Исманова Тынара Кубанычбековна
Абдразакова Гүлмира Абдужапаровна

ОМЗ: Объектке багытталган программалоо тилдерин колдонуу менен программалык кодду жазуу, маалыматтарды аныктоо жана манипуляциялоо

(ОКУУ УСУЛДУК КОЛДОНМО)

Манас 2026

КЫРГЫЗ РЕСПУБЛИКАСЫ ИЛИМ, ЖОГОРКУ БИЛИМ БЕРҮҮ ЖАНА ИННОВАЦИЯЛАР МИНИСТРЛИГИ

“Б.ОСМОНОВ АТЫНДАГЫ ЖАЛАЛ-АБАД МАМЛЕКЕТТИК УНИВЕРСИТЕТИ” ИЛИМИЙ БИЛИМ БЕРҮҮ ӨНДҮРҮШТҮК КОМПЛЕКСИ

АГРАРДЫК ТЕХНИКАЛЫК ИНСТИТУТУ

ЖАЛАЛ-АБАД КОЛЛЕДЖИ

Автоматташтырылган системалар жана математика ПЦКсы

Исманова Тынара Кубанычбековна
Абдразакова Гүлмира Абдужапаровна

ОМ3: Объектке багытталган программалоо тилдерин колдонуу менен программалык кодду жазуу, маалыматтарды аныктоо жана манипуляциялоо

Орто кесиптик билим берүү мекемелеринин мугалимдери жана студенттери үчүн арналган окуу усулдук колдонмо. Колдонмодо объектке багытталган программалоонун негизги түшүнүктөрүн камтыйт.

Манас 2026-ж.

УДК 004.438.5
ББК 32.973.26
Т51

Т 51 Басууга Б.Осмонов атындагы Жалал-Абад мамлекеттик университетинин методикалык кеңеши сунуш кылган.

Рецензенттер: Б.Осмонов атындагы ЖАМУ ЖАКтын АСжМ ПЦКсынын окутуучусу Маманова Б.А
Осмонов атындагы ЖАМУ ЖАКтын АСжМ ПЦКсынын окутуучусу Мусаева Э.Т. Б
ЖАКтын методикалык иштери боюнча инспектору Сатимбаева А.К.

Исманова Т.К. **Т 51.** ОМЗ: Объектке багытталган программалоо тилдерин колдонуу менен программалык кодду жазуу, маалыматтарды аныктоо жана манипуляциялоо.(Объектке багытталган программалоо) Окуу усулдук колдонмо. – Манас: 2025. – 48 б.

ISBN_____

Бул окуу-методикалык колдонмо «Объектке багытталган программалоо» дисциплинасы боюнча орто кесиптик билим берүү мекемелеринин студенттери үчүн арналган. Колдонмодо объектке багытталган программалоонун негизги түшүнүктөрү, принциптери жана Python программалоо тилиндеги практикалык мисалдар камтылган. материал практикалык тапшырмалар жана тесттер менен бекемделет.

УДК 004.438.5
ББК 32.973.26
ЖАМУ, 2025

Мазмуну

Киришүү

Технологиялардын тез өнүгүүсү жана программалык системалардын татаалдашуусу доорунда программалык камсыздоону иштеп чыгуунун салттуу ыкмалары көбүнчө натыйжасыз болуп калат. Объектке багытталган программалоо (ОБП) татаал, масштабдуу жана оңой тейлөөгө мүмкүн болгон колдонмолорду иштеп чыгуунун негизги куралы болуп калган заманбап жана күчтүү парадигма болуп саналат.

Объектке багытталган программалоо – бул синтаксистик эрежелердин жыйындысы эмес, кодду уюштуруунун принципиалдуу жаңы ыкмасын сунуш кылган методология. Процедуралык ыкмадагыдай нускамалардын ырааттуулугуна басым жасоонун ордуна, объектке багытталган программалоо бири-бири менен өз ара аракеттенген объекттердин жыйындысы катары курат. Бул объекттер реалдуу дүйнөдөн алынган объекттерди моделдештирет, маалыматтарды (абалдарды) жана функцияларды (жүрүм-турумдарды) класс деп аталган бир бүтүнгө бириктирет.

Максаты жана мааниси

Объектке багытталган программалоо изилдөөнүн негизги максаты – сапаттын заманбап стандарттарына жооп берген программалык продуктуларды долбоорлоо жана түзүү үчүн зарыл болгон теориялык билимдерди жана практикалык көндүмдөрдү алуу. сыяктуу объектке багытталган программалоо принциптерин киргизүү иштеп чыгуучуларга төмөнкүлөргө мүмкүндүк берет:

- Коддун модулдуулугун жана кайра колдонулушун жогорулатуу.
- Ири масштабдуу системаларды оңдоону жана тейлөөнү жөнөкөйлөтүү.
- Жамааттык долбоорлордун үстүндө иштөөдө татаалдыкты азайтуу.

Объектке багытталган программалоо (ОБП) принциптерин билүү Java, C++, Python, C# жана Swift сыяктуу популярдуу тилдер менен иштеген көпчүлүк заманбап программалык камсыздоону иштеп чыгуучулар үчүн фундаменталдуу талап болуп саналат.

«Объектке багытталган программалоо» окуу дисциплинасы ОМ3 «Объектке багытталган программалоо тилдерин колдонуу менен программалык кодду жазуу, маалыматтарды аныктоо жана манипуляциялоо» профессионалдык циклинин базалык дисциплинасы болуп саналат жана адистик боюнча бүтүрүүчүлөрдүн даярдык деңгээлине жана мазмунунун минимумуна мамлекеттик талаптарды ишке ашырууга арналган. 230109 «Автоматташтырылган системаларды жана эсептөө техникаларын программалык камсыздоо» орто кесиптик билим берүү жана бардык окуу формалары үчүн бирдиктүү болуп саналат.

1. **Тема. Программалоо тилдерине киришүү**

План:

1. Программалоо түшүнүгү
2. Негизги терминдер жана түшүнүктөр
3. Маалымат типтери жана маалымат структуралары программалоонун негизги парадигмалары

Максаты: Катышуучуларды программалоонун негиздери менен тааныштыруу, анын ичинде негизги терминдер жана түшүнүктөр, маалымат типтери, маалымат структуралары жана программалоонун негизги парадигмалары, алар бул билимдерди практикалык маселелерди чечүү жана программалык камсыздоону иштеп чыгуу үчүн ишенимдүү колдоно алышат.

1. Программалоо деген эмне жана анын азыркы дүйнөдөгү ролун түшүндүрүү.
2. Алгоритмдер, өзгөрмөлөр, функциялар, циклдер жана шарттар сыяктуу негизги түшүнүктөрдү изилдөө.
3. Программалоодогу маалыматтардын примитивдүү типтери жана аларды колдонуу менен тааныштыруу.
4. Массивдер, тизмелер, стектер, кезектер жана сөздүктөр аркылуу маалыматтарды кантип уюштуруу жана сактоону түшүнүү.
5. Катышуучуларга практикалык маселелерди чечүү жана жөнөкөй программаларды жазуу үчүн алынган билимдерин колдонууга үйрөтүү.

Программалоо — бул компьютерде белгилүү тапшырмаларды аткаруучу программаларды түзүү процесси. Программалар ар кандай программалоо тилдеринде жазылат, алардын ар биринин өзүнүн өзгөчөлүктөрү жана синтаксиси бар. Программалоо тапшырмаларды автоматташтырууга, маалыматтарды талдоого жана жашообузду жеңилдеткен тиркемелерди түзүүгө мүмкүндүк берет. Заманбап дүйнөдө программалоо илим, медицина, каржы жана көңүл ачууну кошкондо, көптөгөн тармактардын ажырагыс бөлүгү болуп калды. Бул чоң көлөмдөг ү маалыматтарды иштетүүгө жана кол менен мүмкүн болбогон тапшырмаларды аткарууга мүмкүндүк берүүчү татаал системаларды түзүүгө мүмкүндүк берет.

Программалоо ошондой эле жасалма интеллект, машиналык окутуу жана нерселердин интернети сыяктуу технологияларды өнүктүрүүдө негизги ролду ойнойт. Бул технологиялар биздин дүйнөнү өзгөртүп, аны акылдуу жана байланышкан кылат. Мисалы, программалоо жарыкты жана температураны автоматтык түрдө жөнгө салуучу акылдуу үйлөрдү же айдоочунун катышуусуз жүргүнчүлөрдү коопсуз ташууга жөндөмдүү автопилоттуу унааларды түзүүгө мүмкүндүк берет.

Негизги терминдер жана түшүнүктөр

Алгоритм

Алгоритм — бул белгилүү бир маселени чечүү үчүн аткарылышы керек болгон кадамдардын ырааттуулугу. Алгоритмдер жөнөкөй (мисалы, сандардын тизмесин иреттөө алгоритми) же татаал (мисалы, машиналык окутуу алгоритмдери) болушу мүмкүн. Алгоритмдер программалоонун негизи болуп саналат, анткени алар программанын өз милдеттерин кантип аткарарын аныктайт. Жакшы ишталип чыккан алгоритм программанын өндүрүмдүүлүгүн жана эффективдүүлүгүн бир топ жакшыртат.

Алгоритмдердин мисалдарына издөө алгоритмдери кирет, мисалы, иреттелген массивдеги элементти тез табууга мүмкүндүк берген бинардык издөө жана тез иреттөө жана

бириктирүү иреттөө сыяктуу иреттөө алгоритмдери. Ар кандай экенин түшүнүү маанилүү алгоритмдер ар кандай татаалдыкка жана өндүрүмдүүлүккө ээ болушу мүмкүн жана туура алгоритмди тандоо программанын иштешине олуттуу таасир этиши мүмкүн.

Өзгөрмө

Өзгөрмө — бул маалыматтарды сактоо үчүн колдонулган аталган эс тутумунун аймагы. Өзгөрмөлөр программанын аткарылышы процессинде маанисин өзгөртө алат. Мисалы, age өзгөрмөсү колдонуучунун жашын сактай алат. Өзгөрмөлөр программадагы маалыматтарды сактоонун жана иштетүүнүн негизги жолу болуп саналат. Алар сандар, саптар жана логикалык маанилер сыяктуу маалыматтардын ар кандай түрлөрүн сактай алат. Өзгөрмөлөр жергиликтүү болушу мүмкүн, башкача айтканда, белгилүү бир функциянын же код блогунун ичинде гана жеткиликтүү, же глобалдык, башкача айтканда, бүт программада жеткиликтүү. Өзгөрмөлөрдү туура колдонуу кодду окумдуу жана башкарууга оңой кылууга жардам берет. Мисалы, totalPrice же userName сыяктуу мааниге ээ өзгөрмө аттарын колдонуу кодду башка иштеп чыгуучулар үчүн түшүнүктүү кылат.

Функция

Функция — бул белгилүү бир тапшырманы аткаруучу жана программанын башка бөлүктөрүнөн чакырыла турган код блогу. Функциялар кодду структуралаштырууга жардам берет жана аны окумдуу жана кайра колдонууга мүмкүндүк берет. Мисалы, calculateSum(a, b) функциясы эки сандын суммасын кайтара алат. Функциялар параметрлерди кабыл алып, маанилерди кайтара алат, бул аларды программалоодо абдан ийкемдүү жана күчтүү куралдарга айлантат.

Функциялар ошондой эле кодду кайталоодон качууга жардам берет, анткени бир эле код блогун программанын ар кайсы бөлүктөрүндө колдонсо болот. Бул программаны кыска жана колдоого оңой кылат. Мындан тышкары, функциялар рекурсивдүү болушу мүмкүн, башкача айтканда, өздөрүн чакырып, факториалдарды эсептөө же даректарды кыдыруу сыяктуу татаал маселелерди чечүүгө мүмкүндүк берет.

Цикл

Цикл — бул код блогун бир нече жолу кайталоого мүмкүндүк берген конструкция. Циклдердин негизги түрлөрүнө for, while жана do-while кирет. Циклдер массивдерди иштетүү, кайталануучу тапшырмаларды аткаруу жана башкалар үчүн колдонулат. Циклдер программалоодо маанилүү курал болуп саналат, анткени алар кайталануучу тапшырмаларды автоматташтырууга жана чоң көлөмдөгү маалыматтарды иштетүүгө мүмкүндүк берет.

Мисалы, for цикли массивдин бардык элементтерин кайталоо жана ар бир элемент менен белгилүү бир аракеттерди аткаруу үчүн колдонулушу мүмкүн. while цикли белгилүү бир шарт аткарылганга чейин код блогун аткарууга мүмкүндүк берет. do-

while цикли while циклине окшош, бирок код блогу жок дегенде бир жолу аткарылышына кепилдик берет.

Шарт

Шарт – бул белгилүү бир шарт аткарылса гана код блогун аткарууга мүмкүндүк берген конструкция. Негизги шарттуу операторлорго if, else if жана else кирет. Шарттар программада чечим кабыл алууга жардам берет. Шарттуу операторлор программага ар кандай кырдаалдарга жооп берүүгө жана киргизүү дайындарына же абалына жараша өз жүрүм-турумун өзгөртүүгө мүмкүндүк берет.

Мисалы, if шарты сандын оң экенин текшерүү жана натыйжага жараша белгилүү бир аракеттерди аткаруу үчүн колдонулушу мүмкүн. else if шарты удаалаш бир нече шарттарды текшерүүгө мүмкүндүк берет, ал эми else шарты мурунку шарттардын бири да аткарылбаса аткарылат. Шарттар дагы татаал логикалык конструкцияларды түзүүгө мүмкүндүк берет.

Маалымат түрлөрү жана маалымат структуралары

Примитивдүү маалымат түрлөрү – бул программалоо тилдери тарабынан колдоого алынган негизги маалымат түрлөрү. Аларга төмөнкүлөр кирет:

Бүтүн сандар (int): бөлчөк бөлүгү жок сандар, мисалы, 1, 42, -7. Бүтүн сандар массивдеги элементтердин саны же колдонуучунун жашы сыяктуу сандык маанилерди сактоо үчүн колдонулат.

Чыныгы сандар (float, double): бөлчөк бөлүгү бар сандар, мисалы, 3.14, -0.001. Чыныгы сандар калкып жүрүүчү чекит маанилерин, мисалы, эсептөөлөрдүн натыйжаларын же координаттарды сактоо үчүн колдонулат.

Белгилер (char): жалгыз белгилер, мисалы, 'a', 'Z', '1'. Белгилер жеке тамгаларды, сандарды же атайын белгилерди сактоо үчүн колдонулат.

Логикалык маанилер (bool): чыныгы (true) же жалган (false) болушу мүмкүн болгон маанилер. Логикалык маанилер логикалык туюнтмалардын жана шарттардын натыйжаларын сактоо үчүн колдонулат.

Примитивдүү маалымат түрлөрү татаал маалымат структуралары үчүн негиз болуп саналат жана бардык программалоо тилдеринде колдонулат. Алар маалыматтарды сактоо жана иштетүү үчүн негизги мүмкүнчүлүктөрдү камсыз кылат.

Маалымат структуралары

Маалымат структуралары – бул натыйжалуу пайдалануу үчүн маалыматтарды уюштуруу жана сактоо жолдору. Негизги маалымат структуралары төмөнкүлөрдү камтыйт:

Массивдер: индекс боюнча жеткиликтүү болгон бир типтеги элементтердин жыйнагы. Массивдер сандардын же саптардын тизмеси сыяктуу чоң көлөмдөгү маалыматтарды сактоого жана иштетүүгө мүмкүндүк берет.

Тизмелер: өлчөмүн өзгөртө ала турган элементтердин динамикалык жыйнактары. Тизмелер ийкемдүүлүктү жана маалыматтар менен иштөөдө ыңгайлуулукту камсыз кылат, анткени алар зарылчылыкка жараша элементтерди кошууга жана жок кылууга мүмкүндүк берет.

Стектер: "акыркы келген – биринчи кеткен" (LIFO) принциби боюнча иштеген маалымат структуралары. Стектер функция чакыруулары же калькулятордогу операциялар сыяктуу убактылуу маалыматтарды сактоо үчүн колдонулат.

Кезектер: "биринчи келген – биринчи кеткен" (FIFO) принциби боюнча иштеген маалымат структуралары. Кезектер басып чыгаруу кезегиндеги тапшырмалар же чаттагы билдирүүлөр сыяктуу келип түшкөн тартипте иштетилиши керек болгон маалыматтарды сактоо үчүн колдонулат.

Сөздүктөр (хеш-таблицалар): ачкыч боюнча маанилерди тез табууга мүмкүндүк берген "ачкыч-маани" жуптарынын жыйнагы. Сөздүктөр телефон китептери же конфигурация параметрлери сыяктуу ассоциативдик маалыматтарды сактоо үчүн колдонулат.

Маалымат структуралары программалоодо негизги ролду ойнойт, анткени алар маалыматтарды натыйжалуу уюштурууга жана иштетүүгө мүмкүндүк берет. Маалымат структурасын туура тандоо программанын өндүрүмдүүлүгүн жана колдонуунун ыңгайлуулугун бир топ жакшыртат.

Бышыктоочу суроолор

1. Программалоо деген эмне?
2. Алгоритм деген эмне?
3. Өзгөрмө деген эмне жана ал эмне үчүн колдонулат?
4. Функциянын ролу эмнеде?
5. Цикл жана шарт деген эмне? Кандай учурларда колдонулат?

Тест

1. Программалоо деген эмне?
а) Оюндарды түзүү процесси
б) Компьютерде белгилүү бир тапшырмаларды аткарган программаларды түзүү процесси
в) Илимий макалаларды жазуу процесси
г) Программалык камсыздоону орнотуу процесси

2. Алгоритм деген эмне?
а) Программалоо тилинде жазылган программа
б) Конкреттүү маселени чечүү үчүн кадамдардын ырааттуулугу
в) Маалыматтарды сактоо ыкмасы г) Кодду оңдоо куралы

3. Төмөнкү аныктамалардын кайсынысы отнөзгөрмөгө тиешелүү?
а) Маалыматтарды сактоо үчүн аталган эс тутум аймагы
б) Белгилүү бир тапшырманы аткарган код блогу
в) Бир нече жолу аткарылуучу цикл
г) Туюнтманын аныктыгын текшерүүчү шарт

4. Программалоодогу функция эмнедеген эмне?

а) Кайталануучу тапшырмаларды аткаруу үчүн конструкция
б) Белгилүү бир тапшырманы аткаруучу жана программанын башка бөлүктөрүнөн чакырыла турган код блогу
в) Маалыматтарды сактоо ыкмасы г) Өзгөртүүгө мүмкүн болбогон алгоритм

5. Төмөнкү циклдердин кайсынысы код блогун шарт чын болгонго чейин аткарууга мүмкүндүк берет?
а) үчүн б) while в) do-while г) foreach

6. Төмөнкү маалымат түрлөрүнүн кайсынысы логикалык маанини көрсөтөт?
а) int б) float в) char г) bool

7. Массив маалымат структураларынын контекстинде эмне?
а) Индекс боюнча жеткиликтүү болгон бир типтеги элементтердин жыйнагы
б) Элементтердин динамикалык жыйнагы
в) "Биринчи келген - биринчи кеткен" принциби боюнча иштеген маалымат структурасы
г) "Ачкыч-маани" жуптарынын жыйнагы

8. Программалоодо функцияларды колдонуунун негизги артыкчылыктары эмнеде?
а) Кодду жөнөкөйлөтүү жана анын көлөмүн азайтуу
б) Коддун татаалдыгын жогорулатуу
в) Маалымат түрүн өзгөртүү
г) Өзгөрмөлөр менен иштөөнү жөнөкөйлөтүү

9. Программалоодогу шарт деген эмне?
а) Код блогун шарт аткарылганда гана аткарууга мүмкүндүк берүүчү конструкция
б) Маанилерди сактоо үчүн маалымат структурасы
в) Тапшырманы аткаруучу алгоритм г) Маалыматтарды сактоочу өзгөрмө

10.Төмөнкү маалымат структураларынын кайсынысы "акыркы келген - биринчи кеткен" (LIFO) принциби боюнча иштейт?
а) Кезек б) Массив в) Тизме г) Стек
Жооптор:
1. б, 2. б, 3. а, 4. б, 5. б, 6. г, 7. а, 8. а, 9. а, 10. Г

2. Тема: Класс жана объекттер

План:

1. Конструкторлор
2. Объекттин атрибуттары
3. Класс методдору
4. Деструкторлор

Максат: Python тилинде конструкторлорду, атрибуттарды, методдорду жана деструкторлорду колдонуп өзүңүздүн класстарыңызды ишенимдүү түзүүнү үйрөнүү, ошондой эле объекттер жана алардын жашоо цикли кандайча иштээрин түшүнүү.

Python тилинде көптөгөн орнотулган типтер бар, мисалы, int, str жана башкалар, аларды программада колдонсок болот. Бирок Python тили өз алдынча типтерди аныктоого да мүмкүндүк берет. **Класстардын жардамы менен. Класс кандайдыр бир маанини билдирет. Класстын конкреттүү чагылдырылышы объект болуп саналат.**

Дагы бир аналогия жүргүзүүгө болот. Ар бирибизде адам жөнүндө түшүнүк бар, анын аты, жашы жана башка мүнөздөмөлөрү бар. Адам кээ бир аракеттерди аткара алат - басуу, чуркоо, ойлонуу ж.б. Башкача айтканда, **мүнөздөмөлөрдүн жана аракеттердин жыйындысын камтыган бул түшүнүктү класс деп атоого болот**. Бул шаблондун конкреттүү чагылдырылышы ар кандай болушу мүмкүн, мисалы, кээ бир адамдардын бир аты болсо, башкаларынын аты башкача болот. Жана реалдуу жашаган адам бул класстын объектисин көрсөтөт.

Python тилинде класс **class**:

```
class класстын_аты:
    класс_атрибуттары
    класс_методдору
```

Класстын ичинде анын ар кандай мүнөздөмөлөрүн сактаган атрибуттары жана методдору - класс функциялары аныкталат.

Эң жөнөкөй классты түзөлү:

```
class Person:
    pass
```

Бул учурда адамды билдирген Person классы аныкталды. Бул учурда класста эч кандай методдор же атрибуттар аныкталган эмес. Бирок анда бир нерсе аныкталышы керек болгондуктан, класс функционалынын ордуна **pass**операторду колдонулат. Бул оператор синтаксистик жактан кандайдыр бир кодду аныктоо зарыл болгондо колдонулат, бирок тапшырмадан улам бизге коддун кереги жок жана конкреттүү коддун ордуна pass операторун киргизебиз.

Классты түзгөндөн кийин, бул класстын объекттерин аныктоого болот. Мисалы:

```
class Person: # класстын_аты: class.py
    pass

tom = Person() # tom объектисин аныктоо
bob = Person()     # bob объектисин аныктоо
```

"Person" классы аныкталгандан кийин, бул класстын эки объектиси түзүлөт — tom жана bob.

Объектти түзүү үчүн конструктор деп аталган атайын функция колдонулат. Конструктор класстын аты менен аталат жана класстын объектисин кайтарат. Башкача айтканда, бул учурда Person() чакыруусу конструкторду чакырууну билдирет.

Ар бир класста демейки боюнча параметрлери жок конструктор болот:

```
tom = Person() # Person() — конструкторду чакыруу, ал Person классынын объектисин кайтарат
```

Конструкторлор

Ошентип, класстын объектисин түзүү үчүн конструктор колдонулат. Жогоруда биз Person классынын объекттерин түзгөнүбүздө, биз параметрлерди кабыл албаган жана бардык класстарда кыйыр түрдө болгон **демейки конструкторду** колдондук.

Бирок, биз класстарда **__init__()** (эки жагында эки сызыкчасы бар) деп аталган атайын методдун жардамы менен конструкторду ачык аныктай алабыз.

Мисалы, **Person классына конструкторду кошуп, аны өзгөртөлү:**

```
class Person:
    # конструктор
    def __init__(self):

        print("Person объектисин түзүү")
tom = Person() # Консолго: "Person объектисин түзүү" чыгат
```

Ошентип, бул жерде Person классынын кодунда конструктор – __init__ функциясы аныкталган.

Конструктор жок дегенде бир параметрди — **учурдагы объектке шилтемени** (self) кабыл алышы керек. Адатта, конструкторлор объект түзүлгөндө аткарылуучу аракеттерди аныктоо үчүн колдонулат.

Эми объект түзүлгөндө:

tom = Person()

Person классындагы __init__() конструкторунун чакырылышы аткарылат, ал консолго "Person объектисин түзүү" сабын чыгарат.

Белгилей кетүүчү нерсе, конструктор иш жүзүндө жөнөкөй функцияны билдирет, бирок конструкторду чакыруу үчүн __init__ эмес, класстын аты колдонулат. Мындан тышкары, конструкторду чакырууда self параметрине эч кандай маани ачык түрдө берилбейт. Программаны аткарууда Python selfти динамикалык түрдө аныктайт.

Объект Атрибуттары

Атрибуттар объекттин абалын сактайт. Класстын ичинде атрибуттарды аныктоо жана орнотуу үчүн self сөзүн колдонсо болот.

Мисалы, төмөнкү Person классын аныктайлы:

Python

```
class Person:

    def __init__(self, name, age):
        self.name = name     # Адамдын аты
        self.age = age       # Адамдын жашы


tom = Person("Tom", 22)

# Атрибуттарга кайрылуу
# Маанилерди алуу
print(tom.name)    # Tom
print(tom.age)     # 22
# Маанини өзгөртүү
tom.age = 37
print(tom.age)     # 37
```

Эми Person классынын конструктору дагы эки параметрди — name жана age кабыл алат. Бул параметрлер аркылуу түзүлүп жаткан адамдын аты жана жашы конструкторго берилет. Конструктордун ичинде эки атрибут — name жана age орнотулат (шарттуу түрдө адамдын аты жана жашы):

Python

```
def __init__(self, name, age):
    self.name = name
    self.age = age
```

self.name атрибутуна name өзгөрмөсүнүн мааниси ыйгарылат. age атрибуту age параметринин маанисин алат. Атрибуттардын аталышы параметрлердин аталыштарына дал келиши шарт эмес.
Эгерде биз класста параметрлери бар (selftан башка) __init__ конструкторун аныктаган болсок, анда конструкторду чакырганда бул параметрлерге маанилерди берүү керек:
Python
tom = Person("Tom", 22)
Башкача айтканда, бул учурда name параметрине "Tom" сабы, ал эми age параметрине 22 саны берилет.
Андан кийин, биз объекттин аты аркылуу анын атрибуттарына кайрылып, алардын маанилерин ала алабыз жана өзгөртө алабыз:
Python
print(tom.name) # name атрибутунун маанисин алуу
tom.age = 37 # age атрибутунун маанисин өзгөртүү

Бир нече объект түзүү
Ошол сыяктуу эле, биз атрибуттардын ар кандай маанилери менен Person классынын ар кандай объекттерин түзө алабыз:

```
class Person:

    def __init__(self, name, age):
        self.name = name     # Адамдын аты
        self.age = age       # Адамдын жашы


tom = Person("Tom", 22)
bob = Person("Bob", 43)

print(tom.name)      # Tom
print(bob.name)      # Bob
```

Бул жерде Person классынын эки объектиси түзүлөт: tom жана bob. Алар Person классынын аныктамасына ылайык келет, атрибуттардын бирдей топтомуна ээ, бирок алардын абалы ар башка болот. Жана ар бир учурда Python self объектисин динамикалык түрдө аныктайт.
Мисалы, төмөнкү учурда:

```
tom = Person("Tom", 22)
```

Бул **tom** объектиси болот.
Ал эми чакырууда:
Python

```
bob = Person("Bob", 43)
```

Бул **bob** объектиси болот.

Динамикалык Атрибуттар
Негизи, биз атрибуттарды класстын ичинде аныктоого милдеттүү эмеспиз – Python муну коддон тышкары динамикалык түрдө жасоого мүмкүндүк берет:

```
class Person:

    def __init__(self, name, age):
        self.name = name     # Адамдын аты
        self.age = age       # Адамдын жашы
```

```
tom = Person("Tom", 22)

tom.company = "Microsoft" # Динамикалык атрибутту орнотуу
print(tom.company)        # Microsoft
```
Бул жерде company атрибуту динамикалык түрдө орнотулган, ал адамдын жумуш ордун сактайт. Жана орнотулгандан кийин, биз анын маанисин да ала алабыз. Ошол эле учурда, мындай аныктоо каталарга алып келиши мүмкүн. Мисалы, эгерде биз атрибутту аныктаганга чейин ага кайрылууга аракет кылсак, программа ката берет:
Python
```
tom = Person("Tom", 22)
print(tom.company) # ! Ката: AttributeError: Person object has no attribute 'company'
```

Класс Методдору (Методы классов)
Класс методдору иш жүзүндө класстын ичинде аныкталган жана анын жүрүм-турумун аныктаган функцияларды билдирет.
Мисалы, Person классын бир метод менен аныктайлы:

```
class Person:       # Person классын аныктоо
   def say_hello(self):
      print("Hello")

tom = Person()
tom.say_hello()     # Hello
```
Бул жерде say_hello() методу аныкталган, ал шарттуу түрдө саламдашууну аткарат – консолго сапты чыгарат. Каалаган класстын методдорун аныктоодо, конструктор сыяктуу эле, методдун биринчи параметри учурдагы объектке шилтемени билдирет, ал шарттуу түрдө self деп аталат. Бул шилтеме аркылуу класстын ичинде биз учурдагы объектинин функционалдуулугуна кайрыла алабыз. Бирок методду чакырууда бул параметр эске алынбайт.
Объекттин атын колдонуп, биз анын методдоруна кайрыла алабыз. Методдорго кайрылуу үчүн чекит нотациясы колдонулат – объекттин атынан кийин чекит коюлуп, андан кийин метод чакырылат:
объект.метод([методдун параметрлери])
Мисалы, саламдашууну консолго чыгаруу үчүн say_hello() методуна кайрылуу:

```
tom.say_hello()   # Hello
```
Жыйынтыгында, бул программа консолго "Hello" сабын чыгарат.
Эгер метод башка параметрлерди кабыл алышы керек болсо, алар self параметринен кийин аныкталат жана мындай методду чакырганда аларга маанилерди берүү зарыл:

```
class Person:       # Person классын аныктоо
   def say(self, message):  # Метод
      print(message)

tom = Person()
tom.say("Hello METANIT.COM") # Hello METANIT.COM
```
Бул жерде say() методу аныкталган. Ал эки параметрди кабыл алат: self жана message. Жана экинчи параметр — message үчүн методду чакырууда маани берүү керек.

Атрибуттарга жана Методдорго Кайрылуу
Класс методдорунун ичинде объектинин атрибуттарына жана методдоруна кайрылуу үчүн да self сөзү колдонулат:

self.атрибут # атрибутка кайрылуу
self.метод # методго кайрылуу
Мисалы, төмөнкү Person классы:

```
class Person:

    def __init__(self, name, age):
        self.name = name        # Адамдын аты
        self.age = age       # Адамдын жашы

    def display_info(self):
        # Методдо атрибуттарга self аркылуу кайрылуу
        print(f"Name: {self.name}  Age: {self.age}")

tom = Person("Tom", 22)
tom.display_info()          # Name: Tom  Age: 22

bob = Person("Bob", 43)
bob.display_info()          # Name: Bob  Age: 43
```

Бул жерде display_info() методу аныкталган, ал консолго маалыматты чыгарат. Жана методдо объектинин атрибуттарына кайрылуу үчүн self сөзү колдонулат: self.name жана self.age.
Жыйынтыгында биз консолго төмөнкүдөй чыгарууну алабыз:
Name: Tom Age: 22
Name: Bob Age: 43

Тапшырма

1. Фамилия, дарек, телефон номер, элек. почта – кошуу
2. 4 адамдын маалыматын экранга чыгаруу

Деструкторлор (Деструкторы)
Конструкторлордон тышкары, Python'догу класстар атайын методдорду – деструкторлорду аныктай алышат, алар объект жок кылынганда чакырылат.
Деструктор __del__(self) методун билдирет, ага конструктордогудай эле учурдагы объектке шилтеме берилет. Деструктордо объект жок кылынганда аткарылышы керек болгон аракеттер аныкталат, мисалы, объект колдонгон кээ бир ресурстарды бошотуу же жок кылуу.
Деструктор интерпретатор тарабынан автоматтык түрдө чакырылат, биз аны ачык чакыруунун кереги жок. Эң жөнөкөй мисал:

```
class Person:

    def __init__(self, name):
        self.name = name
        print("Аты", self.name, "болгон адам түзүлдү")

    def __del__(self):
        print("Аты", self.name, "болгон адам жок кылынды")
```

tom = Person("Tom")
Бул жерде деструктордо жөн гана Person объектисинин жок кылынгандыгы тууралуу билдирүү чыгарылат. Программа бир Person объектисин түзөт жана ага шилтемени tom өзгөрмөсүндө сактайт. Объектти түзүү конструктордун аткарылышын чакырат. Программа аяктаганда, tom объектисинин деструктору автоматтык түрдө аткарылат.
Натыйжада, программанын консолдук чыгаруусу төмөнкүдөй болот:
Аты Tom болгон адам түзүлдү
Аты Tom болгон адам жок кылынды
Дагы бир мисал:

```
class Person:

    def __init__(self, name):
        self.name = name
        print("Аты", self.name, "болгон адам түзүлдү")

    def __del__(self):
        print("Аты", self.name, "болгон адам жок кылынды")

def create_person():
    tom = Person("Tom")

create_person()
print("Программанын аягы")
```

Бул жерде Person объектиси create_person функциясынын ичинде түзүлүп жана колдонулгандыктан, түзүлгөн Person объектисинин өмүрү бул функциянын чөйрөсү менен чектелет. Демек, функция аткарылып бүткөндө, Person объектисинин деструктору чакырылат.
Натыйжада биз төмөнкүдөй консолдук чыгарууну алабыз:
Аты Tom болгон адам түзүлдү
Аты Tom болгон адам жок кылынды
Программанын аягы

Тест

1. Python'до класс деген эмне?

a. Операция системасынын бир бөлүгү **б.** Объекттерди түзүү үчүн шаблон
c. Өзүнчө Python командасы д. Өзгөрмөнүн тиби

Туура жооп: б

2. Объект деген эмне?

a. Сап (строка) б. Сан
c. Класстын инстанциясы (көчүрмөсү) д. Модуль

Туура жооп: c

3. Классты түзүү үчүн кайсы ачкыч сөз колдонулат?

a. def б. Class c. object д. init

Туура жооп: б

4. Класстын конструктору кантип аталат?

a. start б. create c. __init__ д. build

Туура жооп: c

5. Класс методдорунда кайсы милдеттүү параметр болушу керек?

a. obj б. Class c. Me д. self

Туура жооп: д

6. self ачкыч сөзү эмне кылат?

а. Объектти жок кылат б. Учурдагы объектке шилтеме берет

с. Классты түзөт д. Эч нерсе кылбайт

Туура жооп: б

7. Объекттин методун кантип чакыруу керек?

а. method(obj) б. obj.method()

с. call method obj д. method->obj

Туура жооп: б

8. Жок атрибутка кайрылууда эмне болот?

а. Программа көңүл бурбайт б. None кайтарылат

с. AttributeError катасы пайда болот д. Атрибут автоматтык түрдө түзүлөт

Туура жооп: с

9. Класстан тышкары объектке атрибут кошууга болобу?

а. Ооба б. Жок

с. Эгерде гана debug режими күйгүзүлсө д. Эгерде гана Python 2де болсо

Туура жооп: а

10. Класс методу деген эмне?

а. Класстын ичиндеги өзгөрмө б. Класстын ичинде аныкталган функция

с. Өзүнчө файл д. Компилятор командасы

Туура жооп: б

11. Деструктор кантип аталат?

а. delete б. Destroy с. __del__ д. stop

Туура жооп: с

12. tom = Person("Tom") жазуусу эмне кылат?

а. Модуль түзөт б. Person конструкторун чакырат

с. Person объектисин жок кылат д. Person функциясын түзөт

Туура жооп: б

13. selfтен тышкары __init__ методуна параметрлерди кошууга болобу?

а. Жок б. Ооба, каалаган параметрлерди

с. Бир гана параметр д. Бир гана саптарды (строкаларды)

Туура жооп: б

14. Атрибуттун маанисин кантип алууга болот?

а. get.attribute(obj) б. attribute.obj с. obj.attribute д. attribute(obj)

Туура жооп: с

15. Атрибуттун маанисин кантип өзгөртүүгө болот?

а. obj.attribute == value б. obj.attribute = value

с. change(obj.attribute, value) д. obj.attribute -> value

Туура жооп: б

3. **Тема: Инкапсуляция, атрибуттар жана касиеттер (Свойства)**

План:

1. Python'догу Ачык (Публичные) атрибуттар.
2. Жабык (Приватные) атрибуттар (__attribute).
3. Жетүү методдору: Геттерлер жана Сеттерлер.
4. Касиеттер (@property) – заманбап ыкма.
5. Ыкмаларды практикалык салыштыруу.
6. Жыйынтыктоочу көнүгүү. Касиеттер аркылуу маалыматтарды текшерүү менен өз классын ишке ашыруу.

Максаты:

Python'до инкапсуляция кантип ишке ашырыларын түшүнүү, жабык атрибуттар эмне үчүн керектигин билүү жана объектинин маалыматтарына жетүүнү башкаруунун үч жолун өздөштүрүү:

1. Жабык атрибуттар (__attr),
2. Геттерлер жана сеттерлер,
3. Касиеттер (@property).

1. Ачык атрибуттар (демейки жетүү)

Демейки боюнча, класстардагы атрибуттар жалпыга жеткиликтүү (общедоступные), демек, программанын каалаган жеринен биз объектинин атрибутун алып, аны өзгөртө алабыз.

Мисалы:

```
class Person:

    def __init__(self, name, age):
        self.name = name     # Атын орнотобуз
        self.age = age       # Жашын орнотобуз

    def print_person(self):
        print(f"Аты: {self.name}\ t Жашы: {self.age}")

tom = Person("Tom", 39)

tom.name = "Человек-паук"    # name атрибутун өзгөртөбүз
tom.age = -129               # age атрибутун өзгөртөбүз

tom.print_person()           # Аты: Человек-паук     Жашы: -129
```

Бирок бул учурда, мисалы, адамдын жашына же атына туура эмес маанини, мисалы, терс жашты ыйгарып коюшубуз мүмкүн, ошондуктан объектинин атрибуттарына жетүүнү көзөмөлдөө маселеси келип чыгат.

2. Инкапсуляция жана жабык атрибуттар

Бул маселе инкапсуляция түшүнүгү менен тыгыз байланыштуу. Инкапсуляция – бул объектке багытталган программалоонун негизги концепциясы, ал функционалды жашырууну жана ага сырттан түз жетүүнү болтурбоону билдирет.

Python программалоо тили жабык (приваттык) же чектелген атрибуттарды аныктоого мүмкүндүк берет. Бул үчүн атрибуттун аты кош астынкы сызыкчадан башталышы керек – __name.

Мисалы, мурунку программаны өзгөртүп, эки атрибутту тең (name жана age) жабык кылабыз:

```
class Person:
```

```
    def __init__(self, name, age):
        self.__name = name     # Атын орнотобуз (жабык)
        self.__age = age       # Жашын орнотобуз (жабык)

    def print_person(self):
        print(f"Аты: {self.__name}\tЖашы: {self.__age}")


tom = Person("Tom", 39)

tom.__name = "Человек-паук"    # __name атрибутун өзгөртүүгө аракет кылабыз
tom.__age = -129               # __age атрибутун өзгөртүүгө аракет кылабыз

tom.print_person()          # Аты: Tom     Жашы: 39
```

Биз __name жана __age атрибуттарына жаңы маанилерди орнотууга аракет кылганыбыз менен, print_person методунун жыйынтыгы атрибуттардын маанилери өзгөрбөгөнүн көрсөтөт:

Python

```
tom.print_person() # Аты: Tom     Жашы: 39
```

Иштөө Принциби (Name Mangling)

Кош астынкы сызыкча менен башталган (__attribute) атрибутту жарыялаганда, Python чындыгында _ClassName__attribute үлгүсү боюнча аталган атрибутту аныктайт. Жогорудагы учурда, _Person__name жана _Person__age атрибуттары түзүлөт.

Ошондуктан, биз ал атрибутка ошол эле класстан гана кайрыла алабыз. Ал эми класстан тышкары кайрылууга аракет кылсак, ал иштебейт.

```
tom.__age = 43
```

Бул учурда, жөн гана динамикалык түрдө жаңы __age атрибуту аныкталат, бирок анын self.__age же self._Person__age атрибутуна эч кандай тиешеси жок.

Бирок, бул жердеги жабыктык бир аз салыштырмалуу. Мисалы, биз атрибуттун толук атын колдоно алабыз:

```
tom = Person("Tom", 39)

tom._Person__name = "Человек-паук"  # _Person__name аркылуу өзгөртөбүз
tom.print_person()                  # Аты: Человек-паук     Жашы: 39
```

Бирок тышкы коддун автору атрибуттардын кантип аталганын (мисалы, _Person__name экенин) билиши керек. Ошондуктан, кош астынкы сызыкча эскертүү катары кызмат кылат, бул атрибутка сырттан жетүүнү чектейт.

3. Жетүү методдору: геттерлер жана сеттерлер

Жабык атрибуттарга кантип кайрылуу керек деген суроо туулушу мүмкүн. Бул үчүн адатта атайын жетүү методдору колдонулат:

- Геттер (Getter) – атрибуттун маанисин алууга мүмкүндүк берет.
- Сеттер (Setter) – атрибуттун маанисин орнотууга мүмкүндүк берет.

Жогоруда аныкталган классты өзгөртүп, ага жетүү методдорун киргизели:

```
class Person:
    def __init__(self, name, age):
        self.__name = name
        self.__age = age

    # Сеттер: Жашты орнотуу үчүн
    def set_age(self, age):
        if 0 < age < 110:
            self.__age = age
        else:
            print("Жаштын туура эмес маани си")

    # Геттер: Жашты алуу үчүн
    def get_age(self):
        return self.__age

    # Геттер: Атын алуу үчүн
    def get_name(self):
        return self.__name

    def print_person(self):
        print(f"Аты: {self.__name}\tЖашы: {self.__age}")

tom = Person("Tom", 39)
tom.print_person()        # Аты: Tom  Жашы: 39

tom.set_age(-3486)        # Жаштын маани си туура эмес
tom.set_age(25)

tom.print_person()        # Аты: Tom  Жашы: 25
```

Методдор аркылуу атрибуттарга жетүүнү камсыздоо кошумча логиканы кошууга мүмкүндүк берет. Мисалы, set_age методунда биз берилген жаш маани си 0дөн 110го чейинки диапазондо экенин текшеребиз.
Ошондой эле, ар бир жабык атрибут үчүн мындай жуп методдорду түзүү милдеттүү эмес. Мисалда, адамдын атын конструктордон гана орното алабыз, ал эми аны алуу үчүн get_name методу аныкталган.

4. Касиеттер (@property) – заманбап ыкма

Жогоруда биз жетүү методдорун түзүүнү карадык. Бирок Python'до дагы бир – касиеттер деп аталган жарашыктуу ыкма бар. Бул ыкма @ символу менен башталуучу аннотацияларды колдонууну камтыйт.

- Геттер-касиетти түзүү үчүн методдун үстүнө @property аннотациясы коюлат.
- Сеттер-касиетти түзүү үчүн методдун үстүнө @геттердин_аты.setter аннотациясы орнотулат.

Person классын аннотацияларды колдонуу менен кайра жазалы:

```
class Person:
    def __init__(self, name, age):
        self.__name = name
        self.__age = age

    # 1. Касиет-геттер: Жашты алуу үчүн
```

```
    @property
    def age(self):
        return self.__age

    # 2. Касиет-сеттер: Жашты орнотуу үчүн (Геттердин атын колдонуу менен @age.setter)
    @age.setter
    def age(self, age):
        if 0 < age < 110:
            self.__age = age
        else:
            print("Жаштын туура эмес мааниси")

    # 3. Касиет-геттер: Атын алуу үчүн (сеттери жок, өзгөртүүгө болбойт)
    @property
    def name(self):
        return self.__name

    def print_person(self):
        print(f"Аты: {self.__name}\tЖашы: {self.__age}")

tom = Person("Tom", 39)
tom.print_person()       # Аты: Tom  Жашы: 39

tom.age = -3486          # Жаштын туура эмес мааниси (Сеттерге кайрылуу)
print(tom.age)           # 39 (Геттерге кайрылуу)

tom.age = 25             # (Сеттерге кайрылуу)
tom.print_person()       # Аты: Tom  Жашы: 25
```

Негизги Эрежелер:

1. Касиет-сеттер касиет-геттерден кийин аныкталышы керек.
2. Сеттер да, геттер да бирдей аталышта болушу керек – бул учурда age.
3. Геттер age деп аталгандыктан, сеттердин үстүнө @age.setter аннотациясы орнотулат.
4. Андан кийин, геттерге да, сеттерге да биз жөнөкөй атрибут сыяктуу кайрылабыз: tom.age (алуу) же tom.age = 25 (орнотуу).
5. Сеттерди аныктабастан, бир гана геттерди аныктоого болот, мисалы name касиетинде – аны өзгөртүүгө болбойт, маанисин гана алууга болот.

Тест

1. Төмөндөгү концепциялардын кайсынысы объектке багытталган программалоодогу (ООП) инкапсуляцияны эң жакшы сүрөттөйт?

а. Ата-эне класстардан туунду класстарды түзүү.

б. Маалыматтарды (атрибуттарды) ал маалыматтар менен иштеген методдор менен байланыштыруу жана ишке ашыруу деталдарын жашыруу.

с. Бир эле методдун контекстке жараша ар кандай жүрүм-турумга ээ болуу мүмкүнчүлүгү.

д. Класстын атрибуттары үчүн маалымат түрлөрүн аныктоо.

Жооп: б

2. Материалга ылайык, Python'до келишим боюнча "жабык" (жашырылган) деп эсептелген атрибут кантип жарыяланат?

а. _attribute_name (бир астынкы сызыкча менен)

б. __attribute_name (эки астынкы сызыкча менен)

с. private attribute_name

д. attribute_name (сызыкчасыз)

Жооп: б

3. Python'до кош астынкы сызыкчадан башталган ат менен атрибутту (мисалы, __age) жарыялаганыңызда эмне болот?
а. Python ката берет, анткени жабык атрибуттарды түз жарыялоого тыюу салынат.
б. Атрибут катуу жабык болуп калат жана класстын ичинде да өзгөртүү мүмкүн болбойт.
с. Python аны _ClassName__attributeName формасына "кайра атайт" (name mangling), тышкы жетүүнү кыйындатат.
д. Атрибутту окууга гана мүмкүн болот.
Жооп: с
4. class Animal: def __init__(self): self.__weight = 50 классын карап көрөлү. dog = Animal() объектисинин "жабык" __weight атрибутун өзгөртүүгө аракет кылганда, кайсы туюнтма КАТАГА алып келбейт?
а. dog.__weight = 10
б. dog._Animal__weight = 10
с. print(dog.__weight)
д. dog.weight = 10
Жооп: б (Бул "кайра аталган" атрибутка түз кайрылуу, бирок сунушталбайт.)

5. Атрибут үчүн сеттерди (setter) колдонуунун негизги максаты эмне?
а. Атрибуттун маанисин алуу (окуу).
б. Атрибуттун маанисин текшерүү логикасын кошуу мүмкүнчүлүгү менен орнотуу (өзгөртүү).
с. Объект түзүлгөндө атрибутту инициализациялоо.
д. Объекттен атрибутту жок кылуу.
Жооп: б
6. Материалга ылайык, сеттерде (мисалы, set_age(self, age)) кошумча логика эмне үчүн керек?
а. Конструктордо age атрибутунун инициализацияланышына жол бербөө үчүн.
б. __age атрибутун катуу жабык кылуу үчүн.
с. Орнотулуп жаткан маанинин тууралыгын текшерүү үчүн (мисалы, жаш оң сан болушу керек).
д. Маанилерди кэштөөнү камсыздоо үчүн.
Жооп: с
7. Эгерде атрибут үчүн геттер гана (get_name), бирок сеттер (set_name) аныкталбаса, анда бул эмнени билдирет?
а. Атрибутту конструктордо гана орнотууга жана каалаган убакта окууга болот.
б. Атрибутту такыр окууга болбойт.
с. Атрибутту стандарттык синтаксисти колдонуу менен окууга жана орнотууга болот (методдорсуз).
д. Атрибутту орнотууга болот, бирок окууга болбойт.
Жооп: а
8. Python'до касиет-геттерди (маанини алуу үчүн метод) түзүү үчүн кайсы аннотация колдонулат?
а. @getter
б. @property
с. @get.attribute
д. @read_only
Жооп: б
9. Value деген аты бар геттер үчүн касиет-сеттерди түзүү үчүн кайсы аннотация колдонулат?
а. @setter(value)
б. @value.set
с. @value.setter

д. @property.setter
Жооп: с
10. Ачык геттерлер/сеттерлерге (get_age(), set_age(age)) салыштырмалуу касиеттерди (@property, @...setter) колдонуунун негизги артыкчылыгы эмнеде?
а. Алар атрибуттардын катуу жабыктыгын камсыз кылат.
б. Алар текшерүү логикасын сактоо менен, методдорго атрибуттар сыяктуу кайрылууга мүмкүндүк берет (tom.age = 25 ордуна tom.set_age(25)).
с. Касиеттер тезирээк иштейт.
д. Касиеттер геттерди алдын ала аныктабастан сеттерди аныктоого мүмкүндүк берет.
Жооп: б
11. Эгерде класста касиет-геттер @property def age(self): ... жана касиет-сеттер @age.setter def age(self, age): ... аныкталган болсо, print(tom.age) туюнтмасы кантип чечмеленет?
а. age(self, age) сеттер функциясы чакырылат.
б. age(self) геттер функциясы чакырылат.
с. Бул катага алып келет, анткени кашаалар жок.
д. tom.__age атрибутуна түз жетүү ишке ашат.
Жооп: б
12. Эмне үчүн аннотациялуу касиеттердеги (@property) геттер менен сеттер бирдей аталышта (def age(self):) болот?
а. Python контекстке негизделип (алуу же ыйгаруу) кайсы методду чакырууну автоматтык түрдө аныктай алышы үчүн.
б. Бул бардык класс методдору үчүн Python синтаксисинин талабы.
с. Геттер менен сеттер ар кандай аттарга ээ боло албайт.
д. Бул жөн гана макулдашуу, бирок алар ар кандай аталышы мүмкүн.
Жооп: а
13. Эгерде Person классында (жабык __age менен) age касиет-геттери гана аныкталса, анан tom.age = 50 аткарылууга аракет кылынса, эмне болот?
а. Геттер чакырылып, маанини орнотот.
б. Жаңы динамикалык tom.age = 50 өзгөрмөсү түзүлөт.
с. Сеттер жок болгондуктан, AttributeError катасы пайда болот.
д. Жабык __age атрибуту түздөн-түз өзгөртүлөт.
Жооп: с
14. Аннотациялуу касиет-геттерди жана касиет-сеттерди аныктоо тартиби жөнүндө төмөндөгү кайсы билдирүү туура?
а. Касиет-сеттер ар дайым касиет-геттерден мурун аныкталышы керек.
б. Касиет-геттер (@property) ар дайым тиешелүү касиет-сеттерден (@геттердин_аты.setter) мурун аныкталышы керек.
с. Тартиптин мааниси жок.
д. Сеттер башка класста гана аныкталышы керек.
Жооп: б
15. Эгерде __name атрибуту үчүн @property def name(self): ... аннотациясы менен геттер гана түзүлсө, анда анын маанисин кантип өзгөртүүгө болот?
а. Стандарттык синтаксис аркылуу tom.name = "NewName".
б. @name.setter аннотациясы менен сеттерди түзүү аркылуу.
с. "Кайра аталган" ат аркылуу өзгөртүү, мисалы, tom._Person__name = "NewName".
д. Б) жана С) варианттарынын экөө тең. (С) техникалык жактан мүмкүн, бирок инкапсуляцияны бузат, ал эми Б) класстын дизайнынын көз карашынан алганда туура жол).
Жооп: д

4. Тема: Мурастоо (Наследование)
План:
1. Мурастоого киришүү.
2. Жалгыз мурастоо (Мисал: Person -> Employee).
3. Көптүк мурастоо (Множественное наследование).

Максаты
Python тилиндеги мурастоонун негизги концепцияларын, синтаксисин жана колдонуу мисалдарын жалгыз жана көптүк мурастоону, ошондой эле көптүк мурастоодо методдордун аталыштарын чечүү эрежелерин жалпылоо жана структуралаштыруу.

1. Мурастоого Киришүү
Мурастоо – бул бар болгон класстын негизинде жаңы классты түзүүгө мүмкүндүк берет. Инкапсуляция менен бирге мурастоо объектке багытталган программалоонун (ООП) негизги принциптеринин бири болуп саналат.
Мурастоонун негизги түшүнүктөрү суперкласс жана подкласс болуп саналат. Подкласс суперкласстан бардык жалпыга жеткиликтүү атрибуттарды жана методдорду мурастайт.

- Суперкласс: Базалык класс (base class) же ата-эне класс (parent class) деп да аталат.
- Подкласс: Туунду класс (derived class) же балдар классы (child class) деп да аталат.

Мурастоо үчүн синтаксис төмөнкүдөй көрүнөт:

```
class подкласс (суперкласс):
  подкласстын_методдору
э
```

2. Жалгыз Мурастоо (Person -> Employee Мисалы)
Мисалы, бизде адамды билдирген Person классы бар:

```
class Person:

  def __init__(self, name):
    self.__name = name  # адамдын аты

  @property
  def name(self):
    return self.__name

  def display_info(self):
    print(f"Name: {self.__name} ")
```

Эми бизге кайсы бир ишканада иштеген кызматкердин классы керек дейли. Биз башынан баштап жаңы класс түзсөк болот, мисалы, Employee классы:

```
class Employee:

  def __init__(self, name):
```

```
        self.__name = name  #   кызматкердин аты
        # ... Person классындагы коддун кайталанышы

    # ... башка кайталанган методдор

    def work(self):
        print(f"{self.name} works")
```

Бирок Employee классында Person классындагыдай эле атрибуттар жана методдор болушу мүмкүн, анткени кызматкер дагы адам. Жогорудагы Employee классында work методу гана кошулган, калган код Person классынын функционалын кайталайт. Бир класстын функционалын экинчисинде кайталабоо үчүн, бул учурда мурастоону колдонуу туура.
Ошентип, Employee классын Person классынан мурастайлы:

```
class Person:
    # ... __init__, name property, display_info методдору

    def __init__(self, name):
        self.__name = name  # адамдын аты

    @property
    def name(self):
        return self.__name

    def display_info(self):
        print(f"Name: {self.__name} ")

class Employee(Person):  # Person классынан мурастоо

    def work(self):
        print(f"{self.name} works")

# Колдонуу:
tom = Employee("Tom")
print(tom.name)      # Tom (Person классынан мурасталды)
tom.display_info()   # Name: Tom (Person классынан мурасталды)
tom.work()           # Tom works (Employee классынын өзүнүн методу)
```

Employee классы Person классынын функционалын толугу менен кабыл алып, болгону work() методун кошот. Демек, Employee объектисин түзгөндө, биз Person классынан мурастаган конструкторду колдоно алабыз:

```
tom = Employee("Tom")
```

Ошондой эле, мурасталган атрибуттарга/касиеттерге жана методдорго кайрыла алабыз. Маанилүү: Employee классы үчүн __name сыяктуу жабык (приваттык) атрибуттар жеткиликтүү ЭМЕС. Мисалы, биз work методунда жеке атрибутка self.__name аркылуу кайрыла албайбыз:

```
def work(self):
    print(f"{self.__name} works")  # ! Ката болот
```

3. Көптүк мурастоо (Множественное наследование)
Python тилинин айырмалоочу өзгөчөлүктөрүнүн бири – көптүк мурастоону колдоо, б.а., бир классты бир нече класстан мурастоого болот:

```
# Кызматкер классы
class Employee:
    def work(self):
        print("Employee works")

# Студент классы
class Student:
    def study(self):
        print("Student studies")

# Иштеген студент классы (Employee жана Student класстарынан мурастайт)
class WorkingStudent(Employee, Student):
    pass

# Колдонуу:
tom = WorkingStudent()
tom.work()      # Employee works
tom.study()     # Student studies
```

Бул жерде WorkingStudent классы эч кандай функционалды аныктабайт, ошондуктан анда pass оператору берилген. WorkingStudent жөн гана эки класс – Employee жана Student функционалын мурастайт. Демек, бул класстын объектисинде биз эки класстын методдорун тең чакыра алабыз.
Мурасталып жаткан класстар функционалдуулугу боюнча татаалыраак болушу мүмкүн, мисалы, конструкторлору бар класстар:

```
class Employee:
    # ... __init__ (name) жана work() методдору

class Student:
    # ... __init__ (name) жана study() методдору

class WorkingStudent(Employee, Student):
    pass

tom = WorkingStudent("Tom") # Эскертүү: Конструктор Employee классынан алынат

tom.work()      # Tom works
tom.study()     # Tom studies
```

4. Көптүк Мурастоодогу аталыштарды чечүү
Көптүк мурастоо ыңгайлуу көрүнгөнү менен, эгерде мурасталып жаткан эки класста тең аттары окшош методдор/атрибуттар болсо, бул башаламандыкка алып келиши мүмкүн. Мисалы:

```
class Employee:
    def do(self):
        print("Employee works")
```

```
class Student:
    def do(self):
        print("Student studies")

class WorkingStudent(Employee, Student):
    pass

tom = WorkingStudent()
tom.do()  # ? Кайсы do() методу чакырылат?
```

Эки базалык класс тең – Employee жана Student – консолго ар кандай сапты чыгаруучу do методун аныктайт. WorkingStudent мурасчы классы бул ишке ашыруулардын кайсынысын колдонот?

Классты аныктоодо базалык класстардын тизмесинде биринчи Employee классы келет:

Python

```
class WorkingStudent(Employee, Student)
```

Ошондуктан, do методун ишке ашыруу Employee классынан алынат.

Эгерде биз класстардын иретин өзгөртсөк:

```
class WorkingStudent(Student, Employee)
```

анда Student классынын ишке ашырылышы колдонулмак.

MRO (Method Resolution Order)

Зарыл болгон учурда, биз базалык класстардын функционалын колдонуу иретин программалык түрдө көрө алабыз. Бул үчүн __mro__ атрибуту же mro() методу колдонулат:

```
print(WorkingStudent.__mro__)
print(WorkingStudent.mro())
# Жыйынтыгы: (<class '__main__.WorkingStudent'>, <class '__main__.Employee'>, <class '__main__.Student'>, <class 'object'>)
```

Бул тизме Python класстарды жана методдорду издөөдө кайсы иретте карап чыгаарын көрсөтөт.

Тест

1. ООПдо мурастоонун негизги түшүнүгү кайсы?

a. Атрибут жана Функция
б. Инкапсуляция жана Полиморфизм
c. Подкласс жана Суперкласс
д. Объект жана Инстанция (Көчүрмө)

Жооп: c

2. Employee классын Person классынан мурастоо үчүн туура синтаксис кайсы?

a. class Employee(Person):
б. class Employee inherits Person:
c. class Employee:
д. inherit Person class Employee extends Person:

Жооп: a

3. Мурастоо жүрүп жаткан класс кандай аталат?

a. Туунду класс
б. Балдар классы
c. Подкласс
д. Базалык класс (же Суперкласс)

Жооп: д

4. Төмөнкү коддо, Employee классынын work() методу Person классынын кайсы мүчөсүнө кайрыла алат?

```
class Person:
    def __init__(self, name):
        self.__name = name # Жабык атрибут
```

```
    @property
    def name(self):
        return self.__name # Ачык касиет (геттер)

class Employee(Person):
    def work(self):
        # Кайсы атрибут же метод жеткиликтүү?
        pass
```

а. self.__name га гана б. self.name ка гана
с. Бардык мүчөлөргө, анын ичинде self.__name га
д. Методдорго гана, бирок атрибуттарга эмес
Жооп: б

5. Person жана Employee класстарынын мисалында Employee классын нөлдөн түзүүгө караганда мурастоону колдонгон оңбу?
а. Мурастоо полиморфизмди камсыз кылат.
б. Мурастоо жабык методдорду түзүүгө мүмкүндүк берет.
с. Мурастоо коддун кайталанышынан сактанууга мүмкүндүк берет.
д. Мурастоо Python'до бардык класстар үчүн милдеттүү.
Жооп: с

6. Эгерде суперкласста атрибут жабык деп жарыяланса (кош __ колдонуу менен), ал подкласста жеткиликтүү болобу?
а. Ооба, дайыма.
б. Ооба, бирок жалпыга ачык геттер-методдор аркылуу гана.
с. Жок, жабык атрибуттар мурасталат.
д. Жок, алар түздөн-түз мурасталып алынбайт.
Жооп: б

7. Python'до көптүк мурастоо кандай артыкчылык берет?
а. Бир гана конструктор чакырылышына кепилдик берет.
б. Мурасчы класска бир нече көз карандысыз ата-тегинин функционалын бириктирүүгө мүмкүндүк берет.
с. Методдордун аткарылышынын катуу иретин камсыз кылат.
д. Методдордун аталыштарындагы бардык конфликттерди автоматтык түрдө чечет.
Жооп: б

8. Көптүк мурастоо учурунда, бирдей аталыштагы методду чакыруунун приоритетин кайсы класс аныктайт?
а. Методду эң акыркы аныктаган класс.
б. Подклассты аныктоодо кашаанын ичинде биринчи көрсөтүлгөн класс.
с. Эң эски класс.
д. Подклассты аныктоодо кашаанын ичинде акыркы көрсөтүлгөн класс.
Жооп: б

9. Python көптүк мурастоо иерархиясында методдорду жана атрибуттарды издеген тартип кандай аталат?
а. Издөө методу (MSP) б. Мурастоону чечүү буйругу (IRO)
с. Методдорду чечүү тартиби (MRO) д. Иерархиялык издөө жолу (HSP)
Жооп: с

10. Эгерде WorkingStudent классы өзүнүн __init__ конструкторун аныктабаса, бирок конструкторлору бар Employee жана Student класстарынан мурастаса, эмне болот?

```
class WorkingStudent(Employee, Student):
    pass
tom = WorkingStudent("Tom")
```

а. Student конструктору чакырылат. б. Employee конструктору чакырылат.

с. Эки класстын тең конструкторлору чакырылат.
д. Конструктор аныкталбагандыктан ката болот.
Жооп: б
11. Эгерде подкласс өзүнүн атрибуттарын жана методдорун аныктабаса, анын денесинин ичинде эмне колдонуу керек?
а. pass оператору б. Break с. None д. return
Жооп: а
12. A -> B -> C иерархиясында, мында C B дан, ал эми B A дан мурастайт, C үчүн A классы кандай класс болуп саналат?
а. Подкласс б. Түздөн-түз суперкласс
с. Түздөн-түз эмес (кыйыр) суперкласс д. Туунду класс
Жооп: с
13. Мурастоо контекстинде суперкласстын жабык атрибуттары жөнүндө кайсы ырастоо ТУУРА ЭМЕС?
а. Алар подкласстын методдорунда түздөн-түз колдонулбайт.
б. Алар мурасталат, бирок аты өзгөртүлгөн (name mangling) түрүндө.
с. Алар суперкласстын жалпыга ачык геттер-методдору аркылуу жеткиликтүү болушу мүмкүн.
д. Алар подкласста толугу менен жоголот.
Жооп: д

5. Тема: Базалык Класстын Функционалын кайра аныктоо (Переопределение функционала базового класса**)**

План:

1. Функционалды кайра аныктоо (методду кайра аныктоо)
2. Объекттин тибин текшерүү (isinstance()

Максаты

Базалык класстын функционалын (конструкторлорду жана методдорду) super() функциясын колдонуу менен туунду класста кайра аныктоо ыкмаларын жалпылоо жана структуралаштыруу, ошондой эле Python'догу мурастоо иерархиясынын контекстинде объекттин тибин текшерүү үчүн орнотулган isinstance() функциясын кантип колдонууну түшүндүрүү.

3. Функционалды кайра аныктоо (методду кайра аныктоо)

Мурунку темаларыбызда Employee классы Person классынын функционалын толугу менен кабыл алган:

```
class Person:
    # ... __init__, name property, display_info методдору

class Employee(Person):
    # ... work() методу
```

Бирок, эгерде биз ошол функционалдын кайсы бир бөлүгүн өзгөртүүнү кааласакчы? Мисалы, кызматкерге конструктор аркылуу ал иштеген компанияны сактоочу жаңы

атрибут кошууну же display_info методунун ишке ашырылышын өзгөртүүнү кааласак. Python базалык класстын функционалын кайра аныктоого (переопределение) мүмкүндүк берет.

Конструкторду кайра аныктоо (__init__)

Туунду класска жаңы атрибуттарды (мисалы, company) кошуу үчүн конструкторду кайра аныктоо зарыл.

Базалык класстын конструкторун милдеттүү түрдө чакыруу: Эгер базалык класста __init__ методу аркылуу конструктор аныкталса жана биз туунду класста конструктордун логикасын өзгөртүүнү кааласак, туунду класстын конструкторунда базалык класстын конструкторун чакырышыбыз керек.

Базалык класска кайрылуу үчүн super() туюнтмасы колдонулат.

Employee классынын конструкторунда биз төмөнкүдөй чакырууну аткарабыз:

super().__init__(name) # Person классынын конструкторун чакыруу

Бул туюнтма Person классынын конструкторун чакырууну билдирет, ага кызматкердин аты (name) берилет. Бул логикалуу, анткени кызматкердин аты так ошол Person классынын конструкторунда орнотулат. Employee конструкторунда биз company атрибутун гана орнотобуз.

Жөнөкөй Методду кайра аныктоо (display_info)

Кайра аныктоо унасталган методдун логикасын өзгөртүүгө мүмкүндүк берет.

super().метод() колдонуу: Эгерде кайра аныкталган методдун бир бөлүгү базалык класстын методу менен дал келсе (мисалы, атты чыгаруу), кодду кайталабоо үчүн биз super() туюнтмасы аркылуу display_info методунун Person классындагы ишке ашырылышына кайрылабыз:

```
def display_info(self):
    super().display_info()     # Person классындагы display_info методуна кайрылуу
    print(f"Company: {self.company}") # Жаңы логиканы кошуу
```

Жыйынтыктоочу код мисалы

```
class Person:

    def __init__(self, name):
        self.__name = name
    # ... name property

    def display_info(self):
        print(f"Name: {self.__name}")

class Employee(Person):

    def __init__(self, name, company):
        # 1. super() аркылуу базалык конструкторду чакырабыз
        super().__init__(name)
        # 2. Жаңы атрибутту орнотобуз
        self.company = comp any

    def display_info(self):
        # 3. super() аркылуу базалык методду чакырабыз
        super().display_info()
        # 4. Жаңы маалыматты кошуп чыгарабыз
        print(f"Company: {self.company}")
```

```
    def work(self):
        print(f"{self.name} works")

tom = Employee("Tom", "Microsoft")
tom.display_info()

# Консольдогу жыйынтык:
# Name: Tom
# Company: Microsoft
```

2. Объекттин тибин текшерүү (isinstance()

Объекттер менен иштөөдө, алардын тибине жараша тигил же бул операцияларды аткаруу зарылчылыгы келип чыгат. Орнотулган isinstance() функциясы аркылуу биз объекттин тибин текшере алабыз.
Синтаксиси:

```
isinstance(object, type)
```

- Биринчи параметр – текшерилип жаткан объект.
- Экинчи параметр – текшерилүүчү тип (класс).

Эгерде объект көрсөтүлгөн типти билдирсе, функция True кайтарат.
Мисалы, Person – Employee/Student класстарынын иерархиясын алалы:

```
class Person:
    # ... __init__, name property, do_nothing()

class Employee(Person):
    def work(self):
        print(f"{self.name} works")

class Student(Person):
    def study(self):
        print(f"{self.name} studies")

def act(person):
    # Student тибин текшерүү
    if isinstance(person, Student):
        person.study()
    # Employee тибин текшерүү
    elif isinstance(person, Employee):
        person.work()
    # Person тибин текшерүү
    elif isinstance(person, Person):
        person.do_nothing()

tom = Employee("Tom")
bob = Student("Bob")
sam = Person("Sam")

act(tom)    # Tom works
act(bob)    # Bob studies
act(sam)    # Sam does nothing
```

Бул жерде act функциясы isinstance функциясынын жардамы менен person параметринин тибин текшерет жана текшерүүнүн жыйынтыгына жараша объекттин тиешелүү методун чакырат.

Тест

1. Методду кайра аныктоо (Method Overriding) деген эмне?
а. Суперкласста жок болгон жаңы методду подкласста түзүү.
б. Суперкласстын методдун атын подкласста өзгөртүү.
с. Подкласста суперкласстын методдун функционалын алмаштыруу же кеңейтүү.
д. Суперкласстын методун подкласстан чакыруу.
Жооп: с

2. Подкласстын конструкторунда super().__init__(аргументтер) чакыруусунун негизги максаты эмне?
а. Суперкласстын бардык атрибуттарын кайра аныктоо.
б. Подкласс үчүн уникалдуу болгон жаңы атрибуттарды орнотуу.
с. Мурасталып алынган атрибуттардын суперкласс тарабынан туура инициализацияланышын камсыз кылуу.
д. Базалык класстын конструкторунун чакырылышына бөгөт коюу.
Жооп: с

3. Эгерде Employee классында (Person классынан мурасталган) биз __init__ конструкторун кайра аныктасак, бирок Person конструкторун чакырбасак, эмне болот?
а. Employee атрибуттары гана инициализацияланат.
б. Person атрибуттары гана инициализацияланат.
с. Синтаксистик ката пайда болот.
д. Python автоматтык түрдө Person конструкторун чакырбайт.
Жооп: а (Атрибуттары гана инициализацияланат, бирок Person классында аныкталган name сыяктуу атрибуттар инициализацияланбайт, бул катага алып келиши мүмкүн.)

4. Employee классынын кайра аныкталган display_info методунда super().display_info() туюнтмасы эмне үчүн колдонулат?
а. Ал каалаган башка класстан методду чакырат.
б. Ал MRO боюнча эң жакын ата-тегинен (Person) методду чакырат.
с. Ал Employee классындагы display_info методунун ишин аяктайт.
д. Ал методдун атын өзгөртүүгө мүмкүндүк берет.
Жооп: б

5. Эгерде подкласста методду кайра аныктоодо super() колдонбосоңуз, эмне болот?
а. Суперкласстын методдун ишке ашырылышы подкласстын ишке ашырылышынан мурун аткарылат.
б. Суперкласстын методдун ишке ашырылышы подкласстын ишке ашырылышынан кийин аткарылат.
с. Суперкласстын методдун ишке ашырылышы подкласстын ишке ашырылышы менен толугу менен алмаштырылат.
д. Ката пайда болот, анткени super() милдеттүү.
Жооп: с

6. display_info сыяктуу методдорду кайра аныктоодо super().method() колдонуу кайсы көйгөйдү чечет?
а. Инкапсуляцияны камсыз кылат. б. Синтаксисти жөнөкөйлөтөт.
с. Суперкласстагы коддун кайталанышынан сактайт.
д. Программанын аткарылышын тездетет.
Жооп: с

7. Төмөндөгү туюнтмалардын кайсынысы isinstance() функциясынын максатын туура сүрөттөйт?
а. Объект белгилүү бир маалымат түрүбү (мисалы, str, int) деп текшерет.
б. Объекттин класс атын кайтарат.
с. Объект көрсөтүлгөн класстын же анын подклассынын инстанциясыбы деп текшерет.
д. Класс көрсөтүлгөн модулга таандыкпы деп текшерет.
Жооп: с
8. Эмне үчүн act функциясында Student жана Employee класстарын текшерүү Person классын текшерүүдөн мурун жүрүшү керек?
а. Бул маанилүү эмес, иреттин кереги жок.
б. Бул катанын алдын алат.
с. Анткени Student жана Employee дагы Person классынын инстанциялары болуп саналат, isinstance(person, Person) текшерүүсү биринчи иштеп, адистештирилген метод (work же study) чакырылбай калат.
д. Бул Python синтаксисинин талабы.
Жооп: с
9. Эгерде Person классында __init__ методу аныкталбаса, ал эми Employee(Person) классында ал аныкталса, super().__init__() чакыруу керекпи?
а. Ооба, дайыма, ар кандай учурга карата.
б. Жок, анткени Person классында чакыруу үчүн конструктор жок.
с. Ооба, бирок Employee классында өзүнүн атрибуттары жок болсо гана.
д. Бул суперкласста метод жок болгондуктан катага алып келет.
Жооп: б
10. super() оператору методду чакырбастан өзү эмне кылат?
а. Базалык класстын инстанциясын кайтарат.
б. Базалык класстын конструкторун чакырат.
с. MROдогу тиешелүү класска метод чакырууларды өткөрүп берүүчү прокси-объектти кайтарат. д. Ката чакырат.
Жооп: с
11. Төмөнкү код эмнени чыгарат?

```
class Parent:
    def greet(self):
        print("Hello")

class Child(Parent):
    def greet(self):
        super().greet()
        print("World")

c = Child()
c.greet()
```

а. World б. Hello с. Hello World д. Hello World
Жооп: с (Биринчи super().greet() аркылуу "Hello", андан кийин Child классынан "World" чыгарат).

12. Эгерде Person классы display_info методун аныктабаса, ал эми Employee (Person классынан мурастаган) аны super().display_info() колдонуу менен кайра аныктаса, эмне болот?
а. MROдо кийинки турган класстан метод чакырылат.
б. Демейки ишке ашыруу чакырылат.
с. AttributeError же NameError катасы пайда болот, анткени super() методду таба албайт.

д. Метод чакырылат, бирок эч нерсе чыгарбайт.
Жооп: с

13. Кайсы функция бир класс экинчи класстын подклассы экендигин текшерүүгө мүмкүндүк берет?
a. issubclass(Subclass, Superclass) б. super()
c. issubclass() д. type()
Жооп: a

6. Тема: Класстын атрибуттары жана статикалык методдор

План:

1. Класстын Атрибуттары (Class Attributes). Аныктамасы, Синтаксиси, Жетүү жана Өзгөртүү.
2. Аталыштардын карама каршылыгы (класстын атрибуту vs. объекттин атрибуту).
3. Статикалык методдор (Static Methods).

Максаты

Python тилиндеги класстын атрибуттары (бардык объекттер үчүн жалпы) жана статикалык методдор (объекттин абалына көз карандысыз) концепцияларын, анын ичинде аларды аныктоону, колдонууну, жетүү эрежелерин жана инстанциянын атрибуттары менен өз ара аракеттенүүсүн жалпылоо жана структуралаштыруу.

1. Класстын атрибуттары

Объекттин атрибуттарынан тышкары, класста класстын атрибуттарын да аныктоого болот. Мындай атрибуттар класстын деңгээлиндеги өзгөрмөлөр түрүндө аныкталат.
Синтаксис жана Жетүү:

```
class Person:
    type = "Person"
    description = "Describes a person"

print(Person.type)        # Person
print(Person.description)  # Describes a person

# Класстын атрибутун өзгөртүү:
Person.type = "Class Person"
print(Person.type)        # Class Person
```

Бул жерде Person классында эки атрибут аныкталган: type (класстын атын сактайт) жана description (класстын сүрөттөмөсүн сактайт).
Класстын атрибуттарына кайрылуу үчүн класстын атын колдонобуз (мисалы, Person.type). Объекттин атрибуттары сыяктуу эле, биз алардын маанилерин ала алабыз жана өзгөртө алабыз.
Бардык Объекттер үчүн Жалпы:
Класстын атрибуттары класстын бардык объекттери үчүн жалпы болуп саналат:

```
class Person:
    type = "Person"
```

```
    def __init__(self, name):
        self.name = name

tom = Person("Tom")
bob = Person("Bob")

print(tom.type)    # Person
print(bob.type)    # Person

# Класстын атрибутун өзгөртүү
Person.type = "Class Person"

print(tom.type)    # Class Person (том объектиси дагы өзгөртүүнү кабыл алды)
print(bob.type)    # Class Person (боб объектиси дагы өзгөртүүнү кабыл алды)
```

Класстын атрибуттары бардык объекттер үчүн кээ бир жалпы маалыматтарды аныктоо керек болгондо колдонулат.

Методдордун Ичинде Колдонуу Мисалы:

Python

```
class Person:
    default_name = "Undefined"

    def __init__(self, name):
        if name:
            self.name = name
        else:
            # Методдун ичинде класстын аты аркылуу кайрылуу
            self.name = Person.default_name

tom = Person("Tom")
bob = Person("")

print(tom.name)  # Tom
print(bob.name)  # Undefined
```

Бул мисалда default_name атрибуту демейки атты сактайт. Эгер конструкторго ат үчүн бош сап берилсе, name атрибутуна класстын default_name атрибутунун мааниси ыйгарылат.

2. Аталыштардын карама каршылыгы (Класстын Атрибуту vs. Объекттин Атрибуту)

Класстын атрибутунун жана объекттин атрибутунун аттары дал келген учур болушу мүмкүн. Эгерде коддо объекттин атрибуту үчүн маани берилбесе, анда ал үчүн класстын атрибутунун мааниси колдонулушу мүмкүн:

```
class Person:
    name = "Undefined" # Класстын атрибуту

    def print_name(self):
        print(self.name) # self.name объекттин атрибутун издейт

tom = Person()
bob = Person()
```

tom.print_name() # Undefined (Объекттин атрибуту жок, класстын атрибутун колдонот)
bob.print_name() # Undefined

 # Объекттин атрибутун орнотуу:
bob.name = "Bob"
bob.print_name() # Bob (Эми объекттин өзүнүн name атрибуту колдонулат)
tom.print_name() # Undefined (Tom дагы эле класстын атрибутун колдонууда)

Өзгөртүүлөрдүн Иерархиясы:
Эгерде биз класстын атрибутун өзгөртсөк, объекттин өзүнүн атрибуту бар объектилерге таасир этпейт, бирок класстын атрибутун колдонгон объекттер өзгөртүүнү кабыл алат:
Python
tom = Person()
bob = Person()
# ...

Person.name = "Some Person" # Класстын атрибутунун маанисин өзгөртүү

bob.name = "Bob" # Объекттин атрибутун орнотуу (bob үчүн класстын атрибуту бөгөттөлөт)

bob.print_name() # Bob (Объекттин өзүнүн атрибутун колдонот)
tom.print_name() # Some Person (Класстын өзгөртүлгөн атрибутун колдонот)

3. Статикалык методдор (@staticmethod)
Жөнөкөй методдордон тышкары, класс статикалык методдорду аныктай алат. Мындай методдор @staticmethod аннотациясы менен алдын ала белгиленет жана жалпысынан класска таандык.
Статикалык методдор адатта конкреттүү объектке көз каранды болбогон жүрүм-турумду аныктайт:

```
class Person:
    __type = "Person" # Жабык класстык атрибут

    @staticmethod
    def print_type():
        # Статикалык методдорго self параметри берилбейт
        # Класстын атрибутуна түз кайрылат
        print(Person.__type)

Person.print_type()  # Person - Класстын аты аркылуу статикалык методго кайрылуу

tom = Person()
tom.print_type()  # Person - Объекттин аты аркылуу статикалык методго кайрылуу
```

Бул мисалда:

- __type – класстын атын сактаган жабык класстык атрибут (кош астынкы сызыкча менен, ал жол берилбеген өзгөртүүлөрдөн коргойт).
- print_type – статикалык метод, ал __type атрибутунун маанисин чыгарат.

Бул методдун аракети конкреттүү объектке көз каранды эмес жана жалпы класска таандык, ошондуктан аны статикалык кылууга болот.

Тест

1. Person классында класстын атрибуту кайсы жерде аныкталган?

```
class Person:
   A = "Class"
   def __init__(self, name):
      self.B = name
```

a. B б. __init__ c. A д. self.name

Жооп: c

2. Person классынан тышкары type класстын атрибутуна туура жетүү жолу кайсы?

a. Person__type__ б. Person.type

c. Person.typetype д. Person().type

Жооп: б

3. Кайсы учурда объекттин атрибуту (self.name) бирдей аталыштагы класстын атрибутунун маанисин колдонот?

a. Объекттин атрибуту конструктордо инициализацияланганда.

б. Объекттин атрибуту инициализацияланбай калып, Python аны ата-теги классынан издегенде.

c. Объекттин атрибуту инициализацияланбай калып, Python аны класстын деңгээлинен издегенде.

д. Эгерде класстын атрибуту жабык деп жарыяланса гана.

Жооп: c

4. Класстын name атрибуту бар болгон учурда, эгерде bob.name = "Bob" деп орнотулса, эмне болот?

a. Класстын атрибутунун мааниси бардык объекттер үчүн өзгөрөт.

б. bob объектиси үчүн класстын атрибутун "көмүскөдө калтырган" жаңы инстанциялык атрибут түзүлөт.

c. Аталыштар дал келгендиктен ката пайда болот.

д. Person.name атрибуту өзгөрөт, бирок tom.name өзгөрбөйт.

Жооп: б

5. Класстын атрибуттары көбүнчө эмне үчүн колдонулат?

a. Ар бир объекттин уникалдуу, өзгөрүлүүчү маалыматтарын сактоо үчүн.

б. Объекттин абалына көз каранды болбогон операцияларды аткаруу үчүн.

c. Бардык объекттер үчүн жалпы маалыматтарды, константаларды же демейки маанилерди аныктоо үчүн.

д. Инкапсуляцияны камсыз кылуу үчүн.

Жооп: c

6. Статикалык методду аныктоо үчүн кайсы декоратор колдонулат?

a. @abstractmethod б. @property

c. @staticmethod д. @classmethod

Жооп: c

7. Төмөнкү код эмнени чыгарат?

Python

```
class Test:
   COUNT = 0
   @staticmethod
   def increment():
      Test.COUNT += 1

t = Test()
Test.increment()
Test.increment()
```

print(Test.COUNT)
а. 0 б. 1 с. 2 д. Ката, анткени статикалык метод класстын атрибуттарын өзгөртө албайт.
Жооп: с
8. Person классындагы print_type() методун (ал Person.__type чыгарат) статикалык кылуу эмне үчүн максатка ылайыктуу?
а. Ал объекттин абалын өзгөртөт.
б. Ал милдеттүү түрдө self параметрин кабыл алат.
с. Анын аракети объекттин конкреттүү инстанциясына көз каранды эмес жана дайыма класска жалпы бирдей маанини чыгарат.
д. Ал класстын атрибутуна жетүү мүмкүнчүлүгүнө ээ эмес.
Жооп: с
9. Статикалык метод print_type кантип чакырылышы мүмкүн?
а. Класстын аты аркылуу гана: Person.print_type()
б. Объект аркылуу гана: tom.print_type()
с. Класстын аты (Person.print_type()) жана объект аркылуу тең (tom.print_type())
д. Класстын башка методунун ичинен гана.
Жооп: с
10. Объекттин методунун (print_name) ичинде, эгерде биз default_name класстын атрибутуна кайрылууну кааласак (мисалы, аны колдонуу үчүн), муну кантип жасаган туура?
а. globals().default_name б. super().default_name
с. Person.default_name д. self.default_name
Жооп: с
11. Эгерде статикалык метод объекттин атрибутуна, мисалы, self.name ге жетүүгө аракет кылса, эмне болот?
а. Ал биринчи түзүлгөн объекттен маани алат.
б. Ал класстын атрибутунан маани алат.
с. Ката пайда болот, анткени статикалык методдун self параметрине жетүүсү жок.
д. Python аны автоматтык түрдө класс методуна айлантат.
Жооп: с
12. Статикалык методдун мисалындагы __type атрибуту эмне үчүн жабык класстын атрибуту деп аталган?
а. Анткени ага класстын ичинен да кайрылуу мүмкүн эмес.
б. Анткени ал кош астынкы сызыкча менен башталат (__), бул name mangling механизмин активдештирет.
с. Анткени ал дайыма статикалык методдордо колдонулушу керек.
д. Анткени ал конструктордон тышкары аныкталган.
Жооп: б
13. Эгерде класста инстанциянын атрибуттары жок болсо (б.а. __init__ жок же бош), бирок класстын атрибуттары бар болсо, анын объекттеринин ортосундагы негизги айырмачылык эмнеде?
а. Алардын маалыматтары бирдей, бирок методдору ар башка.
б. Алардын методдору бирдей, бирок маалыматтары ар башка (класстын атрибуттары).
с. Алар маалыматтар жана жүрүм-турум боюнча толугу менен окшош, анткени жалпы маалыматтарга жана статикалык жүрүм-турумга "жетүү чекиттери" гана болуп саналат.
д. Алардын эс тутум даректери ар башка болот, бирок атрибуттары бирдей.
Жооп: с

7. Тема Object классы. Объекттин саптык түрү

План:

1. Python класстары жөнүндө кыскача эскертүү.
2. __str__ методунун ролу.
3. Кайра аныктабастан стандарттык жүрүм-турум.
4. Колдонуучу классында __str__ методун кайра аныктоо.
5. Person классынын негизиндеги мисал.
6. __repr__ методу менен байланышы.

Максаты

Окуучуну Python'дун колдонуучу класстарында **__str__** методун туура кайра аныктоого, объекттердин оңой окулуучу саптык түрүн түзүүгө жана коддун сапатын жогорулатууга үйрөтүү.

Python программалоо тилинде 3-версиядан баштап, бардык класстар жашыруун түрдө бир жалпы суперкласска – object классына ээ жана бардык класстар демейки боюнча анын методдорун мурастайт.
object классынын эң көп колдонулган методдорунун бири – __str__() методу.
Объекттин саптык түрүн алууда же объектти сап түрүндө чыгаруу керек болгондо, Python дал ушул методду чакырат. Классты аныктоодо бул методду кайра аныктоо жакшы практика болуп саналат.
Мисалы, Person классын алып, анын саптык түрүн чыгаралы:

```
class Person:
    def __init__(self, name, age):
        self.name = name  # атты орнотобуз
        self.age = age    # жашты орнотобуз

    def display_info(self):
        print(f"Name: {self.name}  Age: {self.age}")

tom = Person("Tom", 23)
print(tom)
```

Программа иштегенде, ал төмөнкүдөй бир нерсени чыгарат:
<__main__.Person object at 0x10a63dc00>
Бул объект жөнүндө анча маалымат бербеген маалымат. Биз, албетте, Person классында объекттин маалыматтарын чыгаруучу кошумча методду (жогорудагы мисалда display_info) аныктоо менен бул кырдаалдан чыга алабыз.
Бирок дагы бир жолу бар – Person классында **__str__()** методун кайра аныктайбыз (эки астынкы сызыкча менен):

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age
```

```
  def __str__(self):
      return f"Name: {self.name}  Age: {self.age}"

tom = Person("Tom", 23)
print(tom)          # Name: Tom  Age: 23
```

__str__ методу **сапты кайтарышы** керек. Бул учурда биз адам жөнүндөгү негизги маалыматты кайтарып жатабыз.
Эми консольдогу жыйынтык башкача болот:
Name: Tom Age: 23
Hobby: sport

Тест

1. Python 3'тө бардык класстар кайсы суперкласстысы мурастайт?
a. base б. object c. parent д. super
Жооп: б
2. Объекттин саптык түрүн алуу үчүн кайсы метод колдонулат?
а. __repr__()
б. __init__()
с. __str__()
д. __print__()
Жооп: с
3. __str__() методу эмнени кайтарууга милдеттүү?
а. Каалаган объект
б. Тизме
с. Сан
д. Сап (String)
Жооп: д
4. Эгерде __str__() кайра аныкталбаса, эмне болот?
а. Объект бош сап менен чыгарылат.
б. Программа ката берет.
с. __init__() чакырылат.
д. Стандарттык <Class object at ...> түрүндөгү көрсөтүү чыгарылат.
Жооп: д
5. __str__() методу качан чакырылат?
а. Объект түзүлгөндө.
б. Сапка айландырууда (print, str функциялары аркылуу).
с. Объект өчүрүлгөндө.
д. Класс импорттолгондо.
Жооп: б
6. Ката оңдоо (отладка) үчүн көбүнчө кайсы метод колдонулат?
а. __del__()
б. __repr__()
с. __copy__()
д. __call__()
Жооп: б
7. Класстын ичинен __str__() методун кол менен кантип чакырса болот?
а. display()
б. print()
с. self.__str__()
д. object.str()
Жооп: с

8. __str__() жана __repr__() методдору ар кандай саптарды кайтара алабы?
а. Жок
б. Ооба
с. Python 2'де гана
д. Мурастоосуз класстарда гана
Жооп: б
9. Эгерде __str__() сан кайтарса, эмне болот?
а. Эч нерсе болбойт, бардыгы туура.
б. Объект автоматтык түрдө сапка айландырылат.
с. Нөл чыгарылат. д. TypeError катасы пайда болот.
Жооп: д
10. Төмөнкү код эмнени чыгарат?

```
class A:
   pass

a = A()
print(a)
```

а. Ката б. Бош сап
с. Объекттин стандарттык көрсөтүлүшү (<__main__.A object at ...>)
д. Класстын аты
Жооп: с

8. Тема: Операторлорду кайра жүктөө (Overloading) жана абстракттуу класстар
План:

1. Атайын (магиялык) метод түшүнүгү.
2. Операторлорду кайра жүктөө (Overloading).
3. Практикалык мисалдар.
4. Абстракттуу класстар.
5. Өзүбүздүн абстракттуу классты түзүү.
6. Абстракттуу класстарды интерфейстер менен салыштыруу.

Максаты
Python'до операторлорду кайра жүктөө механизмдерин жана абстракттуу класстар менен иштөөнү изилдөө, кеңири функционалдуулукка ээ өз класстарды түзүүнү үйрөнүү, атайын методдорду (__add__, __getitem__, __bool__, ж.б.) колдонууну, ошондой эле abc модулунун жардамы менен абстракттуу класстарды иштеп чыгуу жана колдонуу.

1. Операторлорду кайра жүктөө (Operator Overloading)
Python өз класстары үчүн кошуу, кемитүү сыяктуу орнотулган операторлорду аныктоого мүмкүндүк берет. Бул үчүн, класстарга атайын (магиялык) методдорду ишке ашыруу керек. Төмөндө кээ бир операторлор жана аларга туура келген атайын методдор келтирилген:

Операция	**Синтаксис**	**Функция**
Сложение	a + b	__add__(a, b)
Объединение	seq1 + seq2	__concat__(seq1, seq2)
Проверка наличия	obj in seq	__contains__(seq, obj)
Деление	a / b	__truediv__(a, b)
Деление	a // b	__floordiv__(a, b)
Поразрядное И	a & b	__and__(a, b)
Поразрядное XOR	a ^ b	__xor__(a, b)
Поразрядная инверсия	~ a	__invert__(a)
Поразрядное ИЛИ	a | b	__or__(a, b)
Степень	a ** b	__pow__(a, b)
Присвоение по индексу	obj[k] = v	__setitem__(obj, k, v)
Удаление по индексу	del obj[k]	__delitem__(obj, k)
Обращение по индексу	obj[k]	__getitem__(obj, k)
Сдвиг влево	a << b	__lshift__(a, b)
Остаток от деления	a % b	__mod__(a, b)
Умножение	a * b	__mul__(a, b)
Умножение матриц	a @ b	__matmul__(a, b)
Арифметическое отрицание	-a	__neg__(a)
Логическое отрицание	not a	__not__(a)
Положительное значение	+a	__pos__(a)
Сдвиг вправо	a >> b	__rshift__(a, b)
Установка диапазона	seq[i:j] = values	__setitem__(seq, slice(i, j), values)
Удаление диапазона	del seq[i:j]	__delitem__(seq, slice(i, j))
Получение диапазона	seq[i:j]	__getitem__(seq, slice(i, j))
Вычитание	a - b	__sub__(a, b)
Проверка на True/False	obj	__bool__(obj)
Меньше чем	a < b	__lt__(a, b)
Меньше чем или равно	a <= b	__le__(a, b)
Равенство	a==b	__eq__(a, b)
Неравенство	a != b	__ne__(a, b)
Больше чем или равно	a >= b	__ge__(a, b)
Больше чем	a > b	__gt__(a, b)
Сложение с присваиванием	a += b	__iadd__(a, b)
Объединение с присваиванием	a += b	__iconcat__(a, b)
Поразрядное умножение с присваиванием	a &= b	__iand__(a, b)
Деление с присваиванием	a //= b	__ifloordiv__(a, b)
Сдвиг влево с присваиванием	a <<= b	__ilshift__(a, b)
Сдвиг вправо с присваиванием	a >>= b	__irshift__(a, b)
Деление по модулю с присваиванием	a %= b	__imod__(a, b)
Умножение с присваиванием	a += b	__imul__(a, b)
Умножение матриц с присваиванием	a @= b	__imatmul__(a, b)

Поразрядное сложение с присваиванием	a |= b	__ior__(a, b)
Возведение в степень с присваиванием	a **= b	__ipow__(a, b)
Вычитание с присваиванием	a -= b	__isub__(a, b)
Деление с присваиванием	a /= b	__itruediv__(a, b)
Операция XOR с присваиванием	a ^= b	__ixor__(a, b)

Практикалык Мисалдар
Операторду класска аныктоо үчүн, класстын ичинде тиешелүү атайын методду ишке ашыруу керек.

Кошуу Операторун кайра жүктөө (__add__)

Биз эки Counter объектисин кошууну каалайбыз, анда алардын value атрибуттары кошулуп, жаңы Counter объектиси кайтарылат:

```
class Counter:
    def __init__(self, value):
        self.value = value

    # Кошуу операторун кайра аныктоо
    def __add__(self, other):
        # other экинчи Counter объектин билдирет
        return Counter(self.value + other.value)

counter1 = Counter(5)
counter2 = Counter(15)
counter3 = counter1 + counter2
print(counter3.value)  # 20
```

__add__ методунун экинчи параметри (other) башка бир типти да билдириши мүмкүн. Мисалы, Counter объектисин санга кошуу:

```
class Counter:
    # ... __init__

    def __add__(self, other):
        # other эми сан болуп эсептелет
        return Counter(self.value + other)

counter1 = Counter(5)
counter3 = counter1 + 6
print(counter3.value)  # 11
```

Объекттин Чындыгын Аныктоо (__bool__)

__bool__ функциясын аныктоо объекттин чындыгын орнотууга (True/False маанилерине айландырууга) мүмкүндүк берет:

```
class Counter:
    def __init__(self, value):
        self.value = value

    def __bool__(self):
```

```
        # value 0дон чоң болсо True кайтарат
        return self.value > 0

counter1 = Counter(3)
counter2 = Counter(-3)

if counter1:
    print("Counter1 = True") # Counter1 = True

if counter2:
    print("Counter2 = True")
else:
    print("Counter2 = False") # Counter2 = False
```

Индекс Боюнча Жетүү Операциялары (__getitem__)

__getitem__ функциясы объектке квадрат кашаалар аркылуу (obj[index]) кайрылууну ишке ашырууга мүмкүндүк берет:

```
class Person:
    def __init__(self, name, age):
        self.__name = name
        self.__age = age

    def __getitem__(self, prop):
        # prop экинчи параметр - индекс (бул учурда атрибуттун аты)
        if prop == "name": return self.__name
        elif prop == "age": return self.__age
        return None

tom = Person("Tom", 39)

print("Name:", tom["name"])  # Name: Tom
print("Age:", tom["age"])    # Age: 39
```

Касиеттин Бар-жоктугун Текшерүү (__contains__)

in операторунун ишке ашырылышы үчүн __contains__() функциясы жооп берет:

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def __contains__(self, prop):
        # prop - текшерилип жаткан маани (мисалы, "name")
        if prop == "name" or prop == "age": return True
        return False

tom = Person("Tom", 39)

print("name" in tom)  # True
print("id" in tom)    # False
```

Операторлорду жуп-жубу менен ишке ашыруу

Салыштыруу операторлорун (мисалы, == жана !=) кайталанууну болтурбоо үчүн бирөөнү экинчиси аркылуу ишке ашыруу ыңгайлуу:

```
class Counter:
    def __init__(self, value):
        self.value = value

    def __eq__(self, other): return self.value == other.value
    def __ne__(self, other): return not (self == other) # != операторун == аркылуу ишке ашыруу

    def __gt__(self, other): return self.value > other.value
    def __le__(self, other): return not (self > other) # <= операторун > аркылуу ишке ашыруу

c1 = Counter(1)
c2 = Counter(2)

print(c1 == c2)  # False
print(c1 != c2)  # True
```

2. Абстракттуу Класстар жана Методдор

Кээде бизде "жаныбар" же "геометриялык фигура" сыяктуу конкреттүү ишке ашырылышы жок болгон түшүнүктөр менен иштөө керек болот. Мындай түшүнүктөрдү сүрөттөө үчүн абстракттуу класстар колдонулат.

Абстракттуу классты түзүү

Python'до абстракттуу класстарды түзүү үчүн бардык куралдар атайын abc модулунда аныкталган:

```
import abc
```

Негизги компоненттери:

ABC классы: Абстракттуу классты түзүүнү жеңилдетет (бардык абстракттуу класстар аны мурастайт).

@abstractmethod аннотациясы: Абстракттуу методду түзүү үчүн колдонулат.

Абстракттуу Класс Мисалы (Shape)

```
import abc

# Shape классы ABC классынан мурастайт
class Shape(abc.ABC):

    @abc.abstractmethod
    def area(self):
        # Абстракттуу методдун ишке ашырылышы жок, pass оператору колдонулат
        pass
```

Абстракттуу класстын абстракттуу методдору бар болсо, биз анын конструкторун колдонуп, түздөн-түз объект түзө албайбыз.

Класс-Мураскорлордун Милдети

Класс-мураскорлор абстракттуу класстын бардык абстракттуу методдорун ишке ашырууга милдеттүү.

Rectangle (Тик бурчтук) классынын мисалы:

```
# ... Shape абстракттуу классы жогоруда аныкталган ...
```

```
class Rectangle(Shape):
    def __init__(self, width, height):
        self.width = width
        self.height = height

    # Абстракттуу методду ишке ашыруу МИЛДЕТТҮҮ
    def area(self):
        return self.width * self.height

rect = Rectangle(30, 50)
print("Rectangle area:", rect.area()) # Rectangle area: 1500
```

Абстракттуу Класстагы Конструкторлор жана Кадимки Методдор

Абстракттуу класстар ошондой эле конструкторлорду, атрибуттарды жана абстракттуу эмес методдорду да аныктай алат, алар мураскор класстарда колдонулушу мүмкүн:

```
import abc

class Shape(abc.ABC):
    def __init__(self, x, y):
        self.x = x
        self.y = y

    @abc.abstractmethod
    def area(self): pass

    def print_point(self): # Абстракттуу эмес метод
        print("X:", self.x, "\tY:", self.y)

class Rectangle(Shape):
    def __init__(self, x, y, width, height):
        # super() аркылуу абстракттуу класстын конструкторун чакыруу
        super().__init__(x, y)
        self.width = width
        self.height = height

    def area(self): return self.width * self.height

rect = Rectangle(10, 20, 100, 100)
rect.print_point()  # X: 10  Y: 20 (Shape классынан мурасталган метод)
```

Бул мисалда Shape абстракттуу классы фигура түзүлгөн чекиттин координаталарын (X, Y) кабыл алат жана аларды чыгаруучу print_point аттуу абстракттуу эмес методго ээ.

Тест

1. Кошуу операторун кайра жүктөөгө кайсы метод жооп берет: +?

А. __sum__ Б. __plus__ В. __add__ Г. __iadd__

Жооп: В

2. += операторун кайсы метод ишке ашырат?

А. __add__ Б. __sum__ В. __iadd__ Г. __append__

Жооп: В

3. Объектке индекс аркылуу кайрылганда кайсы метод чакырылат: obj[key]?

А. __get__() Б. __getitem__() В. __item__() Г. __index__()

Жооп: Б
4. in оператору аркылуу маанини текшерүүгө кайсы метод жооп берет?
А. __inside__()
Б. __contains__()
В. __in__()
Г. __has__()
Жооп: Б
5. __bool__() методу эмнени кайтарышы керек?
А. Каалаган объект Б. bool классынын объектиси (True/False)
В. Сан Г. Сап
Жооп: Б
6. Салыштыруу методу __gt__() кайсы натыйжаны кайтарышы керек?
А. Сан Б. True/False В. Сап Г. Класс объектиси
Жооп: Б
7. == операторун кайсы метод кайра жүктөйт?
А. __eq__() Б. __equal__() В. __compare__() Г. __same__()
Жооп: А
8. != операторуна кайсы метод жооп берет?
А. __neq__() Б. __notequal__() В. __ne__() Г. __not__()
Жооп: В
9. Абстракттуу класстарды түзүү үчүн кайсы модуль колдонулат?
А. abstract Б. abc В. abs Г. classbase
Жооп: Б
10. Абстракттуу методду кантип белгилөө керек?
А. @method.abstract Б. @abstract
В. @abc.abstractmethod Г. @abs.method
Жооп: В
11. Абстракттуу методдору бар абстракттуу класстын инстанциясын (объектин) түзүүгө болобу?
А. Ооба
Б. Ооба, эгер конструктор болсо
В. Жок
Г. Эгерде super() чакырылса гана
Жооп: В
12. - операторуна кайсы метод жооп берет?
А. __minus__() Б. __remove__() В. __sub__() Г. __neg__()
Жооп: В
13. Унардык минус үчүн, мисалы, -obj үчүн кайсы метод жооп берет?
А. __sub__() Б. __minus__() В. __neg__() Г. __invert__()
Жооп: В
14. Логикалык терс мааниге (not obj) кайсы метод жооп берет?
А. __not__() Б. __bool__() В. __invert__() Г. __false__()
Жооп: А
15. Индекс боюнча элементти өчүрүү үчүн кайсы метод колдонулат: del obj[key]?
А. __removeitem__() Б. __del__() В. __delitem__() Г. __delete__()
Жооп: В
16. Даражага көтөрүү операторуна (**) кайсы метод жооп берет?
А. __power__() Б. __pow__() В. __exp__() Г. __mul__()

Жооп: Б

17. >= салыштыруусу үчүн кайсы метод колдонулат?

А. __ge__() Б. __gte__() В. __eg__() Г. __moreequal__()

Жооп: А

18. @ операторун (матрицаларды көбөйтүү) кайсы метод кайра жүктөйт?

А. __mat__() Б. __matrix__() В. __mul__() Г. __matmul__()

Жооп: Г

19. print(obj) учурунда объекттин саптык түрүнө кайсы метод жооп берет?

А. __repr__() Б. __str__() В. __print__() Г. __show__()

Жооп: Б

20. Объектти int тибине айландыруу аракетинде кайсы метод чакырылат?

А. __int__() Б. __toint__() В. __convert__() Г. __num__()

Жооп: А

Колдонулган адабияттардын тизмеси

1. Объектно-ориентированное программирование. В 3-х частях. Ч.1: учебное пособие / П.П. Степанов, А.А. Кабанов, В.А. Никонов, Т.С. Павлюченко. – Омск: Омский государственный технический университет, 2021. – 112 с. – Режим доступа: https://www.iprbookshop.ru/124850.html (ЭБС «IPRbooks»).
2. Букунов, С.В. Объектно-ориентированное программирование на языке Python: учебное пособие / С.В. Букунов, О.В. Букунова. – Санкт-Петербург: Санкт-Петербургский государственный архитектурно-строительный университет, ЭБС АСВ, 2020. – 119 с. – Режим доступа: https://www.iprbookshop.ru/117194.html (ЭБС «IPRbooks»).
3. Зыков, С.В. Введение в теорию программирования. Объектно-ориентированный подход: учебное пособие / С.В. Зыков. – 3-е изд. – Москва: Интернет-Университет Информационных Технологий (ИНТУИТ), Ай Пи Ар Медиа, 2021. – 187 с. – Режим доступа: https://www.iprbookshop.ru/102007.html (ЭБС «IPRbooks»).
4. Щерба, А.В. Программирование на Python: первые шаги / А.В. Щерба. – Москва: Лаборатория знаний, 2022. – 251 с. – Режим доступа: https://www.iprbookshop.ru/120878.html (ЭБС «IPRbooks»).
5. 1. Ашарина, И.В. Объектно-ориентированное программирование в C++: лекции и упражнения: Учебное пособие для вузов / И.В. Ашарина. - М.: РиС, 2015. - 336 с.
   2. Ашарина, И.В. Язык C++ и объектно-ориентированное программирование в C++. Лабораторный практикум: Учебное пособие для вузов / И.В. Ашарина, Ж.Ф. Крупская. - М.: ГЛТ , 2015. - 232 с.
   3. Ашарина, И.В. Язык C++ и объектно-ориентированное программирование в C++. Лабораторный практикум: Учебное пособие / И.В. Ашарина, Ж.Ф. Крупская. - М.: ГЛТ, 2015. - 232 с.
   4. Белов, В.В. Программирование в Delphi: процедурное, объектно-ориентированное, визуальное: Учебное пособие / В.В. Белов. - М.: ГЛТ, 2009. - 240 с.
   5. Белов, В.В. Программирование в Delphi: процедурное, объектно-ориентированное, визуальное: Учебное пособие для вузов / В.В. Белов, В.И. Чистякова. - М.: РиС, 2014. - 240 с.
   6. Белов, В.В. Программирование в Delphi: процедурное, объектно-ориентированное, визуальное: Учебное пособие для ву / В.В. Белов, В.И. Чистякова. - М.: ГЛТ, 2009. - 240 с.
   7. Васильев, А. C#. Объектно-ориентированное программирование: Учебный курс / А.

Васильев. - СПб.: Питер, 2012. - 320 с.
8. Васильев, А. C#. Объектно-ориентированное программирование. Учебный курс / А. Васильев. - СПб.: Питер, 2012. - 320 с.
9. Васильев, А. Java. Объектно-ориентированное программирование: Учебное пособиеСтандарт третьего поколения / А. Васильев. - СПб.: Питер, 2013. - 400 с.
10. Васильев, А.Н. Объектно-ориентированное программирование на C++ / А.Н. Васильев. - СПб.: Наука и техника, 2016. - 544 с.

Колдонулган интернет ресурстары

1. METANIT.COM - Сайт о программировании
2. Лабораторные работы «Язык программирования Python». – Режим доступа: http://agpu.net/fakult/ipimif/fpiit/kafinf/MetodicheskoyeObespecheniye/KPP_CHernyshov.pdf
3. Задорожный С.С., Фадеев Е.П. Объектно-ориентированное программирование на языке Python. – М.: Физический факультет МГУ им. М. В. Ломоносова, 2022. – 40 с.