Блок 2.3. Ішкі бағдарламалар

Сабақ 2.3.1. Рекурсия
Сабақ 2.3.2. Жолдар
Сабақ 2.3.3. Сөздіктер
Сабақ 2.3.4. Кортеждер

Сабақ 2.3.1. Рекурсия

Рекурсия – бұл бағдарламалық функцияның өзін-өзі шақыру процесі. Рекурсия бағдарламалаудың маңызды элементі болып табылады, себебі ол көптеген күрделі есептерді жеңіл және тиімді шешуге мүмкіндік береді. Оның негізінде үлкен есептерді кішірейтіп, оларды қайталанатын қадамдар арқылы шешу тәсілі жатыр.

Рекурсия табиғи түрде ағаш құрылымдары, графтар, қайталанатын есептеулер, математикалық теңдеулер сияқты алгоритмдерге бейімделген.

Рекурсияның негізгі ерекшеліктері

- **Базалық жағдай (Base Case)**
  Базалық жағдай – бұл рекурсияның тоқтау нүктесі. Ол рекурсивті функцияның тоқтайтын шартын анықтайды. Базалық жағдай болмаса, функция өзін шексіз шақырып, жадының толып кетуіне әкеледі.
- **Рекурсивті қадам (Recursive Step)**
  Рекурсивті қадам – бұл функцияның өзін-өзі қайта шақыратын бөлігі. Әрбір рекурсивті қадам негізгі есепті жеңілдетуге және базалық жағдайға жақындатуға тиіс.
- **Шығу механизмі**
  Рекурсиядағы барлық шақырулар базалық жағдай орындалғаннан кейін аяқталады. Әрбір аяқталған қадам стекке жиналған нәтижелерді шығарып, есептеу нәтижесін қайтарады.

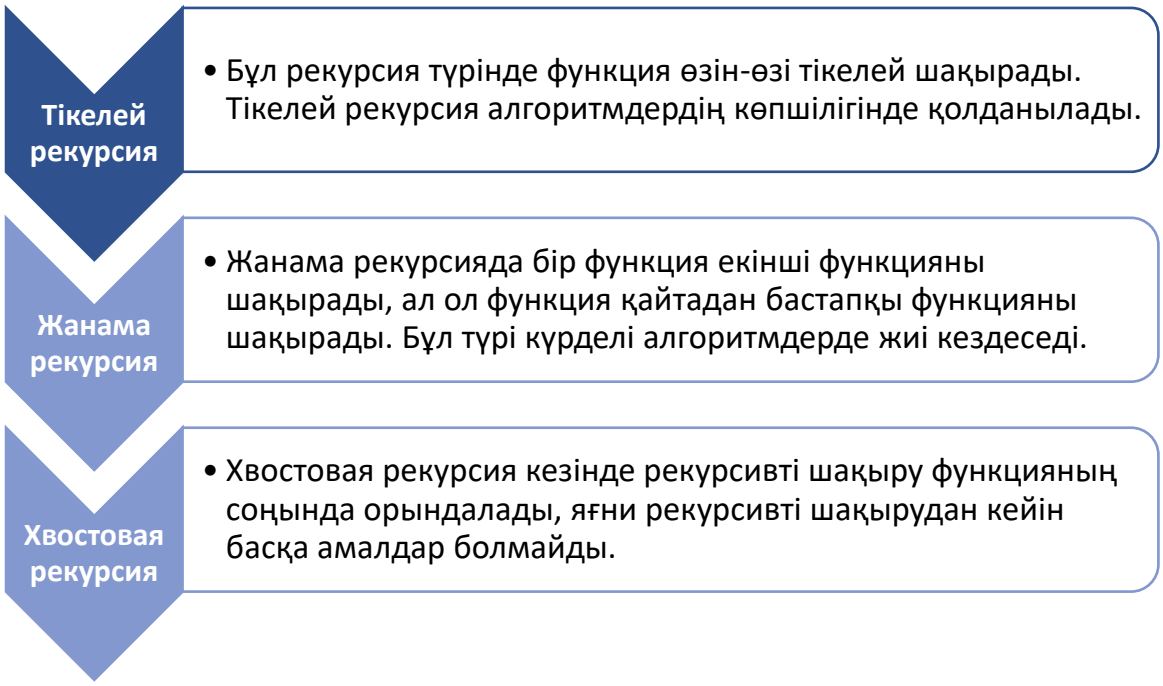
Есте сақта!

- ✓ Әрбір рекурсивті функцияда базалық жағдай нақты анықталуы тиіс.
- ✓ Рекурсивті қадам функцияны базалық жағдайға жақындатуы керек.
- ✓ Рекурсияны тереңдігі шектеулі есептерде қолданған жөн, себебі рекурсия терең болған сайын жады ресурстарын көп пайдаланады.

Рекурсияның жұмыс істеу принципі

Рекурсивті функция шақырылған кезде жаңа деңгей (stack frame) құрылады. Бұл деңгейде функцияның аргументтері және ішкі нәтижелері сақталады. Әрбір жаңа шақыру стекке қосылады. Базалық жағдай орындалған кезде стек босатыла бастайды, нәтижелер біртіндеп қайтарылады.

Рекурсияның түрлері

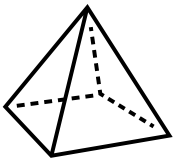


Рекурсияның артықшылықтары мен кемшіліктері

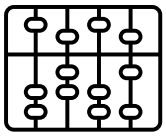
Артықшылықтары	**Кемшіліктері**
Бағдарламаның құрылымын логикалық әрі түсінікті етеді.	Рекурсия көп жадыны пайдаланады, себебі әрбір шақыру стекте жаңа деңгей құрады.
Қайталанатын есептерді шешуде қысқа әрі тиімді код жазуға мүмкіндік береді.	Базалық жағдай дұрыс анықталмаса, бағдарлама шексіз циклға түсіп, жүйенің өнімділігін төмендетеді.
Ағаш құрылымдары мен графтарды өңдеу сияқты алгоритмдерде интуитивті түсінікті шешімдер ұсынады.	Кей жағдайларда циклдер рекурсияға қарағанда тиімдірек болуы мүмкін.

Рекурсияны қолдану салалары

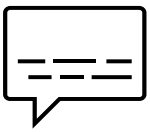
Рекурсия әртүрлі алгоритмдік мәселелерді шешу үшін қолданылады:



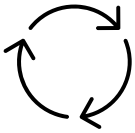
Ағаш және граф алгоритмдері
Рекурсия ағаштардағы және графтардағы түйіндерді іздеу, айналып өту және олардың құрылымын талдауда тиімді әдіс болып табылады.



Математикалық есептер
Рекурсия факториал, Фибоначчи сандары, дәрежелерді есептеу және т.б. математикалық есептерді шешуде жиі қолданылады.



Мәтіндер мен деректерді өңдеу
Рекурсия мәтіннің палиндром екендігін тексеру, тізім элементтерін өңдеу, ішкі құрылымдарды іздеу сияқты тапсырмаларды жеңілдетеді.



Қайталанатын үлгілерді құру
Мысалы, фракталдарды салу немесе белгілі бір үлгілерді бейнелеу рекурсияның қолданыс аясына жатады.

Іздеу алгоритмдері
Бинарлы іздеу, жылдам сұрыптау (QuickSort) және біріктіру сұрыптау (MergeSort) сияқты алгоритмдерде рекурсия негізгі рөл атқарады.

Рекурсияны қолданудағы маңызды кеңестер

- **Базалық жағдайды анықтау**
  Рекурсия жазған кезде бірінші кезекте базалық жағдайды анықтап, оның орындалуын қамтамасыз ету керек. Базалық жағдайдың болмауы бағдарламаның дұрыс жұмыс істемеуіне әкеледі.
- **Ресурстарды басқару**
  Егер рекурсияның тереңдігі өте үлкен болса, циклдік алгоритмдерді немесе басқа әдістерді қолданған жөн. Бұл жадыны үнемдейді және бағдарламаның тұрақтылығын арттырады.
- **Қарапайым есептерден бастау**
  Алдымен шағын және оңай есептерге рекурсияны қолдану арқылы оның жұмыс принципін түсініп алу қажет.

- **Тестілеу және оңтайландыру**
  Рекурсивті функцияны жазғаннан кейін оны әртүрлі жағдайларда тексеріп, мүмкіндігінше тиімділігін арттыру қажет. Кейде мемоизация (есептелген нәтижелерді сақтау) әдісін қолдану арқылы рекурсияның жылдамдығын арттыруға болады.

Рекурсия – бағдарламалаудағы ең қуатты әдістердің бірі. Ол алгоритмдерді жеңілдетіп, кодтың қысқа және түсінікті болуын қамтамасыз етеді. Алайда рекурсияны тиімді пайдалану үшін оның қағидаларын түсініп, жады мен процессор ресурстарын дұрыс басқару қажет. Тиісті білім мен тәжірибе арқылы рекурсия кез келген бағдарламалаушының маңызды құралы бола алады.

Сабақ 2.3.2. Жолдар

Жол – бұл алфавиттік таңбалардың реттелген тізбегі. Python тілінде жолды анықтау үшін бір немесе қос тырнақшаларды ('...' немесе "…") қолданады. Жолдармен жұмыс істеу мәтіндік деректерді сақтау, өңдеу және көрсету үшін кеңінен қолданылады. Егер жолда сандық деректер сақталса, оны арифметикалық операциялар үшін сандық типке түрлендіру қажет.

Жолдың маңызды ерекшеліктері:

Жолды біріктіру үшін + операторы қолданылады.

Жолдағы сандарды арифметикалық есептерде пайдалану үшін оларды int() немесе float() функциялары арқылы сандық типке түрлендіруге болады.

Жолдар кез келген мәтіндік деректерді сақтауға мүмкіндік беріп, пайдаланушының енгізген деректерімен немесе сыртқы ақпарат

көздерінен алынған мәтіндерді өңдеуге арналған негізгі құрылым болып табылады.

Жолдың құрылымы және индекстеу

Жолдағы таңбаларға олардың индекстері арқылы қол жеткізуге болады. Индекс 0-ден басталады. Әрбір таңба нөмірленеді, және индексті пайдалану арқылы белгілі бір таңбаны алуға болады. Индекстерді дұрыс қолдану жолды талдау және түрлендіру кезінде маңызды рөл атқарады. Егер индексті жолдың ұзындығынан тыс пайдалансаңыз, қате пайда болады.

Жолдың ұзындығы

Жолдың ұзындығын анықтау үшін len() функциясы қолданылады. Бұл функция жолдағы барлық таңбалар санын есептеп, бүтін сан мәнін қайтарады. Бұл функция жолдағы барлық таңбалардың жалпы санын қайтарады, соның ішінде бос орындар да есептеледі.

Жолдарды кесу

Ережелер!!!

- Бастапқы индекс кесудің қай жерден басталатынын көрсетеді.
- Соңғы индекс таңдалған диапазонға кірмейді.
- Егер бастапқы немесе соңғы индекс көрсетілмесе, сәйкесінше жолдың басынан немесе соңынан бастап кесінді алынады.

Жолды кесу (substring) үшін қос нүкте (:) операторын қолдануға болады. Бұл әдіс жолдың үзінділерін алуға мүмкіндік береді. Кесу синтаксисі: жол[бастапқы индекс:соңғы индекс]. Соңғы индекс көрсетілген таңбаны қамтымайды. Егер бастапқы немесе соңғы индекс көрсетілмесе, жолдың басынан немесе соңынан бастап үзінді алынады.

Кесу әдісі жолдың үзінділерін алуға және олармен жұмыс жасауға мүмкіндік береді.

Жолдарды біріктіру

Жолдарды біріктіру үшін + операторы қолданылады. Бұл оператор жолдарды бір бүтін жолға біріктіру мүмкіндігін береді. Сонымен қатар, жолдарға бос орын немесе басқа таңбалар қосу үшін арасына тиісті таңбалар енгізуге болады.

Жол элементтерін қайталау және санау

Жолдағы белгілі бір таңбаның немесе ішкі жолдың қанша рет кездесетінін анықтау үшін циклдар және есептегіштер қолданылады. Циклдардың көмегімен жол элементтерін ретімен тексеріп, қажетті шартты орындауға болады. Сонымен қатар, Python-да жолдарды өңдеу үшін арнайы дайын функциялар бар.

Логикалық операциялар және in операторы

in операторы бір жолдың ішінде басқа жолдың бар-жоғын тексеруге мүмкіндік береді. Бұл оператор шартты тексерулерде жиі қолданылады. Егер ішкі жол негізгі жолда кездессе, нәтиже True, ал егер кездеспесе, False мәнін қайтарады. Бұл мүмкіндік мәтіндік деректерді талдау және сүзу кезінде пайдалы.

Жолдарды өңдеу функциялары

Функциялар	**Сипаттамасы**
lower()	жолдағы барлық әріптерді кіші регистрге түрлендіреді
upper()	жолдағы барлық әріптерді бас регистрге түрлендіреді
find()	жолдың ішінде ішкі жолды іздейді және оның алғашқы кездесу индексін қайтарады
replace()	жолдың бір бөлігін басқасымен ауыстырады
strip()	жолдың басындағы және соңындағы бос орындарды алып тастайды
split()	жолды берілген бөлгіш бойынша бөліктерге бөледі
join()	тізім элементтерін бір жолға біріктіреді

Жолдарды іздеу және ауыстыру

Жолдың ішінде белгілі бір үлгіні табу немесе оны басқа үлгімен ауыстыру үшін find() және replace()функциялары қолданылады. Бұл функциялар мәтіндік деректерді өңдеу кезінде жиі пайдаланылады.

Префикстер мен суффикстерді тексеру

Жолдың белгілі бір таңбадан басталатынын немесе аяқталатынын тексеру үшін startswith() және endswith()функциялары қолданылады. Бұл функциялар арқылы мәтіннің құрылымын талдау оңай болады.

Python тіліндегі жолдар – мәтіндік деректермен жұмыс істеудің негізгі құралы. Жолдармен тиімді жұмыс істеу үшін олардың құрылымын, индекстерін, ұзындығын анықтау және оларды өңдеу функцияларын

меңгеру қажет. Жолдардың икемділігі және олармен жұмыс істеуге арналған құралдардың көптігі Python тілін мәтіндік деректерді талдау үшін қолайлы етеді.

Сабақ 2.3.3. Сөздіктер

Сөздік (dictionary) – Python бағдарламалау тілінде ең жиі қолданылатын деректер құрылымдарының бірі. Сөздіктер "кілт:мән" жұптары арқылы ұйымдастырылған, бұл оларды реттелмеген, бірақ құрылымды ақпаратты сақтау үшін өте қолайлы етеді. Әрбір кілт бірегей болуы керек, ал мәндер кез келген деректер типінде болуы мүмкін: сандар, мәтіндер, тізімдер, немесе тіпті басқа сөздіктер.

Сөздік – бұл "сөмке" тәрізді коллекция, мұнда элементтер реттілігі маңызды емес, ал кілттер элементтерге қол жеткізудің негізгі механизмі болып табылады.

Мысалы, адамның аты-жөні мен жасын сақтау үшін:

python
адам = {'аты': 'Алия', 'жасы': 25}

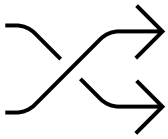
Сөздіктер көбінесе деректер базаларының шағын аналогы ретінде пайдаланылады, себебі олар әр түрлі деректерді бір құрылымда сақтауға мүмкіндік береді.

Сөздіктің ерекшеліктері

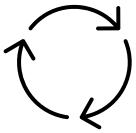
Сөздік құрылымы Python-ның икемді және қуатты мүмкіндіктерінің бірі

болып табылады. Олардың басты ерекшеліктері:

Кілттер мен мәндер: Әрбір элемент кілттен (уникалды идентификатор) және оған сәйкес мәннен тұрады. Кілттер ретінде тек өзгермейтін типтерді (мысалы, жолдар, сандар, кортеждер) қолдануға болады.

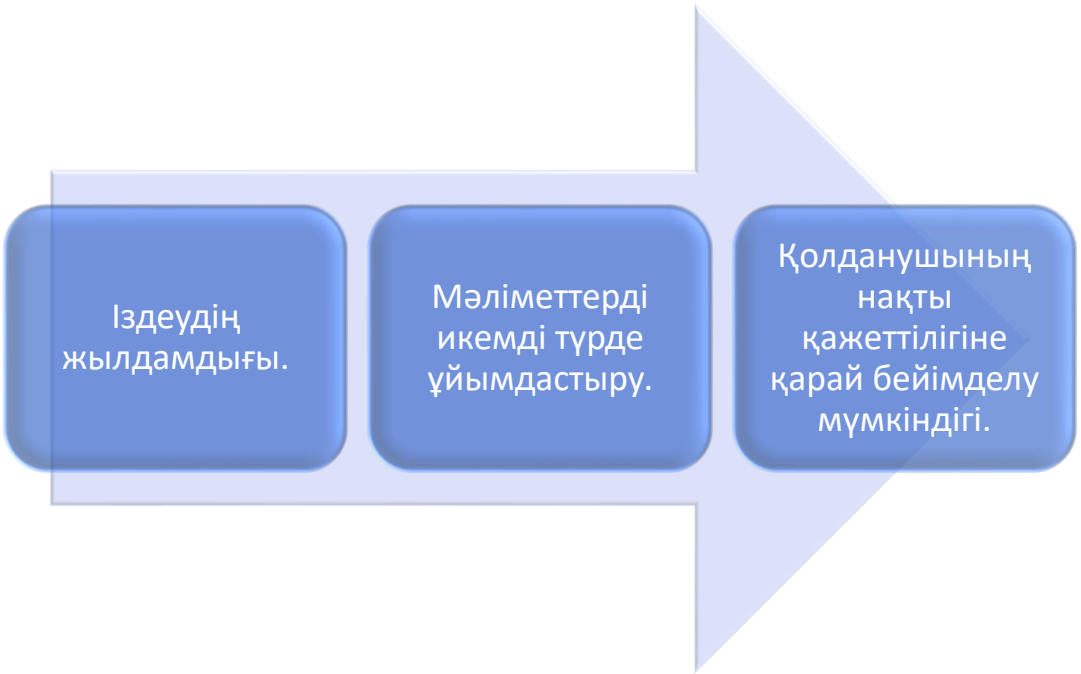


Реттелмеуі: Python 3.7-нұсқасынан бастап сөздіктерде кілттердің реті сақталады. Бірақ, бастапқыда сөздіктер реттіліксіз деректерді сақтауға арналған.



Өзгертілуі: Сөздіктерге жаңа элементтерді қосу,бұрынғыларын өзгерту немесе алып тастауға болады. Бұл оларды динамикалық деректермен жұмыс істеу үшін таптырмас құралға айналдырады.

Сөздік қолдану артықшылықтары:



Сөздіктерді түсіну және қолдану – Python-ды меңгерудегі маңызды қадамдардың бірі.

Сөздікті қолдану

Python сөздіктері көптеген пайдалы сценарийлерде қолданылады: мәліметтерді сақтау, өңдеу, сүзу және іздеу. Сөздік элементтеріне қол жеткізу үшін кілттер пайдаланылады.

Сөздікті жасау және элементтерді өңдеу тәсілдері:

1. **Сөздікті құру**:
    - Бос сөздік жасау: сөздік = {} немесе сөздік = dict().
    - Кілттер мен мәндер арқылы сөздік жасау:

python
студент = {'аты': 'Мақсат', 'жасы': 20, 'курс': 3}

2. **Элементтерді қосу және өзгерту**:
   - Жаңа кілт қосу: сөздік['телефон'] = '123456789'.
   - Қолданыстағы кілттің мәнін өзгерту: сөздік['жасы'] = 21.
3. **Элементтерді жою**:
   - Белгілі бір элементті жою: del сөздік['жасы'].
   - Барлық элементтерді жою: сөздік.clear().

Қателерді болдырмау:

- Егер сөздікте жоқ кілтке сілтеме жасалса, KeyError қатесі туындайды. Бұл қатені болдырмау үшін:
  - in операторы: if 'аты' in сөздік.
  - get() әдісі: мән = сөздік.get('аты', 'Мәлімет жоқ').

Сөздіктер әр түрлі салаларда – мәліметтерді талдау, файлдардан алынған ақпаратты сақтау және өңдеу, API жауаптарын басқару сияқты – кеңінен қолданылады.

Сөздік литералдары

Сөздіктерді құру үшін Python-да фигуралы жақшалар {} пайдаланылады. Сөздік ішінде кілттер мен мәндер қос нүкте (:) арқылы бөлініп, әрбір жұп үтір арқылы ажыратылады.

Сөздік жасау үлгілері:

1. Бос сөздік:

   python
   бос_сөздік = {}

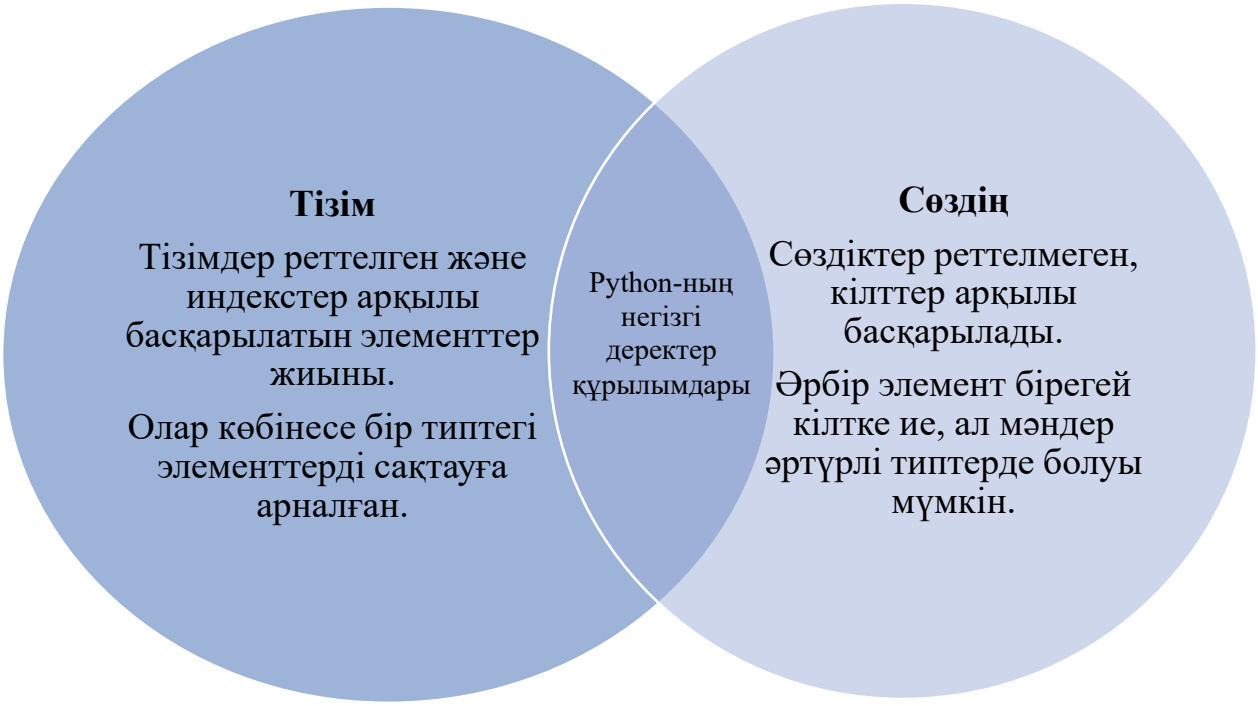
2. Алдын ала анықталған мәндері бар сөздік:

   python
   студент = {'аты': 'Алия', 'жасы': 22, 'курс': 4}

Қасиеттері:

- Кілттер қайталанбайды. Егер бірдей кілт бірнеше рет қолданылса, соңғы мәні сақталады.
- Сөздіктер икемділігімен ерекшеленеді және әртүрлі деректерді сақтауға мүмкіндік береді.

Сөздік литералдары Python бағдарламалауында деректер құрылымдарын жеңіл құруға және оларды нақты қажеттіліктерге бейімдеуге көмектеседі.



Мысалы:

- Тізімде: студенттер = ['Алия', 'Мақсат', 'Ербол'].
- Сөздікте: студент = {'аты': 'Алия', 'жасы': 22, 'курс': 4}.

Сөздіктер мен тізімдерді тиімді қолдану мәліметтерді сақтау және өңдеу кезіндегі қажеттіліктерге байланысты таңдалады.

Сөздік әдістері

Python сөздіктермен жұмыс істеуді жеңілдету үшін бірнеше пайдалы әдістерді ұсынады:

keys()	Сөздіктегі барлық кілттерді қайтарады.
values()	Сөздіктегі барлық мәндерді қайтарады.
items()	Кілттер мен мәндер жұбын қайтарады.

get()	Кілтке сәйкес мәнді қайтарады, егер кілт жоқ болса, әдепкі мәнді береді.
update()	Басқа сөздік элементтерін ағымдағы сөздікке қосады.

Бұл әдістер сөздіктерді басқару, деректерді алу және өңдеу процесін айтарлықтай жеңілдетеді.

Сөздіктер Python-да әртүрлі мәселелерді шешу үшін қолданылады:

1. **Санау**:Мәтін ішіндегі әріптер, сөздер немесе басқа элементтердің жиілігін анықтау.
2. **Іздеу**:Сөздікте кілттің бар-жоғын тексеру.
3. **Деректерді сақтау**:Бірегей идентификаторлармен (кілттермен) байланыстырылған мәліметтерді сақтау.

Сөздіктерді тиімді пайдалану сіздің бағдарламаларыңыздың функционалдығын арттырады және деректерді өңдеуді жеңілдетеді.

Python сөздіктері – бұл құрылымды ақпаратты сақтау және өңдеудің тамаша құралы. Оларды тиімді меңгеру үшін олардың негізгі қасиеттерін, қолдану ережелерін және әдістерін түсіну маңызды. Сөздіктер арқылы бағдарламаларда мәліметтерді басқару жылдам әрі жеңіл болады.

Сабақ 2.3.4. Кортеждер

Кортеж (tuple) – Python бағдарламалау тілінде қолданылатын деректердің реттелген, өзгермейтін (immutable) құрылымы. Кортеждер тізімдерге ұқсас, бірақ негізгі айырмашылығы – оларды өзгертудің мүмкін еместігінде. Кортеж элементтері бір рет анықталған соң, оны қосу, жою немесе өзгерту мүмкін болмайды. Бұл қасиет кортеждерді тұрақты деректерді сақтауға және қорғауға өте ыңғайлы етеді.

Кортеждер дөңгелек жақшалармен (()) белгіленеді. Олар сандарды, жолдарды, басқа кортеждерді және басқа деректер құрылымдарын қамти алады.

Кортеждердің ерекшеліктері

Кортеждер – тізімдердің баламасы ретінде қолданылатын құрылым. Олардың басты ерекшеліктері:

Кортеждер тұрақты деректермен жұмыс істеу қажет болғанда немесе функцияларға тұрақты параметрлер беру үшін жиі қолданылады.

Кортеждерді құру және қолдану

1. **Дөңгелек жақшалар арқылы**:

   python
   кортеж = (1, 2, 3, 4)

2. **Кортежді анықтамай-ақ**:

   Python
   кортеж = 1, 2, 3, 4

   Бұл жағдайда Python кортежді автоматты түрде таниды.

3. **Тізімді кортежге түрлендіру**:

   python
   тізім = [1, 2, 3]
   кортеж = tuple(тізім)

Кортеждердегі элементтерге индекс арқылы қол жеткізуге болады, бірақ оларды өзгерту мүмкін емес:

```
python
кортеж = (10, 20, 30)
мән = кортеж[1]  # 20
```

Кортеждердің өзгермейтін қасиеті олардың сенімділігі мен қауіпсіздігін арттырады. Бұл қасиет, әсіресе, деректерді өзгертуден қорғау қажет болғанда маңызды.

Кортеждер мен тізімдердің айырмашылығы

Кортеждер тізімдерге ұқсас болғанымен, олардың арасында бірнеше маңызды айырмашылықтар бар:

1. **Өзгерту мүмкіндігі**:
    - Тізімдер өзгермелі (mutable), яғни элементтерін қосуға, жоюға немесе өзгертуге болады.
    - Кортеждер өзгермейтін (immutable), оларды тек оқуға болады.
2. **Қолданылуы**:
    - Тізімдер деректердің жиі өзгеруін талап ететін жағдайларда қолданылады.
    - Кортеждер деректерді қорғау және тұрақты ақпаратты сақтау үшін пайдаланылады.
3. **Өнімділік**:
    - Кортеждер тізімдерге қарағанда жылдамырақ, себебі олар өзгермейді және жадты тиімді пайдаланады.

Бұл айырмашылықтар тізімдер мен кортеждерді қолдану аясын айқындап береді.

Кортеждермен жұмыс

1. **Индекстеу**: Кортеждің әрбір элементі индекс арқылы қолжетімді:

    ```
    кортеж = ('a', 'b', 'c')
    элемент = кортеж[0]  # 'a'
    ```

2. **Цикл арқылы қайталау**: Кортеж элементтерін for циклы арқылы қайталап қарауға болады:

    ```
     кортеж = (1, 2, 3)
    for элемент in кортеж:
        print(элемент)
    ```

3. **Кортеждерді салыстыру**: Кортеждер салыстыру операторларымен (<, >, ==) салыстырылады. Python элементтерді рет-ретімен салыстырады және нәтижені қайтарады.
4. **Методтар**: Кортеждерде екі негізгі әдіс бар:
    - **count(элемент)** – берілген элементтің кортежде қанша рет кездесетінін анықтайды.
    - **index(элемент)** – берілген элементтің бірінші кездесу индексін қайтарады.

Кортеждер көптеген практикалық жағдайларда пайдалы:

1. **Тұрақты параметрлер беру**: Кортеждер өзгермейтін болғандықтан, функцияларға параметр ретінде берілген деректерді қорғау үшін қолданылады.

   ```
   def деректер_шығару(деректер):
       for элемент in деректер:
           print(элемент)
   ```

2. **Деректерді ұйымдастыру**: Кортеждер бірнеше мәнді бір уақытта қайтару үшін қолданылады:

   ```
   def координаттар():
       return (10, 20)
   x, y = координаттар()
   ```

3. **Сұрыптау және деректерді өңдеу**: Кортеждер тізімдермен бірге мәндерді реттеу немесе сүзу үшін қолданылады.

Кортеждерді сұрыптау

Кортеждер өздері сұрыпталмайды, бірақ тізімге түрлендіріп, сұрыптауды орындауға болады:

python
```
кортеж = (3, 1, 2)
сұрыпталған = tuple(sorted(кортеж))
```

Сонымен қатар, кортеждер сөздіктердің элементтерін сұрыптау кезінде де қолданылады:

python
```
сөздік = {'a': 3, 'b': 1, 'c': 2}
тізім = sorted(сөздік.items())  # [('a', 3), ('b', 1), ('c', 2)]
```

Кортеждер Python-ның маңызды деректер құрылымдарының бірі болып табылады. Олардың өзгермейтін қасиеті тұрақты деректерді қорғауды, ал тиімділігі үлкен көлемдегі мәліметтермен жұмыс жасауды жеңілдетеді. Кортеждерді қолдану арқылы Python бағдарламаларының өнімділігі мен қауіпсіздігін арттыруға болады.