனை வரலாம்.
- **எப்போது ஏற்படும்?** ஒரு நிரல் சரியாக எழுதப்பட்டிருந்தாலும், பயனர் தவறான உள்ளீட்டைக் கொடுத்தாலோ, ஒரு கோப்பு இல்லாவிட்டாலோ, அல்லது இணைய இணைப்பு துண்டிக்கப்பட்டாலோ இந்த விதிவிலக்குகள் நிகழலாம்.
- **உதாரணம்:** பூஜ்ஜியத்தால் வகுத்தல், இல்லாத மாறிகளைப் பயன்படுத்துதல், தவறான தரவு வகைகளைக் கையாளுதல் போன்றவை.

  Python

  ```
  # ஒரு விதிவிலக்கு உதாரணம்: பூஜ்ஜியத்தால் வகுத்தல்
  # result = 10 / 0
  # ZeroDivisionError: division by zero
  ```

  ஒரு விதிவிலக்கு ஏற்படும்போது, அதை நாம் சரியாகக் கையாளவில்லை என்றால், நிரல் திடீரென்று நின்று (crash ஆகி) ஒரு பிழைச் செய்தியைக் காட்டும். இந்தச் சூழ்நிலைகளைத் தடுக்கவே விதிவிலக்கு கையாளுதல் (Exception Handling) தேவைப்படுகிறது.

8.5.2 விதிவிலக்குகளைக் கையாளுதல்: `try-except` சூத்திரம்

நிரல் எதிர்பாராத விதிவிலக்குகளால் நின்றுவிடாமல், அவற்றை நாம் புத்திசாலித்தனமாகச் சமாளிக்க பைதான் **`try-except`** தொகுதிகளை வழங்குகிறது. இது ஒரு நிரலுக்கு 'பாதுகாப்புக் கவசம்' அணிவிப்பது போல.

அ. `try` மற்றும் `except` தொகுதிகள்: பாதுகாப்புக் கவசம்

- **`try` தொகுதி:** நீங்கள் இயக்கிப் பார்க்க விரும்பும், ஆனால் ஒருவேளை பிழை ஏற்படுத்தக்கூடிய குறியீட்டை இந்தத் தொகுதியின் உள்ளே எழுதுங்கள்.
- **`except` தொகுதி:** `try` தொகுதியில் ஒரு விதிவிலக்கு ஏற்பட்டால், எந்தவிதப் பிழையும் காட்டாமல், இந்த `except` தொகுதியில் உள்ள குறியீடு இயங்கும்.

கட்டமைப்பு:

Python

```
try:
    # இங்கு விதிவிலக்கு ஏற்பட வாய்ப்புள்ள குறியீட்டை எழுதுங்கள்
except: # விதிவிலக்கு ஏற்பட்டால் இந்த பகுதி இயங்கும்
    # விதிவிலக்கைக் கையாள்வதற்கான குறியீடு (எ.கா: ஒரு பிழைச் செய்தியை அச்சிடுதல்)
```

உதாரணம்: பூஜ்ஜியத்தால் வகுக்கும் சிக்கலைக் கையாளுதல்

Python

```
try:
    result = 10 / 0 # இங்கு ZeroDivisionError ஏற்படும்
    print(f"முடிவு: {result}")
except: # ZeroDivisionError ஏற்பட்டால் இந்தப் பகுதி இயங்கும்
    print("கணிதப் பிழை: ஒரு எண்ணை பூஜ்ஜியத்தால் வகுக்க முடியாது!")

print("நிரல் வெற்றிகரமாகத் தொடர்கிறது.")
```

விளக்கம்: `try` உள்ளே பிழை ஏற்பட்டதால், `except` பகுதி இயங்கியது. நிரல் நின்றுவிடாமல், ஒரு பயனர் நட்பு செய்தியைக் காட்டிவிட்டு, தொடர்ந்து இயங்கியது.

ஆ. குறிப்பிட்ட விதிவிலக்குகளைக் கையாளுதல்:

except என்ற பொதுவான வார்த்தையைப் பயன்படுத்துவதற்குப் பதிலாக, எந்த வகையான விதிவிலக்கைக் கையாள வேண்டும் என்பதை நீங்கள் தெளிவாகக் குறிப்பிடலாம். இது ஒரு குறிப்பிட்ட நோய் வந்தால்தான் ஒரு குறிப்பிட்ட மருந்து கொடுப்பது போல.

Python

```
try:
    num1 = int(input("முதல் எண்ணை உள்ளிடவும்: "))
    num2 = int(input("இரண்டாம் எண்ணை உள்ளிடவும்: "))
    result = num1 / num2
    print(f"முடிவு: {result}")
except ValueError: # பயனர் எண் அல்லாத வேறு எதையாவது உள்ளிட்டால்
    print("தவறு! தயவுசெய்து ஒரு முழு எண்ணை உள்ளிடவும்.")
except ZeroDivisionError: # இரண்டாவது எண் பூஜ்ஜியமாக இருந்தால்
    print("கணிதப் பிழை: பூஜ்ஜியத்தால் வகுக்க முடியாது!")
```

விளக்கம்: `ValueError` அல்லது `ZeroDivisionError` என விதிவிலக்குகளின் வகையை நேரடியாகக் குறிப்பிட்டதால், அதற்கு ஏற்ற பிழைச் செய்தி கிடைக்கும்.

இ. பல விதிவிலக்குகளைக் கையாளுதல்:

ஒரே except தொகுதியில் பல விதிவிலக்குகளையும் கையாளலாம்.

Python

```
try:
    my_list = [1, 2, 3]
    print(my_list[5]) # IndexError ஏற்படும்
    # print(10 / 0) # ZeroDivisionError ஏற்படும்
except (IndexError, ZeroDivisionError): # இரண்டு பிழைகளையும் கையாள்கிறது
    print("பட்டியல் இன்டெக்ஸ் பிழை அல்லது பூஜ்ஜிய வகுத்தல் பிழை ஏற்பட்டது!")
```

ஈ. பொதுவான விதிவிலக்குகளைக் கையாளுதல் (Exception as e):

எந்த வகையான விதிவிலக்கு ஏற்பட்டாலும் அதைப் பிடிக்க, Exception என்ற பொதுவான வகையைப் பயன்படுத்தலாம். as e பயன்படுத்தி, ஏற்பட்ட விதிவிலக்கின் தகவலை e என்ற மாறியில் சேமிக்கலாம்.

Python

```
try:
    data = {"name": "Test"}
    print(data["age"]) # KeyError ஏற்படும்
except Exception as e: # எந்த வகை விதிவிலக்கு ஏற்பட்டாலும் இது பிடிக்கும்
    print(f"ஒரு எதிர்பாராத பிழை ஏற்பட்டது: {e}")
```

எச்சரிக்கை: `except Exception as e` என்பதைப் பொதுவாகப் பயன்படுத்துவதைத் தவிர்க்க வேண்டும். இது சிறிய தவறுகளையும் பெரிய பிழைகளையும் வேறுபடுத்திப் பார்க்க விடாது. முடிந்தவரை குறிப்பிட்ட விதிவிலக்குகளைக் கையாளப் பழகுங்கள்.

உ. else தொகுதி: எல்லாம் சரியாக நடந்தால்... (விரும்பினால்)

try தொகுதியில் எந்த விதிவிலக்கும் ஏற்படவில்லை என்றால் மட்டும், else தொகுதி இயங்கும்.

Python

```
try:
    num = int(input("ஓர் எண்ணை உள்ளிடவும்: "))
except ValueError:
    print("தவறான உள்ளீடு!")
else:
    # try தொகுதி வெற்றிகரமாக முடிந்தால் மட்டுமே இது இயங்கும்
    print(f"நீங்கள் உள்ளிட்ட எண்: {num}")
    print("எந்தப் பிழையும் இல்லை.")
```

ஊ. finally தொகுதி: எப்படியும் நடக்கும் செயல் (விரும்பினால்)

finally தொகுதியில் உள்ள குறியீடு, try, except, else எது இயங்கினாலும், அல்லது ஒரு விதிவிலக்கு ஏற்பட்டாலும், ஏற்படவில்லை என்றாலும், கட்டாயமாக இயங்கும். இது கோப்புகளை மூடுவது, இணைய இணைப்புகளைத் துண்டிப்பது போன்ற 'சுத்தப்படுத்தும்' வேலைகளுக்குப் (cleanup) பயன்படும்.

Python

```
try:
    file = open("my_file.txt", "r") # கோப்பை திறக்கிறோம்
    content = file.read()
    print(content)
except FileNotFoundError:
    print("கோப்பு காணப்படவில்லை!")
finally:
    # கோப்பு திறக்கப்படாவிட்டாலும், திறந்திருந்தால் மூட வேண்டும்
    # இங்கு file என்ற மாறி வரையறுக்கப்படாமல் இருக்கலாம், எனவே கவனமாக இருக்க வேண்டும்
    # (இன்னும் கோப்பு கையாளுதலைக் கற்கவில்லை என்பதால், இது ஒரு எடுத்துக்காட்டு மட்டுமே)
    print("செயல்பாடு முடிந்தது.")
```

விளக்கம்: `finally` என்பது ஒரு செயல் முடிந்த பிறகு, சில கட்டாய வேலைகளைச் செய்ய வேண்டியிருக்கும்போது மிகவும் பயனுள்ளது.

8.5.3 விதிவிலக்குகளை உருவாக்குதல் (`raise` statement): பிரச்சனையை அறிவித்தல்

ஒரு புரோகிராமராக, சில சமயங்களில், உங்கள் நிரலில் ஒரு நிபந்தனை பூர்த்தி செய்யப்படாதபோது அல்லது ஒரு தவறான சூழ்நிலை ஏற்படும்போது, நீங்களே ஒரு விதிவிலக்கை **'உருவாக்க' (`raise`)** வேண்டியிருக்கும். இது ஒரு 'சிவப்புக்கொடி' காட்டுவது போல, 'இங்கு ஒரு பிரச்சனை உள்ளது' என்று அறிவிப்பது.

உதாரணம்: வயது எதிர்மறையாக இருந்தால் பிழை உருவாக்குதல்

Python

```python
def set_age(age):
    if age < 0:
        raise ValueError("வயது எதிர்மறையாக இருக்க முடியாது!") # ValueError விதிவிலக்கை உருவாக்குகிறோம்
    print(f"வயது: {age}")

try:
    set_age(30)
    set_age(-5) # இங்கு ValueError உருவாக்கப்படும்
except ValueError as e:
    print(f"பிழை ஏற்பட்டது: {e}")
```

விளக்கம்: `set_age` செயல்பாட்டில், வயது எதிர்மறையாக இருந்தால், நாம் ஒரு `ValueError` ஐ உருவாக்குகிறோம். இது, `try-except` தொகுதியால் பிடிக்கப்பட்டு, பிழைச் செய்தி அச்சிடப்படும்.

8.5.4 பொதுவான சில பைதான் விதிவிலக்குகள்

நாம் ஏற்கனவே சிலவற்றை உதாரணங்களில் பார்த்தோம், ஆனால் பைதான் இன்னும் பல பொதுவான விதிவிலக்குகளைக் கொண்டுள்ளது:

- **`ValueError`**: செயல்பாட்டிற்கு அனுப்பப்பட்ட தரவின் மதிப்பு சரியாக இல்லாதபோது. (எ.கா: `int("abc")`).
- **`TypeError`**: தவறான தரவு வகைகளைச் செயல்பாடுகளுடன் பயன்படுத்தும்போது. (எ.கா: `10 + "hello"`).
- **`NameError`**: வரையறுக்கப்படாத ஒரு மாறியைப் பயன்படுத்தும்போது. (எ.கா: `print(my_variable)` ஆனால் `my_variable` வரையறுக்கப்படவில்லை).
- **`IndexError`**: ஒரு பட்டியல் அல்லது டியூபிளின் எல்லைக்கு வெளியே உள்ள ஒரு இன்டெக்ஸை அணுக முயற்சிக்கும்போது. (எ.கா: `my_list[10]` ஆனால் பட்டியலில் 5 உறுப்புகள் மட்டுமே உள்ளன).
- **`KeyError`**: அகராதியில் இல்லாத ஒரு விசையை அணுக முயற்சிக்கும்போது. (எ.கா: `my_dict["non_existent_key"]`).
- **`FileNotFoundError`**: ஒரு கோப்பைப் படிக்கவோ அல்லது எழுதவோ முயற்சிக்கும்போது, அந்தக் கோப்பு இல்லாவிட்டால்.
- **`ZeroDivisionError`**: ஒரு எண்ணைப் பூஜ்ஜியத்தால் வகுக்கும்போது.

8.5.5 விதிவிலக்கு கையாளுதலின் நன்மைகள்: ஏன் இது முக்கியம்?

விதிவிலக்கு கையாளுதல் என்பது வெறுமனே பிழைகளைச் சமாளிப்பது மட்டுமல்ல. இது உங்கள் நிரல்களை மிகவும் வலிமையானதாகவும், பயனர் நட்புடனும் மாற்றுகிறது:

- **நிரல் உடைவதைத் தடுத்தல் (Preventing Crashes):** ஒரு சிறிய பிழையால் முழு நிரலும் நின்றுவிடுவதைத் தவிர்க்கலாம்.
- **பயனர் அனுபவம் (User Experience):** கணினி மொழியில் குழப்பமான பிழைகளைக் காட்டுவதற்குப் பதிலாக,

பயனருக்குப் புரியும் எளிய, வழிகாட்டும் செய்திகளைக் காட்டலாம்.

- **பிழைகளைக் கண்டறிதல் (Debugging):** பிழைகளைச் சரியான இடத்தில் பிடித்து, அவற்றைப் பதிவு செய்வதன் மூலம், சிக்கல்களைக் கண்டறிந்து சரிசெய்வது எளிதாகிறது.
- **குறியீட்டு நம்பகத்தன்மை (Code Reliability):** உங்கள் நிரல் எதிர்பாராத சூழ்நிலைகளிலும் நம்பகத்தன்மையுடன் செயல்படும் என்பதை உறுதி செய்கிறது.

விதிவிலக்கு கையாளுதல் என்பது ஒரு சிறந்த புரோகிராமருக்கு அத்தியாவசியமான ஒரு திறன். இது உங்கள் குறியீட்டைச் சீராகவும், பிழையின்றி இயங்கவும், உண்மையான உலகப் பயன்பாடுகளுக்கு ஏற்றதாகவும் மாற்றும்.

நாம் இதுவரை, ஒரு நிரலை எப்படி முடிவெடுக்க வைப்பது, மீண்டும் மீண்டும் வேலைகளைச் செய்வது, குறியீட்டை ஒழுங்கமைக்க செயல்பாடுகளைப் பயன்படுத்துவது, மற்றும் எதிர்பாராத பிழைகளைச் சமாளிப்பது எப்படி என்று பார்த்தோம். இப்போது, உங்கள் நிரல்கள் தங்கள் வேலைகளை முடித்தவுடன், தகவல்களைப் பாதுகாப்பாகச் சேமித்து வைக்கவும், அல்லது ஏற்கனவே சேமிக்கப்பட்ட தகவல்களைப் படிக்கவும் எப்படிச் செய்வது என்பதைப் பார்ப்போம்.

உங்கள் டைரி, ஒரு புகைப்பட ஆல்பம், அல்லது ஒரு நிறுவனத்தின் வாடிக்கையாளர் பட்டியல் – இவை அனைத்தும் தகவல்களை நிரந்தரமாகச் சேமித்து வைக்கும் இடங்கள். கணினி உலகில், இந்தத் தகவல்கள் பெரும்பாலும் **கோப்புகளில் (Files)** சேமிக்கப்படுகின்றன. உங்கள் நிரல்கள் இந்த கோப்புகளுடன் நேரடியாக உரையாட வேண்டியது அவசியம்.

இங்குதான் **'கோப்பு உள்ளீடு/வெளியீடு' (File Input/Output - File I/O)** என்ற கருத்துப் படிகிறது. இது உங்கள் பைதான் நிரல்களை, உங்கள் கணினியில் உள்ள கோப்புகளிலிருந்து தகவல்களைப் படிக்கவும், அல்லது புதிய தகவல்களைக் கோப்புகளில் எழுதவும், அவற்றை நிர்வகிக்கவும் உதவுகிறது. இந்த அத்தியாயத்தில், பைதான் எப்படி கோப்புகளுடன் 'பேசுகிறது' என்று விரிவாகக் கற்கப் போகிறோம்.

8.6 கோப்பு உள்ளீடு/வெளியீடு (File I/O): நிரந்தரத் தகவல்களைக் கையாள்தல்

நிரல்கள் இயங்கும்போது உருவாக்கும் தகவல்கள், நிரல் நின்றுவிட்டால் அழிந்துவிடும். இந்தத் தகவல்களை நிரந்தரமாகச் சேமிக்க அல்லது ஏற்கனவே உள்ள தரவுகளைப் பயன்படுத்த, நாம் கோப்புகளைப் பயன்படுத்துகிறோம்.

8.6.1 கோப்புகளைத் திறத்தல் (Opening Files): கதவைத் திறக்கும் `open()` செயல்பாடு

ஒரு கோப்பில் படிக்கவோ அல்லது எழுதவோ செய்வதற்கு முன், நீங்கள் அந்தக் கோப்பைத் 'திறக்க' வேண்டும். இதற்கு பைதான் `open()` என்ற உள்ளமைக்கப்பட்ட செயல்பாட்டைப் பயன்படுத்துகிறது. `open()` செயல்பாடு இரண்டு முக்கிய அளவுருக்களை (arguments) எடுக்கும்:

1. **கோப்பின் பெயர் (filename):** நீங்கள் திறக்க விரும்பும் கோப்பின் பெயர் (அல்லது கோப்பின் முழுப் பாதை).
2. **பயன்முறை (mode):** நீங்கள் கோப்பை என்ன நோக்கத்திற்காகத் திறக்கிறீர்கள் (படிக்கவா, எழுதவா, சேர்க்கவா?).

கோப்புப் பயன்முறைகள் (File Modes):

பயன்முறை	விளக்கம்
`'r'`	**படித்தல் (read):** கோப்பிலிருந்து தகவல்களைப் படிக்க. (இயல்புநிலை)
`'w'`	**எழுதுதல் (write):** கோப்பில் தகவல்களை எழுத. (கோப்பு ஏற்கனவே இருந்தால், அதன் உள்ளடக்கத்தை அழித்துவிட்டுப் புதியதாக எழுதும். கோப்பு இல்லாவிட்டால், புதிய கோப்பை உருவாக்கும்).
`'a'`	**சேர்த்தல் (append):** கோப்பின் இறுதியில் புதிய தகவல்களைச் சேர்க்க. (கோப்பு இல்லாவிட்டால், புதிய கோப்பை உருவாக்கும்).
`'x'`	**பிரத்தியேகப் படைப்பு (exclusive creation):** புதிய கோப்பை மட்டுமே உருவாக்கும். கோப்பு ஏற்கனவே இருந்தால், பிழை (`FileExistsError`) ஏற்படுத்தும்.
`'b'`	**பைனரி பயன்முறை (binary mode):** பைனரி கோப்புகளுக்கு (படங்கள், வீடியோக்கள்) `'r'`, `'w'`, `'a'` உடன் பயன்படுத்தலாம் (எ.கா: `'rb'`, `'wb'`).
`'t'`	**உரை பயன்முறை (text mode):** உரை கோப்புகளுக்கு. (இயல்புநிலை)

கட்டமைப்பு:

Python

```
file_object = open("கோப்பின்_பெயர்.txt", "பயன்முறை")
```

உதாரணம்:

Python

```
# ஒரு கோப்பைப் படிப்பதற்காகத் திறத்தல்
# my_file = open("my_document.txt", "r")

# ஒரு கோப்பை எழுதுவதற்காகத் திறத்தல்
# new_file = open("output.txt", "w")
```

8.6.2 கோப்புகளை மூடுதல் (Closing Files): பொறுப்புடன் மூடுதல்

ஒரு கோப்புடன் உங்கள் வேலை முடிந்தவுடன், அந்தக் கோப்பை **மூடுவது (close)** மிக முக்கியம். இதை `close()` method மூலம் செய்யலாம். ஏன் முக்கியம்?

- **தரவு பாதுகாப்பு:** நீங்கள் எழுதிய தகவல்கள் உண்மையில் கோப்பில் சேமிக்கப்படுவதை உறுதி செய்யும்.
- **கணினி வளங்கள்:** கோப்புகள் திறந்திருக்கும்போது கணினியின் வளங்களைப் பயன்படுத்தும்; மூடுவது அந்த வளங்களை விடுவிக்கும்.

Python

```
# ஒரு கோப்பை திறந்து, மூடுதல்
try:
    file = open("example.txt", "w")
    file.write("வணக்கம் பைதான்!")
finally:
    file.close() # கோப்பு திறந்திருந்தாலும், பிழை ஏற்பட்டாலும் இது கட்டாயமாக இயங்கும்
```

8.6.3 `with` அறிக்கை: பாதுகாப்பான கோப்பு கையாளுதல்

கோப்புகளைக் கையாளும்போது, `try-finally` ஐப் பயன்படுத்துவது சற்றுக் கடினம். கோப்பைத் திறந்த பிறகு, அதை மூடுவதை நீங்கள் மறந்துவிடாமல் இருக்க, பைதான் ஒரு சிறந்த, பாதுகாப்பான வழியை வழங்குகிறது: **`with` அறிக்கை (The `with` statement)**.

`with` அறிக்கை ஒரு கோப்பைத் திறந்தவுடன், உங்கள் வேலை முடிந்ததும் தானாகவே அதை மூடிவிடும், நீங்கள் `file.close()` என்று தனியாக எழுத வேண்டியதில்லை. இது ஒரு ‘தானியங்கி மூடும் கதவு’ போல.

கட்டமைப்பு:

Python

```
with open("கோப்பின்_பெயர்.txt", "பயன்முறை") as file_object:
    # கோப்புடன் செய்ய வேண்டிய வேலைகள் இங்கு வரும்
    # இந்த block-ஐ விட்டு வெளியேறியவுடன், கோப்பு தானாகவே மூடப்படும்
```

உதாரணம்:

Python

```
# 'with' அறிவைப் பயன்படுத்தி கோப்பை எழுதுதல்
with open("my_message.txt", "w") as file:
    file.write("பைதான் நிரலாக்கம் எளிது!\n")
    file.write("கோப்பு கையாளுதல் சுலபம்.")

print("தகவல்கள் 'my_message.txt' கோப்பில் எழுதப்பட்டன.")
```

இனிமேல், கோப்புகளைக் கையாளும்போது எப்போதும் `with` அறிக்கையைப் பயன்படுத்துவதுதான் சிறந்த நடைமுறை.

8.6.4 கோப்பிலிருந்து படித்தல் (Reading from Files): தகவல்களைப் பெறுதல்

ஒரு கோப்பிலிருந்து தகவல்களைப் படிக்க, நீங்கள் கோப்பை `'r'` பயன்முறையில் திறக்க வேண்டும்.

- `read()`: கோப்பின் முழு உள்ளடக்கத்தையும் ஒரு பெரிய ஸ்ட்ரிங்காகப் படிக்கும்.
- `readline()`: கோப்பிலிருந்து ஒரு வரியை மட்டும் படிக்கும்.
- `readlines()`: கோப்பின் அனைத்து வரிகளையும் ஒரு பட்டியலாகப் படிக்கும், ஒவ்வொரு வரியும் பட்டியலில் ஒரு தனி ஸ்ட்ரிங்காக இருக்கும்.

உதாரணம்:

Python

```
# முதலில் ஒரு கோப்பை உருவாக்கி அதில் சில தகவல்களை எழுதுவோம்
with open("sample.txt", "w") as f:
    f.write("முதல் வரி தகவல்.\n")
    f.write("இது இரண்டாம் வரி.\n")
    f.write("கடைசி வரி இதுதான்.")

# இப்போது கோப்பிலிருந்து படிப்போம்

# 1. முழு கோப்பையும் ஒரு ஸ்ட்ரிங்காகப் படித்தல்
with open("sample.txt", "r") as file:
    full_content = file.read()
    print("\n--- முழு கோப்பு ---")
    print(full_content)
# Output:
# --- முழு கோப்பு ---
# முதல் வரி தகவல்.
# இது இரண்டாம் வரி.
# கடைசி வரி இதுதான்.

# 2. ஒரு வரியாகப் படித்தல்
with open("sample.txt", "r") as file:
    line1 = file.readline()
    line2 = file.readline()
    print("\n--- ஒரு வரியாக ---")
    print(f"முதல் வரி: {line1.strip()}") # .strip() பயன்படுத்தி புதிய வரி எழுத்தை நீக்குகிறோம்
    print(f"இரண்டாம் வரி: {line2.strip()}")
# Output:
# --- ஒரு வரியாக ---
# முதல் வரி: முதல் வரி தகவல்.
# இரண்டாம் வரி: இது இரண்டாம் வரி.

# 3. அனைத்து வரிகளையும் பட்டியலாகப் படித்தல்
with open("sample.txt", "r") as file:
    all_lines = file.readlines()
    print("\n--- அனைத்து வரிகளும் பட்டியலாக ---")
    for line in all_lines:
        print(line.strip()) # ஒவ்வொரு வரியையும் சுத்தப்படுத்தி அச்சிடுகிறோம்
# Output:
# --- அனைத்து வரிகளும் பட்டியலாக ---
# முதல் வரி தகவல்.
# இது இரண்டாம் வரி.
# கடைசி வரி இதுதான்.
```

கவனிக்க: `read()` அல்லது `readline()` பயன்படுத்தும்போது, பைல் பாயிண்டர் நகரும். ஒருமுறை படித்த பிறகு, மீண்டும் படிக்க `file.seek(0)` பயன்படுத்தி ஆரம்பத்திற்குச் செல்லலாம், அல்லது கோப்பை மீண்டும் திறக்கலாம்.

8.6.5 கோப்பில் எழுதுதல் (Writing to Files): தகவல்களைச் சேமித்தல்

ஒரு கோப்பில் தகவல்களை எழுத, நீங்கள் கோப்பை `'w'` (எழுதுதல்) அல்லது `'a'` (சேர்த்தல்) பயன்முறையில் திறக்க வேண்டும்.

- `write(string)`: கோப்பில் ஒரு ஸ்ட்ரிங்கை எழுதும். இது ஒரு புதிய வரியைச் சேர்க்காது, நீங்கள் `\n` (newline character) ஐச் சேர்க்க வேண்டும்.

- `writelines(list_of_strings)`: ஸ்ட்ரிங்குகளின் ஒரு பட்டியலை கோப்பில் எழுதும். ஒவ்வொரு ஸ்ட்ரிங்கிற்கும் ஒரு புதிய வரி தானாகச் சேர்க்கப்படாது, தேவைப்பட்டால் `\n` ஐ நீங்களே சேர்க்க வேண்டும்.

உதாரணம்:

Python

```
# 'w' பயன்முறை - ஏற்கனவே உள்ள உள்ளடக்கத்தை அழித்துவிட்டுப் புதியதாக எழுதும்
with open("my_notes.txt", "w") as file:
    file.write("என் முதல் குறிப்பு.\n")
    file.write("இது ஒரு புதிய கோப்பு.")

print("புதிய தகவல்கள் 'my_notes.txt' கோப்பில் எழுதப்பட்டன.")
# Output: my_notes.txt கோப்பைத் திறந்தால், 'என் முதல் குறிப்பு.\nஇது ஒரு புதிய கோப்பு.' என்று
இருக்கும்

# 'a' பயன்முறை - கோப்பின் இறுதியில் தகவல்களைச் சேர்க்கும்
with open("my_notes.txt", "a") as file:
    file.write("\nஇது ஒரு கூடுதல் குறிப்பு.") # புதிய வரியில் சேர்க்கிறோம்

print("கூடுதல் தகவல்கள் 'my_notes.txt' கோப்பில் சேர்க்கப்பட்டன.")
# Output: my_notes.txt கோப்பைத் திறந்தால், பழைய உள்ளடக்கத்துடன் 'இது ஒரு கூடுதல் குறிப்பு.'
என்பதும் இருக்கும்
```

8.6.6 கோப்புகளை நீக்குதல் (Deleting Files)

சில சமயங்களில், உங்கள் நிரல் ஒரு கோப்பை உருவாக்கலாம், பயன்படுத்தலாம், பிறகு அதை நீக்க வேண்டியிருக்கலாம். இதற்கு பைதான் `os` (Operating System) என்ற தொகுப்பை வழங்குகிறது.

Python

```
import os

# ஒரு சோதனை கோப்பை உருவாக்குகிறோம்
with open("temp_file.txt", "w") as f:
    f.write("இது ஒரு தற்காலிக கோப்பு.")

# கோப்பு உள்ளதா என்று சரிபார்க்கிறோம்
if os.path.exists("temp_file.txt"):
    os.remove("temp_file.txt") # கோப்பை நீக்குகிறோம்
    print("temp_file.txt வெற்றிகரமாக நீக்கப்பட்டது.")
else:
    print("temp_file.txt ஏற்கனவே இல்லை.")
```

எச்சரிக்கை: `os.remove()` ஐப் பயன்படுத்தும்போது மிகக் கவனமாக இருங்கள், ஏனெனில் நீக்கப்பட்ட கோப்புகளை மீட்டெடுப்பது கடினம்.

கோப்பு உள்ளீடு/வெளியீடு (File I/O) என்பது உங்கள் பைதான் நிரல்களை உண்மையான உலகத் தரவுகளுடன் 'பேச' அனுமதிக்கும் ஒரு சக்திவாய்ந்த கருவியாகும். தகவல்களைப் நிரந்தரமாகச் சேமிக்கவும், பெரிய தரவுத்தொகுப்புகளைப் படிக்கவும், அறிக்கைகளை உருவாக்கவும் இது அத்தியாவசியமானது.

இந்த அத்தியாயத்தில், பைதான் நிரல்கள் எப்படி முடிவெடுக்கின்றன, வேலைகளை மீண்டும் மீண்டும் செய்கின்றன, குறியீட்டை ஒழுங்கமைக்கின்றன, பிழைகளைச் சமாளிக்கின்றன, மற்றும் கோப்புகளுடன் தொடர்பு கொள்கின்றன என்பதை நாம் விரிவாகக் கற்றோம். இந்த அடிப்படைக் கட்டுப்பாட்டு ஓட்டம், உங்கள் நிரல் எப்படிச் செயல்பட வேண்டும் என்பதை வரையறுக்கிறது.

அடுத்த அத்தியாயத்தில், நாம் இதுவரை கற்றுக்கொண்ட அடிப்படைக் கருத்துக்களைப் பயன்படுத்தி, பைதான் புரோகிராமிங்கின் அடுத்த பெரிய படிக்குச் செல்வோம்: **'பொருள் சார்ந்த நிரலாக்கம்' (Object-Oriented Programming - OOP)**. இது உங்கள் குறியீட்டை மேலும் கட்டமைக்கப்பட்டதாகவும், நிர்வகிக்கக்கூடியதாகவும் மாற்றும் ஒரு புதிய சிந்தனை முறை.

9. பைதானில் பொருள் சார்ந்த நிரலாக்கம் (Object-Oriented Programming - OOP): குறியீட்டு உலகை வடிவமைத்தல்

நாம் இதுவரை, ஒரு நிரலை எப்படி முடிவெடுக்க வைப்பது (`if-elif-else`), வேலைகளை மீண்டும் மீண்டும் செய்வது (`for`, `while` லூப்கள்), குறியீட்டை ஒழுங்கமைக்க செயல்பாடுகளைப் பயன்படுத்துவது, எதிர்பாராத பிழைகளைச் சமாளிப்பது, மற்றும் கோப்புகளுடன் தொடர்பு கொள்வது எப்படி என்று விரிவாகக் கற்றோம். இவை அனைத்தும் ஒரு நிரலை உருவாக்குவதற்கான அத்தியாவசிய அடிப்படைக் கட்டுமானத் தொகுதிகள்.

ஆனால், உங்கள் நிரல் பெரியதாக வளரும்போது என்ன ஆகும்? நூற்றுக்கணக்கான அல்லது ஆயிரக்கணக்கான வரிகள் கொண்ட குறியீட்டை நீங்கள் எழுத வேண்டியிருக்கும்போது, அது ஒரு பெரிய குழப்பமான சுவரொட்டி போல மாறிவிடும். ஒவ்வொரு பகுதியும் மற்றொன்றைச் சார்ந்து, ஒரு சிறிய மாற்றம் கூடப் பெரும் பிழைகளை ஏற்படுத்தலாம். இதை நிர்வகிப்பது மிகவும் கடினமாகிவிடும்.

இங்குதான் **பொருள் சார்ந்த நிரலாக்கம் (Object-Oriented Programming - OOP)** என்ற சக்திவாய்ந்த சிந்தனை முறை பைதான் மொழியில் கைகொடுக்கிறது. OOP என்பது, உங்கள் குறியீட்டை ஒரு புதிய, மிகவும் ஒழுங்கமைக்கப்பட்ட வழியில் கட்டமைப்பதற்கான ஒரு அணுகுமுறையாகும். இது ஒரு பெரிய கட்டிடத்தை வெறும் செங்கற்கள் மற்றும் சிமெண்ட்டைப் பயன்படுத்தி உருவாக்குவதற்குப் பதிலாக, ஏற்கனவே வடிவமைக்கப்பட்ட சமையலறைகள், படுக்கையறைகள், வரவேற்பறைகள் போன்ற பகுதிகளை உருவாக்கி, அவற்றை ஒன்றிணைத்து ஒரு முழுமையான வீட்டை உருவாக்குவது போல.

OOP, நம் நிஜ உலகில் உள்ள பொருட்களைப் போலவே, கணினி நிரல்களில் 'பொருட்களை' (Objects) உருவாக்குவதைச் சுற்றி இயங்குகிறது. இந்தப் பொருட்கள், தங்களுக்குரிய தகவல்களையும் (தரவு - data) மற்றும் அந்தத் தகவல்களின் மீது செயல்படும் திறன்களையும் (செயல்பாடுகள் - functions/methods) ஒருங்கே கொண்டிருக்கும்.

இந்த அத்தியாயத்தில், OOP என்றால் என்ன, அதன் அடிப்படைக் கருத்துகள் என்ன, அவை எப்படி உங்கள் குறியீட்டை மேலும் சக்திவாய்ந்ததாகவும், நிர்வகிக்கக்கூடியதாகவும் மாற்றும் என்று விரிவாகக் கற்கப் போகிறோம்.

9.1 ஏன் OOP? (Why OOP?) - ஒரு புதிய அணுகுமுறை

பாரம்பரியமான, **செயல்முறை சார்ந்த நிரலாக்கத்தில் (Procedural Programming)**, நாம் குறியீட்டை ஒரு தொடர்ச்சியான வழிமுறைகளாக எழுதுவோம். தரவுகள் ஒரு இடத்தில் இருக்கும், அந்தத் தரவுகளின் மீது செயல்படும் செயல்பாடுகள் வேறு இடங்களில் இருக்கும். சிறிய நிரல்களுக்கு இது நன்றாக வேலை செய்யும். ஆனால், நிரல் பெரியதாக வளரும்போது:

- **குறியீட்டுக் குழப்பம் (Code Spaghetti):** குறியீடு ஒரு பெரிய 'நூல் பந்து' போலச் சிக்கலாகிவிடும், புரிந்துகொள்வது கடினம்.
- **மறுபயன்பாடு கடினம் (Hard to Reuse):** ஒரு செயல்பாட்டை மற்றொரு திட்டத்தில் பயன்படுத்தும்போது, அதற்குத் தேவையான தரவுகள் அல்லது மற்ற செயல்பாடுகளையும் இழுத்து வர வேண்டியிருக்கும்.

- **பராமரிப்புச் சிக்கல் (Maintenance Hell):** ஒரு சிறிய மாற்றம்கூட நிரலின் மற்ற பகுதிகளைப் பாதிக்கலாம், பிழைகளைக் கண்டறிவது கடினமாகிவிடும்.

இந்தச் சிக்கல்களைத் தீர்க்கவே OOP உதவுகிறது. இது நிஜ உலகப் பொருட்களைப் போலவே குறியீட்டையும் ஒழுங்கமைக்க ஒரு வழிமுறையை வழங்குகிறது.

9.2 OOP-இன் அடிப்படைக் கருத்துகள் (Core Concepts of OOP)

OOP நான்கு முக்கியத் தூண்களின் மீது நிற்கிறது: **கிளாஸ் (Class), ஆப்ஜெக்ட் (Object), என்கேப்சுலேஷன் (Encapsulation), இன்ஹெறிட்டன்ஸ் (Inheritance), பாலிமார்பிசம் (Polymorphism)** மற்றும் **அப்ஸ்ட்ராக்ஷன் (Abstraction)**. இவை ஒவ்வொன்றையும் விரிவாகப் பார்ப்போம்.

9.2.1 கிளாஸ் (Class): ஒரு வரைபடம் அல்லது வார்ப்புரு

- **என்றால் என்ன?** ஒரு **கிளாஸ் (Class)** என்பது ஒரு ஆப்ஜெக்ட்டை உருவாக்குவதற்கான ஒரு வரைபடம் (blueprint), ஒரு வார்ப்புரு (template), அல்லது ஒரு மாதிரி (model) ஆகும். இது ஒரு ஆப்ஜெக்ட் என்னென்ன பண்புகளைக் கொண்டிருக்கும் (தரவு/attributes) மற்றும் என்னென்ன வேலைகளைச் செய்யும் (செயல்பாடுகள்/methods) என்பதை வரையறுக்கிறது.
- **உதாரணம்:** நீங்கள் ஒரு கார் தொழிற்சாலையில் வேலை செய்கிறீர்கள் என்று வைத்துக்கொள்வோம். `Car` என்ற ஒரு கிளாஸ் என்பது, ஒரு கார் எப்படி இருக்க வேண்டும் (நிறம், மாடல், வேகம்) மற்றும் அது என்ன வேலைகளைச் செய்ய முடியும் (வேகப்படுத்துதல், பிரேக் அடித்தல், ஸ்டார்ட் செய்தல்) என்பதை வரையறுக்கும் ஒரு பொறியியல் வரைபடம் (engineering blueprint) போல. அந்த வரைபடத்திலிருந்து நீங்கள் எத்தனை கார்களை வேண்டுமானாலும் தயாரிக்கலாம்.
- **கட்டமைப்பு:** `class` என்ற சிறப்புச் சொல்லைப் பயன்படுத்தி கிளாஸ்கள் வரையறுக்கப்படுகின்றன.

  Python

  ```
  class Car: # 'Car' என்ற கிளாஸை உருவாக்குகிறோம்
      # கிளாஸின் உள்ளே அதன் பண்புகள் (attributes) மற்றும் செயல்பாடுகள் (methods) வரும்
      pass # இப்போதைக்கு ஒன்றும் இல்லை, பின்னர் சேர்ப்போம்
  ```

- **பண்புகள் (Attributes - தரவு):** ஒரு கிளாஸால் உருவாக்கப்பட்ட ஆப்ஜெக்ட் கொண்டிருக்கும் தரவுகள் அல்லது குணாதிசயங்கள். இவை மாறிகள் (variables) போலச் செயல்படும்.
- **செயல்பாடுகள் / மெத்தட்ஸ் (Methods - நடத்தை):** ஒரு கிளாஸால் உருவாக்கப்பட்ட ஆப்ஜெக்ட் செய்யக்கூடிய வேலைகள் அல்லது செயல்பாடுகள். இவை சாதாரண செயல்பாடுகளைப் போலவே இருக்கும், ஆனால் கிளாஸின் உள்ளே வரையறுக்கப்படும்.
- `self` **அளவுரு:** கிளாஸின் உள்ளே ஒரு செயல்பாட்டை வரையறுக்கும்போது, முதல் அளவுருவாக எப்போதும் `self` என்று இருக்க வேண்டும். இது, அந்தச் செயல்பாடு எந்த **ஆப்ஜெக்ட்டின்** மீது செயல்படுகிறது என்பதைக் குறிக்கும். இது ஒரு குறிப்பிட்ட சமையல் குறிப்பு, 'இந்த பாத்திரத்தில் உள்ள கலவை' என்று சொல்வது போல.
- `__init__` **method (கட்டுமானி - Constructor):**
  - இது ஒரு சிறப்புச் செயல்பாடு. ஒரு கிளாஸிலிருந்து ஒரு புதிய ஆப்ஜெக்ட்டை உருவாக்கும்போது, இந்த `__init__` method தானாகவே இயங்கும்.
  - ஆப்ஜெக்ட் உருவாகும்போது, அதற்கு ஆரம்பகட்ட மதிப்புகளை (attributes) வழங்குவதற்கு இது பயன்படுகிறது. இது ஒரு கார் தொழிற்சாலையில், புதிய கார் ஒன்று அசம்பிள் ஆகும்போது, அதற்கு மாடல் எண், நிறம் போன்றவற்றை ஒதுக்குவது போல.
  - இதையும் முதல் அளவுருவாக `self` கொண்டிருக்கும்.

உதாரணம்: `Car` கிளாஸை உருவாக்குதல்

Python

```python
class Car:
    def __init__(self, brand, model, year): # __init__ method - ஆப்ஜெக்ட் உருவாகும்போது இயங்கும்
        self.brand = brand   # brand என்பது Car ஆப்ஜெக்ட்டின் ஒரு பண்பு (attribute)
        self.model = model   # model என்பது Car ஆப்ஜெக்ட்டின் ஒரு பண்பு
        self.year = year     # year என்பது Car ஆப்ஜெக்ட்டின் ஒரு பண்பு

    def display_info(self): # display_info என்பது Car ஆப்ஜெக்ட்டின் ஒரு செயல்பாடு (method)
        print(f"இது ஒரு {self.year} ஆம் ஆண்டு {self.brand} நிறுவனத்தின் {self.model} கார்.")

    def age(self): # காரின் வயதைக் கணக்கிடும் செயல்பாடு
        current_year = 2023 # உதாரணமாக
        return current_year - self.year
```

9.2.2 ஆப்ஜெக்ட் (Object): வரைபடத்திலிருந்து உருவான நிஜப் பொருள்

- **என்றால் என்ன?** ஒரு **ஆப்ஜெக்ட் (Object)** (அல்லது இன்ஸ்டன்ஸ் - Instance) என்பது ஒரு கிளாஸிலிருந்து உருவாக்கப்பட்ட ஒரு நிஜமான, செயல்படக்கூடிய பொருள். ஒரு வரைபடத்திலிருந்து உருவாக்கப்பட்ட ஒரு நிஜக் கார் போல. ஒவ்வொரு ஆப்ஜெக்ட்டும் கிளாஸ் வரையறுத்த பண்புகளையும், செயல்பாடுகளையும் கொண்டிருக்கும், ஆனால் அதற்குரிய தனிப்பட்ட மதிப்புகளுடன்.
- **உருவாக்குதல் (Instantiation):** ஒரு கிளாஸிலிருந்து ஆப்ஜெக்ட்டை உருவாக்குவதை **'இன்ஸ்டன்ஷியேஷன்' (Instantiation)** என்று அழைக்கிறோம்.
- **அணுகுதல்:** ஆப்ஜெக்ட்டின் பண்புகள் மற்றும் செயல்பாடுகளைப் பயன்படுத்த, நாம் `.` (டாட்) குறியீட்டைப் பயன்படுத்துவோம்.

உதாரணம்: `Car` ஆப்ஜெக்ட்களை உருவாக்குதல்

Python

```python
# Car கிளாஸிலிருந்து இரண்டு ஆப்ஜெக்ட்களை உருவாக்குகிறோம்
my_car = Car("Toyota", "Camry", 2020)
friends_car = Car("Honda", "Civic", 2018)

# ஆப்ஜெக்ட்களின் பண்புகளை அணுகுகிறோம்
print(f"என் கார்: {my_car.brand} {my_car.model}")             # Output: என் கார்: Toyota Camry
print(f"நண்பரின் கார்: {friends_car.brand} {friends_car.model}") # Output: நண்பரின் கார்: Honda Civic

# ஆப்ஜெக்ட்களின் செயல்பாடுகளை அழைக்கிறோம்
my_car.display_info()      # Output: இது ஒரு 2020 ஆம் ஆண்டு Toyota நிறுவனத்தின் Camry கார்.
friends_car.display_info() # Output: இது ஒரு 2018 ஆம் ஆண்டு Honda நிறுவனத்தின் Civic கார்.

print(f"என் காரின் வயது: {my_car.age()} ஆண்டுகள்.") # Output: என் காரின் வயது: 3 ஆண்டுகள்.
```

விளக்கம்: `my_car` மற்றும் `friends_car` இரண்டும் `Car` கிளாஸிலிருந்து உருவானவை. ஆனால், அவற்றுக்குத் தனித்தனி `brand`, `model`, `year` மதிப்புகள் உள்ளன.

9.2.3 என்கேப்சுலேஷன் (Encapsulation): தகவல்களைப் பாதுகாத்தல்

- **என்றால் என்ன? என்கேப்சுலேஷன் (Encapsulation)** என்பது ஒரு கிளாஸின் தரவுகளையும் (attributes) மற்றும் அந்தத் தரவின் மீது செயல்படும் செயல்பாடுகளையும் (methods) ஒரு ஒற்றை அலகாக (single unit) இணைப்பதாகும். மேலும், கிளாஸின் உள்ளே உள்ள சில நுட்பமான விவரங்களை வெளியுலகிற்கு மறைத்து, அவை நேரடியாக மாற்றப்படுவதைத் தடுப்பதாகும்.
- **உதாரணம்:** ஒரு தொலைக்காட்சிப் பெட்டியின் ரிமோட் கண்ட்ரோல் போல. நீங்கள் வால்யூமைக் கூட்டவோ குறைக்கவோ, சேனலை மாற்றவோ ரிமோட்டில் உள்ள பொத்தான்களை (methods) பயன்படுத்துகிறீர்கள். ஆனால், அந்தப் பொத்தான்களை அழுத்துவதன் மூலம் டிவி-க்குள் என்ன சிக்கலான எலெக்ட்ரானிக்ஸ் வேலை செய்கிறது (மறைக்கப்பட்ட தரவு/செயல்பாடுகள்) என்பதைப் பற்றி நீங்கள் கவலைப்படுவதில்லை. நீங்கள் வெறும் இடைமுகத்தைப் (interface) பயன்படுத்துகிறீர்கள்.
- **நன்மைகள்:**
    - **தரவுப் பாதுகாப்பு (Data Protection):** கிளாஸின் உள்ளே உள்ள தரவுகளை நேரடியாக வெளியிலிருந்து அணுகாமல், அதன் செயல்பாடுகள் மூலம் மட்டுமே அணுகவோ, மாற்றவோ முடியும். இது தரவு தவறாக மாற்றப்படுவதைத் தடுக்கிறது.
    - **குறியீட்டுச் செப்பனிடுதல் (Modularity):** ஒவ்வொரு கிளாஸும் சுயாதீனமாகச் செயல்படுவதால், ஒரு கிளாஸில் ஏற்படும் மாற்றம் மற்ற கிளாஸ்களைப் பாதிக்காது.
    - **பராமரிப்பு எளிது (Easier Maintenance):** கிளாஸின் உள் வேலைகள் எப்படி மாறினாலும், வெளியுலகிற்கு அது தரும் இடைமுகம் மாறாததால், குறியீட்டைப் பராமரிப்பது எளிது.
- **பைதானில் என்கேப்சுலேஷன்:** பைதான் `private` அல்லது `protected` என்ற நேரடியான அணுகல் கட்டுப்பாடுகளைக் கொண்டிருக்கவில்லை. ஆனால், நாம் சில மரபுகளைப் (conventions) பயன்படுத்துகிறோம்:
    - ஒரு பண்பின் பெயருக்கு முன்னால் ஒரு ஒற்றை அண்டர்ஸ்கோர் (`_`) சேர்த்தால் (எ.கா: `_balance`), அது 'தனியார்' (protected) போலவும், வெளியிலிருந்து அதை நேரடியாக அணுகக் கூடாது எனவும் ஒரு குறிப்பைத் தருகிறது.
    - இரண்டு அண்டர்ஸ்கோர்கள் (`__`) சேர்த்தால் (எ.கா: `__password`), அது பைதான் அதை 'தனியார்' (private) போல அணுகும்படி மறைக்கும் (name mangling).

  Python

```
class BankAccount:
    def __init__(self, initial_balance):
        self._balance = initial_balance # _balance என்பது protected போல

    def deposit(self, amount):
        if amount > 0:
            self._balance += amount
            print(f"பணம் செலுத்தப்பட்டது. புதிய இருப்பு: {self._balance}")
        else:
            print("செலுத்தப்படும் தொகை நேர்மறையாக இருக்க வேண்டும்.")

    def get_balance(self): # இருப்பு அறிய ஒரு method
        return self._balance

account = BankAccount(1000)
# print(account._balance) # நேரடியாக அணுகலாம், ஆனால் பொதுவாகத் தவிர்க்கவும்
account.deposit(500)
```

```
print(f"தற்போதைய இருப்பு: {account.get_balance()}")
```

9.2.4 இன்ஹெறிட்டன்ஸ் (Inheritance): பண்புகளைப் பெறுதல்

- **என்றால் என்ன? இன்ஹெறிட்டன்ஸ் (Inheritance)** என்பது ஒரு புதிய கிளாஸ் (குழந்தை கிளாஸ் / child class), ஏற்கனவே உள்ள ஒரு கிளாஸிலிருந்து (பெற்றோர் கிளாஸ் / parent class) அதன் பண்புகளையும் (attributes) மற்றும் செயல்பாடுகளையும் (methods) **பெறும் (inherit)** வழிமுறையாகும். இது ஒரு குழந்தை பெற்றோரிடமிருந்து சில குணாதிசயங்களைப் பெறுவது போல, அல்லது ஒரு 'கார்' என்பது ஒரு 'வாகனத்தின்' ஒரு வகையாகும்.
- **நன்மைகள்:**
    - **குறியீட்டு மறுபயன்பாடு (Code Reusability):** பொதுவான குறியீட்டை மீண்டும் மீண்டும் எழுதத் தேவையில்லை.
    - **ஒழுங்கமைப்பு (Hierarchical Organization):** கிளாஸ்களை ஒரு படிநிலை (hierarchy) அமைப்பில் ஒழுங்கமைக்கலாம்.
    - **குறியீட்டு எளிமை (Reduced Code Duplication):** ஒரே குறியீட்டைப் பல இடங்களில் எழுதுவதைத் தவிர்க்கலாம்.
- **கட்டமைப்பு:** `class ChildClass(ParentClass):`
- **`super().__init__()`:** குழந்தை கிளாஸ், அதன் பெற்றோரின் `__init__` method-ஐ அழைக்க இது உதவுகிறது, பெற்றோரின் பண்புகளையும் ஆரம்பிக்க.

உதாரணம்: `Vehicle` -> `Car` இன்ஹெறிட்டன்ஸ்

Python

```
class Vehicle: # பெற்றோர் கிளாஸ் (Parent Class / Base Class)
    def __init__(self, make, model):
        self.make = make
        self.model = model

    def display_info(self):
        print(f"இது ஒரு {self.make} {self.model} வாகனம்.")

class Car(Vehicle): # குழந்தை கிளாஸ் (Child Class / Derived Class)
    def __init__(self, make, model, num_wheels):
        super().__init__(make, model) # பெற்றோரின் __init__ method-ஐ அழைக்கிறோம்
        self.num_wheels = num_wheels # Car-க்கு மட்டும் உள்ள தனிப்பட்ட பண்பு

    def drive(self):
        print(f"இந்த {self.model} ஓட்டப்படுகிறது.")

# ஆப்ஜெக்ட்களை உருவாக்குகிறோம்
my_vehicle = Vehicle("Generic", "Transport")
my_car = Car("Honda", "Civic", 4)

my_vehicle.display_info() # Output: இது ஒரு Generic Transport வாகனம்.
my_car.display_info()     # Output: இது ஒரு Honda Civic வாகனம். (பெற்றோரிடம் இருந்து பெறப்பட்டது)
my_car.drive()            # Output: இந்த Civic ஓட்டப்படுகிறது. (குழந்தைக்கு மட்டும் உள்ளது)
```

விளக்கம்: `Car` கிளாஸ் `Vehicle` கிளாஸின் அனைத்துப் பண்புகளையும் (make, model) மற்றும் செயல்பாடுகளையும் (`display_info`) பெறுகிறது. அதனுடன், `num_wheels` போன்ற தனது சொந்த பண்புகளையும், `drive()` போன்ற செயல்பாடுகளையும் சேர்க்கிறது.

9.2.5 பாலிமார்பிசம் (Polymorphism): பல வடிவங்களில் ஒரு செயல்பாடு

- **என்றால் என்ன? பாலிமார்பிசம் (Polymorphism)** என்றால் “பல வடிவங்கள்” என்று அர்த்தம். வெவ்வேறு ஆப்ஜெக்ட்கள், ஒரே பெயருள்ள செயல்பாட்டை அழைக்கும்போது, அவை தங்களுக்குரிய தனிப்பட்ட வழியில் பதிலளிக்கும் திறன் இது. இது ஒரு ‘ப்ளே’ (Play) பட்டன் போல. ஒரு மியூசிக் ப்ளேயரில் ‘ப்ளே’ அழுத்தினால் பாடல் பாடும், ஒரு வீடியோ ப்ளேயரில் ‘ப்ளே’ அழுத்தினால் வீடியோ இயங்கும். பட்டன் ஒன்று, செயல்பாடு வெவ்வேறு.
- **நன்மைகள்:**
    - **குறியீட்டு நெகிழ்வுத்தன்மை (Flexibility):** வெவ்வேறு கிளாஸ்களின் ஆப்ஜெக்ட்களை ஒரே மாதிரி கையாள முடியும்.
    - **குறியீட்டு விரிவாக்கம் (Extensibility):** புதிய கிளாஸ்களை எளிதாகச் சேர்த்து, அவை பொதுவான செயல்பாட்டிற்கு ஏற்றவாறு செயல்பட வைக்கலாம்.

உதாரணம்:

Python

```python
class Dog:
    def make_sound(self):
        print("பவ் பவ்")

class Cat:
    def make_sound(self):
        print("மியாவ் மியாவ்")

class Cow:
    def make_sound(self):
        print("மா மா")

def animal_sound(animal): # இந்தச் செயல்பாடு எந்த animal ஆப்ஜெக்ட்டையும் உள்ளீடாகப் பெறலாம்
    animal.make_sound() # ஒரே method-ஐ அழைக்கிறோம்

dog = Dog()
cat = Cat()
cow = Cow()

animal_sound(dog) # Output: பவ் பவ்
animal_sound(cat) # Output: மியாவ் மியாவ்
animal_sound(cow) # Output: மா மா
```

விளக்கம்: `animal_sound` என்ற ஒரு செயல்பாடு, `Dog`, `Cat`, `Cow` என்ற மூன்று வெவ்வேறு கிளாஸ்களின் ஆப்ஜெக்ட்களைப் பெற்றாலும், ஒரே `make_sound()` என்ற method-ஐ அழைக்கிறது. ஆனால், ஒவ்வொரு ஆப்ஜெக்ட்டும் தனக்கே உரிய ஒலியை எழுப்புகிறது. இதுவே பாலிமார்பிசம்.

9.2.6 அப்ஸ்ட்ராக்ஷன் (Abstraction): சிக்கலை மறைத்தல், முக்கிய அம்சங்களை மட்டும் காட்டுதல்

- **என்றால் என்ன? அப்ஸ்ட்ராக்ஷன் (Abstraction)** என்பது சிக்கலான உள்செயல்பாட்டு விவரங்களை மறைத்து, பயனருக்கு (அல்லது குறியீட்டின் மற்ற பகுதிகளுக்கு) மிக முக்கியமான, அத்தியாவசியமான அம்சங்களை மட்டும் காட்டுவதாகும். இது ஒரு காரை ஓட்டுவது போல – நீங்கள் ஸ்டீயரிங், பெடல்கள் (abstracted interface) போன்றவற்றை மட்டுமே பயன்படுத்துகிறீர்கள்; என்ஜின் எப்படி வேலை செய்கிறது என்ற சிக்கலான உள்விவரங்களைப் பற்றி நீங்கள் தெரிந்துகொள்ளத் தேவையில்லை.
- **நன்மைகள்:**
  - **சிக்கலை எளிதாக்குதல் (Simplifies Complexity):** பெரிய அமைப்புகளைச் சிறிய, புரிந்துகொள்ளக்கூடிய பகுதிகளாகப் பிரிக்க உதவுகிறது.
  - **குறியீட்டு கவனம் (Focus on What, Not How):** ஒரு பணி என்ன செய்ய வேண்டும் என்பதில் கவனம் செலுத்த உதவுகிறது, அது எப்படிச் செய்யப்படுகிறது என்பதில் அல்ல.

உதாரணம்: (பைதான் நேரடியாக ‘abstract class’ என்ற கருத்தைக் கொண்டிருக்கவில்லை என்றாலும், `abc` தொகுதியைப் பயன்படுத்தி இதைச் செய்யலாம். எளிமையான விளக்கத்திற்கு, `display_info` method-ஐக் கொண்ட `Vehicle` கிளாஸ் ஒரு நல்ல எடுத்துக்காட்டு.)

Python

```python
class LightSwitch: # மின்விசைப் பெட்டி
    def turn_on(self):
        # உள்ளே நடக்கும் சிக்கலான மின் இணைப்பு விவரங்களை மறைத்து
        # வெளிப்படையாக விளக்கை எரிய வைக்கும் செயல்பாட்டை மட்டும் காட்டுகிறது.
        print("விளக்கு எரிந்தது!")

    def turn_off(self):
        print("விளக்கு அணைந்தது!")

# பயனர் நேரடியாக turn_on() அல்லது turn_off() ஐப் பயன்படுத்துவார்
# மின்விசைப் பெட்டியின் உள்ளே உள்ள சிக்கலான வயரிங் பற்றி கவலைப்பட மாட்டார்.
switch = LightSwitch()
switch.turn_on()
```

விளக்கம்: இங்கு `LightSwitch` கிளாஸ், விளக்கை எரிய வைக்கும் அல்லது அணைக்கும் சிக்கலான மின்சுற்று விவரங்களை மறைத்து, `turn_on()` மற்றும் `turn_off()` என்ற எளிமையான இடைமுகத்தை மட்டும் பயனருக்கு வழங்குகிறது. இதுவே அப்ஸ்ட்ராக்ஷன்.

9.3 பைதானில் OOP-இன் நன்மைகள் (Benefits of OOP in Python)

OOP என்பது வெறும் ஒரு புரோகிராமிங் தந்திரம் அல்ல; இது ஒரு சிந்தனை முறை. இது உங்கள் குறியீட்டைப் பல வழிகளில் மேம்படுத்துகிறது:

- **ஒழுங்கமைக்கப்பட்ட குறியீடு (Organized Code):** குறியீட்டை நிஜ உலகப் பொருட்களைப் போல ஒழுங்கமைக்க உதவுகிறது, இதனால் பெரிய நிரல்களை நிர்வகிப்பது எளிது.
- **குறியீட்டு மறுபயன்பாடு (Code Reusability):** கிளாஸ்கள் மற்றும் இன்ஹெரிட்டன்ஸ் மூலம், ஏற்கனவே எழுதிய குறியீட்டை மீண்டும் மீண்டும் பயன்படுத்தலாம், இது நேரத்தையும் முயற்சியையும் மிச்சப்படுத்துகிறது.
- **பராமரிப்பு எளிது (Easier Maintenance):** குறியீடு ஒழுங்கமைக்கப்பட்டு, ஒவ்வொரு பகுதியும் தனித்தன்மை

வாய்ந்ததாக இருப்பதால், ஒரு பகுதியில் ஏற்படும் மாற்றங்கள் மற்ற பகுதிகளைப் பாதிக்கும் வாய்ப்பு குறைவு.

- **பிழை திருத்தம் எளிது (Easier Debugging):** ஒரு பிழை ஏற்பட்டால், அது எந்த ஆப்ஜெக்ட் அல்லது கிளாஸில் ஏற்பட்டது என்பதைக் கண்டறிவது எளிதாகிறது.
- **அமைப்புச் scalability (Scalability):** உங்கள் நிரல் வளரும்போது, புதிய அம்சங்களைச் சேர்ப்பது அல்லது ஏற்கனவே உள்ளவற்றை மாற்றுவது எளிதாக இருக்கும்.

பொருள் சார்ந்த நிரலாக்கம் (OOP) என்பது பைதான் புரோகிராமிங்கின் ஒரு புரட்சிகரமான சிந்தனை முறை. இது, உங்கள் குறியீட்டை வெறும் வழிமுறைகளின் தொகுப்பாக இல்லாமல், **நிஜ உலகப் பொருட்களைப் போலச் செயல்படும், ஒரு கட்டமைக்கப்பட்ட, ஆற்றல் மிக்க அமைப்பாக** மாற்றுகிறது.

OOP பற்றிய இந்த விரிவான புரிதல், நீங்கள் சிக்கலான மற்றும் பெரிய அளவிலான நிரல்களை (large-scale applications) உருவாக்கும்போது, ஒரு புரோகிராமராக உங்கள் திறனைப் பன்மடங்கு பெருக்கும். இது உங்கள் குறியீட்டை மிகவும் நேர்த்தியாகவும், திறமையாகவும், நிர்வகிக்கக்கூடியதாகவும் மாற்றும்.

அடுத்த அத்தியாயத்தில், நாம் இதுவரை கற்றுக்கொண்ட அனைத்து அடிப்படைக் கருத்துக்களையும் ஒன்றாகப் பயன்படுத்தி, பைதான் புரோகிராமிங்கின் அடுத்த முக்கியமான படிக்குச் செல்வோம்: **‘கணக்கீட்டுச் சிக்கல்கள் மற்றும் அல்காரிதங்கள்’ (Computational Thinking and Algorithms)**!

10. உங்கள் தரவின் சிற்பி – தரவுக் கட்டமைப்புகள் (Data Structures)

நாம் இதுவரை பைதான் மொழியின் அடிப்படைச் ‘சமையல் பொருட்களை’ (தரவு வகைகள்) சேகரித்தோம். பிறகு, அவற்றை எப்படிச் சமைப்பது (கட்டுப்பாட்டு ஓட்டம், செயல்பாடுகள், OOP) என்றும் பார்த்தோம். இப்போது, ஒரு பெரிய சமையலறையில், நீங்கள் ஆயிரக்கணக்கான பொருட்களை எங்கேயும் வைக்க முடியாது, இல்லையா? ஒவ்வொரு பொருளுக்கும் ஒரு தனி இடம், ஒரு குறிப்பிட்ட அமைப்பு வேண்டும். சர்க்கரை ஒரு பாத்திரத்தில், மசாலாப் பொருட்கள் ஒரு ரேக்கில், காய்கறிகள் ஒரு குளிர்சாதனப் பெட்டியில் – இப்படி ஒழுங்கமைத்தால் தான், சமையல் விரைவாகவும், திறமையாகவும் நடக்கும்.

கணினி உலகில், நாம் கையாளும் தகவல்கள், வெறும் எண்களோ அல்லது எழுத்துகளோ மட்டுமல்ல; அவை பெரும்பாலும் பெரிய, சிக்கலான, ஒன்றோடொன்று தொடர்புடைய தரவுகளின் ‘குவியல்’. இந்தப் பெரிய குவியல்களைப் பயனுள்ள வகையில் நிர்வகிக்கவும், அவற்றை அதிவேகமாக அணுகவும், மாற்றி அமைக்கவும் ஒரு சிறப்பான ‘அமைப்பு’ தேவைப்படுகிறது. இங்குதான் தரவுக் கட்டமைப்புகள் (Data Structures) என்ற கருத்துப் படிகிறது.

தரவுக் கட்டமைப்புகள் என்பவை, கணினியின் நினைவகத்தில் (memory) தரவுகளை எப்படி ஒழுங்குபடுத்திச் சேமிப்பது மற்றும் அந்தத் தரவின் மீது என்னென்ன செயல்பாடுகளை (operations) திறம்படச் செய்வது என்பதை வரையறுக்கும் வழிகாட்டிகள். நீங்கள் ஒரு சிக்கலைத் தீர்க்கும்போது (அல்காரிதம் உருவாக்குதல் - அத்தியாயம் 4), அந்த அல்காரிதம் வேலை செய்யும் தரவுவைச் சரியாக ஒழுங்கமைக்கவில்லை என்றால், உங்கள் அல்காரிதம் எவ்வளவுதான் புத்திசாலித்தனமாக இருந்தாலும், அது மெதுவாகவோ அல்லது திறமையற்றதாகவோ மாறிவிடும். எனவே, சரியான அல்காரிதத்திற்குச் சரியான தரவுக் கட்டமைப்பு மிக முக்கியம்!

இந்த அத்தியாயத்தில், பைதான் மொழியில் உள்ளமைக்கப்பட்ட தரவுக் கட்டமைப்புகளின் ஆழமான பயன்பாடுகளையும், அதையும் தாண்டி தரவுகளைப் பகுத்தறிந்து ஒழுங்கமைப்பதற்கான சில முக்கியக் கருத்துகளையும் விரிவாகக் கற்கப் போகிறோம்.

10.1 ஏன் தரவுக் கட்டமைப்புகள்? (Why Data Structures?) – தரவை ஒழுங்கமைக்கும் கலை

உங்கள் அலுவலகத்தில், வாடிக்கையாளர் விவரங்கள் முதல் விற்பனைப் பதிவுகள் வரை அனைத்து ஆவணங்களும் ஒரு பெரிய அறையில் எங்கேயோ கொட்டிக் கிடப்பதாக கற்பனை செய்யுங்கள். உங்களுக்கு ஒரு குறிப்பிட்ட வாடிக்கையாளரின் பில் தேவைப்பட்டால், நீங்கள் அந்த முழு அறையையும் தேட வேண்டியிருக்கும், இல்லையா? இது எவ்வளவு நேரம் எடுக்கும்? ஆனால், அதே ஆவணங்கள் முறையாக ஒரு கோப்பு அமைச்சில் (filing cabinet) தேதி வாரியாகவோ, அகர வரிசைப்படியோ வைக்கப்பட்டிருந்தால், நீங்கள் அதை மிக விரைவாகக் கண்டுபிடித்துவிடுவீர்கள்.

இதுதான் தரவுக் கட்டமைப்புகளின் சாராம்சம்! ஒரு புரோகிராமராக, உங்கள் நோக்கம் வெறும் குறியீட்டை எழுதுவது மட்டுமல்ல; நீங்கள் உருவாக்கும் நிரல்கள் மிக வேகமாகச் செயல்பட வேண்டும், குறைந்த நினைவகத்தைப் பயன்படுத்த வேண்டும், மற்றும் எளிதாக நிர்வகிக்கப்பட வேண்டும். இதைச் சாத்தியமாக்குவது தரவுக் கட்டமைப்புகள்தான். அவை:

- தேடும் வேகத்தை அதிகரித்தல்: ஒரு குறிப்பிட்ட தகவலை மிக விரைவாகக் கண்டறிய.
- சேர்ப்பது மற்றும் நீக்குவது எளிதாக்குதல்: புதிய தரவுகளைச் சேர்ப்பது அல்லது தேவையற்றவற்றை நீக்குவது.
- நினைவகப் பயன்பாட்டை மேம்படுத்துதல்: கணினியின் நினைவகத்தைப் புத்திசாலித்தனமாகப் பயன்படுத்துதல்.
- அல்காரிதத்தை திறமையாக்குதல்: ஒரு சிக்கலைத் தீர்க்கும் அல்காரிதத்தின் செயல்திறனை நேரடியாகப் பாதித்தல்.

10.2 தரவு வகைகள் VS தரவுக் கட்டமைப்புகள் (Data Types vs. Data Structures): ஒரு தெளிவான வேறுபாடு

அத்தியாயம் 1-ல் நாம் int, str, float, bool, list, tuple, set, dict போன்றவற்றை 'தரவு வகைகள்' (Data Types) என்று பார்த்தோம். இவை பைதான் மொழியில் உள்ள ஒரு குறிப்பிட்ட வகை மதிப்பைச் சேமிக்கும் அடிப்படை அலகுகள்.

ஆனால், 'தரவுக் கட்டமைப்புகள்' (Data Structures) என்பவை, இந்தத் தனிப்பட்ட தரவு வகைகளை (எ.கா: ஒரு int, ஒரு str) எப்படி ஒழுங்குபடுத்தி, சேமித்து, நிர்வகிப்பது என்பதற்கான ஒரு முறையீடு அல்லது அமைப்பு ஆகும்.

உதாரணமாக:

- int என்பது ஒரு தரவு வகை (ஒரு தனி எண்).
- list என்பது ஒரு தரவுக் கட்டமைப்பு (பல int அல்லது str மதிப்புகளை வரிசையாகச் சேமிக்கும் ஒரு வழி).

பைதானின் உள்ளமைக்கப்பட்ட list, tuple, set, dict என்பவை வெறும் 'தரவு வகைகள்' மட்டுமல்ல, அவை தாமே மிகவும் சக்திவாய்ந்த அடிப்படை தரவுக் கட்டமைப்புகளும்கூட! இந்த அத்தியாயத்தில், அவற்றின் கட்டமைப்பு ரீதியான முக்கியத்துவத்தையும், அவற்றின் செயல்திறன் பண்புகளையும் இன்னும் ஆழமாகப் பார்ப்போம்.

10.3 பைதானின் உள்ளமைக்கப்பட்ட தரவுக் கட்டமைப்புகள்: ஒரு ஆழமான பார்வை

நாம் ஏற்கனவே கற்றுக்கொண்ட பைதானின் அடிப்படைத் தொகுப்புக் (collection) தரவு வகைகளை, அவை தரவுகளை எப்படி ஒழுங்கமைத்து, அணுகுகின்றன என்ற கண்ணோட்டத்தில் மீண்டும் பார்ப்போம்.

10.3.1 பட்டியல்கள் (list): மாறும் வரிசைத் தொகுப்பு

- அமைப்பு: பட்டியல்கள் என்பவை கணினியின் நினைவகத்தில் அருகருகே (contiguously) சேமிக்கப்படும் உறுப்புகளின் வரிசைத் தொகுப்பு (Sequence). இவை மாறக்கூடியவை (mutable) என்பதால், அவற்றை உருவாக்கிக் கொண்டே மாற்றலாம்.
- எப்போது சிறந்தது?
  - உறுப்புகளின் வரிசை முக்கியம்: ஒரு குறிப்பிட்ட வரிசையில் (எ.கா: வாடிக்கையாளர் ஆர்டர் வரலாறு) தகவல்களைச் சேமிக்க.

  - அடிக்கடி சேர்க்க/நீக்க வேண்டும் (முடிவில்): append() மற்றும் pop() (கடைசி உறுப்பு) செயல்பாடுகள் மிக வேகமானவை.
  - இன்டெக்ஸ் மூலம் அணுகுதல்: ஒரு குறிப்பிட்ட இடத்தில் உள்ள உறுப்பை அதன் இன்டெக்ஸ் மூலம் மிக வேகமாக அணுகலாம் (my_list[5]).
- வரம்புகள்:
  - பட்டியலின் நடுவில் ஒரு உறுப்பைச் சேர்ப்பது அல்லது நீக்குவது (insert/delete in middle) சற்றே மெதுவாக இருக்கும், ஏனெனில் அதற்குப் பின் உள்ள அனைத்து உறுப்புகளையும் கணினி நகர்த்த வேண்டும்.
  - ஒரு குறிப்பிட்ட மதிப்பைத் தேடும்போது (in ஆப்பரேட்டர் அல்லது index() method), அது பட்டியலின் ஒவ்வொரு உறுப்பையும் சரிபார்க்க வேண்டியிருக்கும் (நேர்கோட்டுத் தேடல் - Linear Search), இது பெரிய பட்டியல்களுக்கு மெதுவாக இருக்கும்.

10.3.2 அகராதிகள் (dict): அதிவேக ‘விசை-மதிப்பு’ தேடுதல்

- அமைப்பு: அகராதிகள் என்பவை ‘விசை-மதிப்பு’ (key-value) ஜோடிகளாகத் தரவுகளைச் சேமிக்கின்றன. இவை பெரும்பாலும் ‘ஹாஷ் டேபிள்’ (Hash Table) அல்லது ‘ஹாஷ் மேப்’ (Hash Map) என்ற கருத்துருவின் அடிப்படையில் செயல்படுகின்றன.
- எப்படி அதிவேகமாக வேலை செய்கிறது? (சிறு நுணுக்கம்): ஒவ்வொரு ’விசை’யும் ஒரு ‘ஹாஷ் ஃபங்ஷன்’ (Hash Function) மூலம் ஒரு தனிப்பட்ட ‘ஹாஷ் கோடாக’ (Hash Code) மாற்றப்படும். இந்த ஹாஷ் கோடு, நினைவகத்தில் அந்தக் ‘key-value’ ஜோடி எங்கு சேமிக்கப்பட்டுள்ளது என்பதைக் குறிக்கும் ஒரு ‘முகவரி’ போலச் செயல்படும். இதனால், ஒரு விசையைப் பயன்படுத்தி ஒரு மதிப்பைத் தேடுவது அல்லது சேர்ப்பது அதிவேகமாக (சராசரியாக ஒரே நேரத்தில் - constant time) நடக்கும்.
- எப்போது சிறந்தது?
  - விசையைப் பயன்படுத்தி தகவலைத் தேட (Lookup by Key): ஒரு பெயரைக் கொண்டு ஒரு நபரின் வயது, ஐடி கொண்டு ஒரு பொருளின் விலை போன்றவற்றை மிக வேகமாக அணுக.
  - அடிக்கடி சேர்க்க/நீக்க வேண்டும்: புதிய ஜோடிகளைச் சேர்ப்பது அல்லது நீக்குவது மிக வேகமாக நடக்கும்.
  - தகவல்களைப் பெயரிட்டு ஒழுங்கமைக்க: ஒரு நபரின் முழு சுயவிவரம் அல்லது ஒரு பொருளின் பண்புகள் போன்ற கட்டமைக்கப்பட்ட தரவுகளைச் சேமிக்க.
- வரம்புகள்:
  - விசைகள் தனித்துவமானவையாகவும், மாறாதவையாகவும் இருக்க வேண்டும்.
  - சாதாரண பட்டியல்களைப் போல உறுப்புகளின் வரிசையை (அதாவது இன்டெக்ஸ்) அடிப்படையாகக் கொண்ட அணுகல் இல்லை.

10.3.3 தொகுப்புகள் (set): தனித்துவமானவர்களின் கூட்டம்

- அமைப்பு: தொகுப்புகள், அகராதிகளின் விசைகள் எப்படிச் செயல்படுகின்றனவோ அதே ‘ஹாஷிங்’ (Hashing) முறையைப் பயன்படுத்துகின்றன. ஆனால், இதில் ‘மதிப்புகள்’ இருப்பதில்லை, வெறும் தனித்துவமான ‘விசைகள்’ மட்டுமே இருக்கும்.
- எப்போது சிறந்தது?
  - டூப்ளிகேட் மதிப்புகளை நீக்குதல் (Removing Duplicates): ஒரு பட்டியலில் உள்ள டூப்ளிகேட்களை மிக விரைவாக நீக்க.

  - உறுப்பினர் சரிபார்ப்பு (Membership Testing): ஒரு பெரிய தரவுத் தொகுப்பில் ஒரு குறிப்பிட்ட உறுப்பு உள்ளதா என்று மிக விரைவாகச் சரிபார்க்க (item in my_set).
  - கணிதச் செட் செயல்பாடுகள்: union, intersection, difference போன்ற செயல்பாடுகளைத் திறம்படச் செய்ய.
- வரம்புகள்:
  - உறுப்புகளுக்கு வரிசை இல்லை (unordered).
  - உறுப்புகள் மாறாதவையாக இருக்க வேண்டும்.

10.3.4 டியூபிள்ஸ் (tuple): மாறாத உறுதியான வரிசை

- அமைப்பு: பட்டியல்களைப் போலவே வரிசைப்படுத்தப்பட்டவை, ஆனால் மாறாதவை (immutable). அதாவது, உருவாக்கப்பட்ட பின் மாற்ற முடியாது.
- எப்போது சிறந்தது?
  - தரவு ஒருமைப்பாடு (Data Integrity): ஒருமுறை உருவாக்கிய தகவல்கள் (எ.கா: ஆயத்தொலைவுகள், ஒருவரின் பிறந்த தேதி) மாற்றப்படாமல் பாதுகாக்கப்பட வேண்டும் எனில்.
  - அகராதி விசைகளாகப் பயன்படுத்தல்: தொகுப்புகளைப் போலவே, டியூபிள்கள் மாறாதவை என்பதால் அகராதியின் விசைகளாகப் பயன்படுத்தலாம்.
  - செயல்பாடுகளுக்குப் பாதுகாப்பாகத் தரவை அனுப்புதல்: ஒரு செயல்பாட்டிற்கு (function) உள்ளீடாக ஒரு தொகுப்பை அனுப்பும்போது, அந்தச் செயல்பாடு அதை மாற்றாது என்று உறுதியாக இருக்கலாம்.

10.3.5 ஸ்ட்ரிங்ஸ் (str): எழுத்துகளின் மாறாத வரிசை

- அமைப்பு: ஸ்ட்ரிங்குகள் என்பவை எழுத்துகளின் வரிசைப்படுத்தப்பட்ட, மாறாத (immutable) தொகுப்பு.
- எப்போது சிறந்தது? உரைத் தரவுகளைச் சேமிக்கவும், கையாளவும். ஸ்ட்ரிங்ஸ் மாறாதவை என்பதால், அவற்றைச் செயலாக்குவது பெரும்பாலும் பாதுகாப்பானது மற்றும் கணினி வளங்களைப் பொறுத்தவரை திறமையானது.

10.4 அப்ஸ்ட்ராக்ட் தரவுக் கட்டமைப்புகளின் உலகம் (The World of Abstract Data Structures): வடிவமைப்பின் கருத்துக்கள்

பைதானின் உள்ளமைக்கப்பட்ட (list, dict, set, tuple) தரவு வகைகளைத் தவிர, நிரலாக்க உலகில் சில பொதுவான ‘அப்ஸ்ட்ராக்ட் தரவுக் கட்டமைப்புகள்’ (Abstract Data Structures) உள்ளன. இவை ஒரு குறிப்பிட்ட தரவு சேமிப்பு முறையையும், அதன் மீது செய்யக்கூடிய செயல்பாடுகளையும் வரையறுக்கும் கருத்துக்கள். பைதானில், இந்த அப்ஸ்ட்ராக்ட் கட்டமைப்புகளை நாம் ஏற்கனவே பார்த்த list அல்லது collections தொகுப்பில் உள்ள சிறப்பு வகைகளைப் பயன்படுத்தி உருவாக்கலாம்.

10.4.1 ஸ்டேக் (Stack): கடைசியில் வைத்தால் முதலில் வரும்! (LIFO)

- என்றால் என்ன? ஸ்டேக் என்பது ஒரு தகவலைச் சேர்க்கவும் (push), அகற்றவும் (pop) ஒரு குறிப்பிட்ட விதிமுறையைக் கொண்ட தரவுக் கட்டமைப்பு: LIFO (Last In, First Out) – அதாவது, கடைசியாகச் சேர்க்கப்பட்ட உறுப்புதான் முதலில் அகற்றப்படும். இதை ஒரு தட்டுகளின் அடுக்கைப் போலக் கற்பனை செய்யுங்கள்: நீங்கள் ஒரு புதிய தட்டை வைக்கும்போது, அது அடுக்கின் மேல் வரும். நீங்கள் ஒரு தட்டை எடுக்க விரும்பினால், மேலிருக்கும் தட்டைத்தான் முதலில் எடுக்க முடியும்.
- பயன்பாடுகள்:
  - Undo/Redo செயல்பாடுகள்: ஒரு மென்பொருளில், நீங்கள் கடைசியாகச் செய்த செயலை முதலில்

    'Undo' செய்வது (Ctrl+Z).
  - ஃபங்ஷன் கால்கள் (Function Calls): ஒரு நிரல் செயல்பாடுகளை அழைக்கும் வரிசையை நிர்வகிக்க.
  - பிராக்கெட் சரிபார்ப்பு: ஒரு சமன்பாட்டில் அடைப்புக்குறிகள் சரியாக மூடப்பட்டுள்ளனவா என்று சரிபார்க்க.
- பைதானில்: list ஐப் பயன்படுத்தி ஸ்டேக்கை எளிதாக உருவாக்கலாம்: append() (push) மற்றும் pop() (pop).

  Python

```
my_stack = []
my_stack.append("புத்தகம் 1") # push
my_stack.append("புத்தகம் 2") # push
print(f"ஸ்டேக்: {my_stack}") # Output: ['புத்தகம் 1', 'புத்தகம் 2']

last_book = my_stack.pop() # pop - கடைசியாகச் சேர்க்கப்பட்டதை எடுக்கும்
print(f"எடுக்கப்பட்டது: {last_book}") # Output: எடுக்கப்பட்டது: புத்தகம் 2
print(f"ஸ்டேக்: {my_stack}") # Output: ['புத்தகம் 1']
```

10.4.2 க்யூ (Queue): வரிசையில் முதலில் வந்தால் முதலில் போகலாம்! (FIFO)

- என்றால் என்ன? க்யூ என்பது தகவலைச் சேர்க்கவும் (enqueue), அகற்றவும் (dequeue) ஒரு குறிப்பிட்ட விதிமுறையைக் கொண்ட தரவுக் கட்டமைப்பு: FIFO (First In, First Out) – அதாவது, முதலில் சேர்க்கப்பட்ட உறுப்புதான் முதலில் அகற்றப்படும். இதை ஒரு சினிமா டிக்கெட் வரிசையைப் போலக் கற்பனை செய்யுங்கள்: முதலில் வந்தவர்தான் முதலில் டிக்கெட் வாங்கி வெளியே செல்வார்.
- பயன்பாடுகள்:
  - பணிகளை திட்டமிடுதல் (Task Scheduling): ஒரு கணினிப் பணியாளர் பல வேலைகளை வரிசையாகச் செய்யும்போது.
  - பிரிண்டர் க்யூ (Printer Queue): அச்சுப்பொறியில் நீங்கள் கொடுக்கும் வேலைகள் வரிசையாக அச்சிடப்படுவது.
  - தரவுப் பரிமாற்றம்: ஒரு பகுதியில் இருந்து மற்றொரு பகுதிக்குத் தரவுகள் வரிசையாக அனுப்பப்படும்போது.
- பைதானில்: list ஐப் பயன்படுத்தலாம் (append() மற்றும் pop(0)), அல்லது collections தொகுப்பில் உள்ள deque ஐப் பயன்படுத்துவது மிகவும் திறமையானது.

  Python

```
from collections import deque

my_queue = deque()
my_queue.append("பயனர் A") # enqueue
my_queue.append("பயனர் B") # enqueue
print(f"க்யூ: {my_queue}") # Output: deque(['பயனர் A', 'பயனர் B'])

first_user = my_queue.popleft() # dequeue - முதலில் சேர்க்கப்பட்டதை எடுக்கும்
print(f"எடுக்கப்பட்டது: {first_user}") # Output: எடுக்கப்பட்டது: பயனர் A
print(f"க்யூ: {my_queue}") # Output: deque(['பயனர் B'])
```

10.4.3 ட்ரீஸ் (Trees) மற்றும் கிராஃப்கள் (Graphs): சிக்கலான உறவுகளின் வரைபடங்கள்

என்றால் என்ன?

- ட்ரீ (Tree - மரம்): தகவல்களைப் படிநிலை (hierarchical) வடிவத்தில் ஒழுங்கமைக்கும் தரவுக் கட்டமைப்பு. ஒரு குடும்ப மரம், ஒரு கோப்பு முறைமை (file system) அல்லது ஒரு நிறுவனத்தின் அமைப்புப் படம் போல. ஒவ்வொரு உறுப்புக்கும் ஒரு ‘பெற்றோர்’ (parent) மற்றும் பல ‘குழந்தைகள்’ (children) இருக்கும்.
- கிராஃப் (Graph - வரைபடம்): பொருட்களை (nodes/vertices) மற்றும் அவற்றுக்கிடையேயான உறவுகளை (edges) பிரதிநிதித்துவப்படுத்தும் ஒரு சிக்கலான தரவுக் கட்டமைப்பு. சமூக வலைப்பின்னல்கள் (நண்பர்கள் உறவு), சாலை வரைபடங்கள், இணையப் பக்கங்களின் இணைப்பு போன்றவற்றை இவை குறிக்கும்.
- பயன்பாடுகள்:
    - ட்ரீஸ்: கோப்பு மேலாண்மை அமைப்புகள், XML/HTML ஆவணங்கள், தரவுத்தள குறியீடுகள் (database indexes), மரபு வழிமுறை பகுப்பாய்வு.
    - கிராஃப்கள்: சமூக வலைப்பின்னல் பகுப்பாய்வு, வழி கண்டுபிடிப்புகள் (Google Maps), இணைய தேடுபொறிகள் (search engines).
- பைதானில்: பைதான் நேரடியாக ட்ரீஸ் அல்லது கிராஃப்களுக்கான உள்ளமைக்கப்பட்ட வகைகளைக் கொண்டிருக்கவில்லை. ஆனால், நாம் dict மற்றும் list ஐப் பயன்படுத்தி அவற்றை உருவாக்கலாம், அல்லது networkx போன்ற சிறப்பு நூலகங்களைப் பயன்படுத்தலாம். (இவை மேம்பட்ட தலைப்புகள், ஆனால் கருத்து ரீதியாகப் புரிந்துகொள்வது அவசியம்).

10.5 சரியான தரவுக் கட்டமைப்பைத் தேர்ந்தெடுக்கும் கலை (The Art of Choosing the Right Data Structure)

நீங்கள் ஒரு புரோகிராமராக, ஒரு சிக்கலை அணுகும்போது, சரியான தரவுக் கட்டமைப்பைத் தேர்ந்தெடுப்பது என்பது உங்கள் அல்காரிதத்தின் வெற்றிக்கு அடிப்படையாகும். இது ஒரு சமையலுக்குப் பொருத்தமான பாத்திரத்தைத் தேர்ந்தெடுப்பது போல. தவறான தேர்வு உங்கள் நிரலை மெதுவாக்கலாம் அல்லது சிக்கலாக்கலாம்.

சரியான தரவுக் கட்டமைப்பைத் தேர்ந்தெடுக்க இந்தக் கேள்விகளைக் கேளுங்கள்:

- என்ன வகையான தகவல்களைச் சேமிக்கப் போகிறீர்கள்? (எண்கள், எழுத்துகள், கலவை?)
- தரவுகளின் வரிசை முக்கியமா? (ஆம் என்றால் list அல்லது tuple).
- உறுப்புகள் தனித்துவமாக இருக்க வேண்டுமா? (ஆம் என்றால் set).
- ஒரு ’விசை’யைப் பயன்படுத்தி மதிப்புகளை வேகமாகத் தேட வேண்டுமா? (ஆம் என்றால் dict).
- தரவுகள் மாறக்கூடியவையா (dynamic) அல்லது மாறாதவையா (static)? (list, dict, set என்பவை மாறக்கூடியவை; str, int, float, tuple என்பவை மாறாதவை).
- நீங்கள் எந்தச் செயல்பாடுகளை அதிகம் செய்வீர்கள்? (உறுப்புகளைச் சேர்த்தல், நீக்குதல், தேடுதல், வரிசைப்படுத்துதல், முழுமையாக வலம் வருதல்?)
- நினைவகப் பயன்பாடு ஒரு பிரச்சனையா?

சரியான தரவுக் கட்டமைப்பைத் தேர்ந்தெடுப்பதன் மூலம், நீங்கள் உங்கள் நிரலின் செயல்திறனை (performance) வியக்கத்தக்க வகையில் மேம்படுத்த முடியும்.

10.6 தரவுக் கட்டமைப்புகளின் உண்மையான சக்தி

தரவுக் கட்டமைப்புகள் என்பவை வெறும் தியரி கருத்துகள் அல்ல. அவை, உங்கள் கணினியின் நினைவகத்தில் ஒரு மாஸ்டர் திட்டத்தை உருவாக்குவது போல. இணையதளங்களில் நீங்கள் பார்க்கும் வேகமான தேடல் முடிவுகள், சமூக வலைப்பின்னல்களில் உள்ள சிக்கலான உறவுகள், கூகிள் மேப்ஸில் நீங்கள் கண்டுபிடிக்கும் சிறந்த வழி - இவை அனைத்தும், அல்காரிதம்களுடன் இணைந்து திறமையாகப் பயன்படுத்தப்படும் தரவுக் கட்டமைப்புகளின் விளைவே.