ЖИТОМИРСЬКА ПОЛІТЕХНІКА
ДЕРЖАВНИЙ УНІВЕРСИТЕТ

@ZTUEDUUA

АЛГОРИТМІЧНЕ ЗАБЕЗПЕЧЕННЯ КОМП'ЮТЕРИЗОВАНИХ ІНФОРМАЦІЙНО-ВИМІРЮВАЛЬНИХ СИСТЕМ

Лекція 7
Тема: Функція

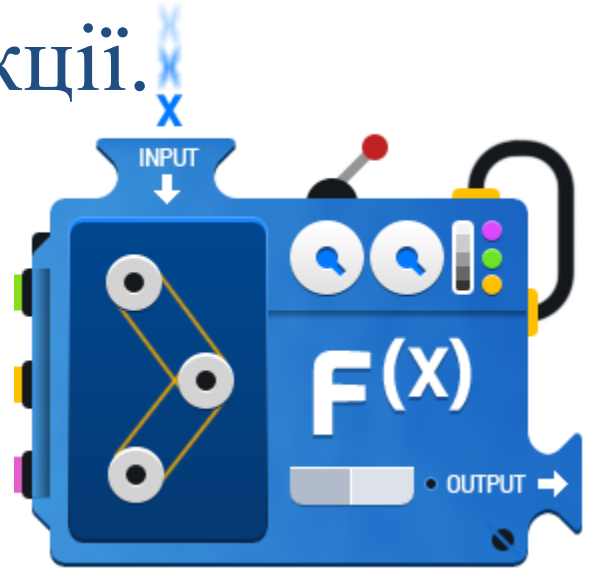
1.Функції.

2.Передача параметрів.

3.Упаковка і розпакування параметрів функції.

4.Функція як змінна.

5.Рекурсія.



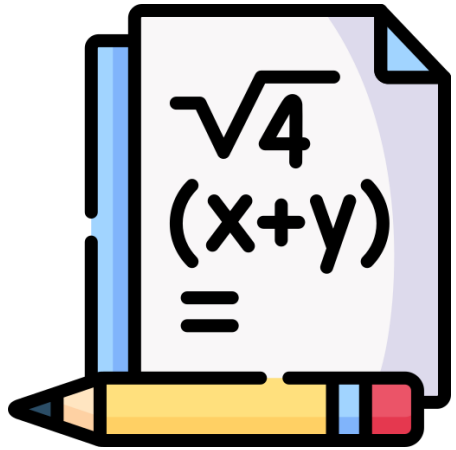
1. Функція – це фрагмент коду, призначений для розв'язання певної задачі, до якого можна багаторазово звертатись з різних місць основної програми. Вона повертає у місце виклику певне значення, що є результатом її виконання

У Python існує безліч вбудованих функцій, але основні функції можна узагальнити в кілька категорій. Ось декілька основних функцій Python:

1. Функції для операцій з числами
2. Функції для роботи з рядками
3. Функції для роботи зі списками
4. Функції для роботи з файлами

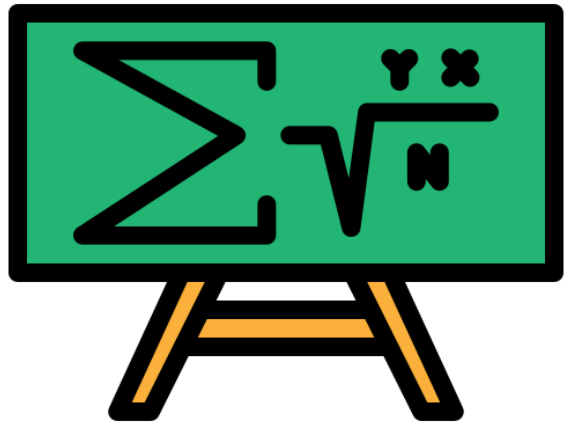
Функції для операцій з числами

Існують ці функції для виконання різних операцій та маніпуляцій з числовими значеннями. Наприклад:



1) ***математичні операції*** дозволяють виконувати стандатрні дії, такі як додавання, віднімання, множення та ділення. Наприклад, функція ***abs()*** використовується для визначення абсолютного значення числа, а ***round()*** заокруглює число.

2) ***Перетворення типів даних:*** Іноді потрібно перетворити числа з одного типу даних в інший. Функції, такі як ***int()*** та ***float()***, дозволяють це робити.



3) ***Знаходження максимуму та мінімуму***: Функції ***max()*** та ***min()*** допомагають знаходити найбільше і найменше значення серед чисел. Це корисно, наприклад, при аналізі даних або визначенні екстремальних значень у наборі даних.

4) ***Обробка помилок***: функції для операцій з числами також можуть бути корисні для перевірки та обробки помилок. Наприклад, при діленні на нуль функція ***divmod()*** повертає відповідне значення та залишок, але генерує виключення в разі ділення на нуль, яке можна обробити.

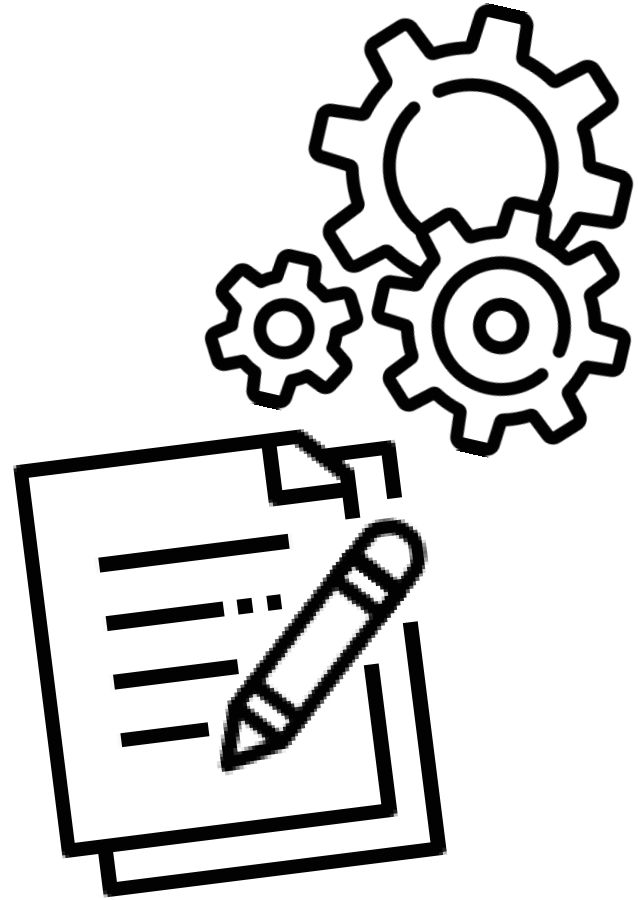
Функції для роботи з рядками, списками і даними

Функції для роботи з рядками в Python існують для обробки та маніпуляції з рядками, що є фундаментальним типом даних в багатьох програмах. Функції для роботи з рядками дозволяють вам виконувати різні операції з текстом, такі як знаходження, видалення та заміна підрядків у рядках. Ви можете форматувати рядки, додавати значення до них та виводити текст в зручному для читання вигляді. Наприклад у *Python* є можливість обчислити довжину рядків використавши функцію ***len()***, що може бути корисним при роботі з текстом.

***Зчитування та запис у файли*:** Функції для роботи з рядками часто використовуються для зчитування та запису тексту в файли. Це дозволяє працювати з текстовими файлами та взаємодіяти з ними.

Обробка відповідей користувача: Функції для роботи з рядками можуть бути корисними при обробці введення користувача. Наприклад, ви можете використовувати ***input()*** для отримання рядкового введення від користувача та обробляти це введення для подальших операцій.

Загалом, функції для роботи з рядками грають ключову роль у багатьох аспектах програмування, включаючи обробку текстової інформації, роботу зі структурованими даними, аналіз даних та взаємодію з користувачем.



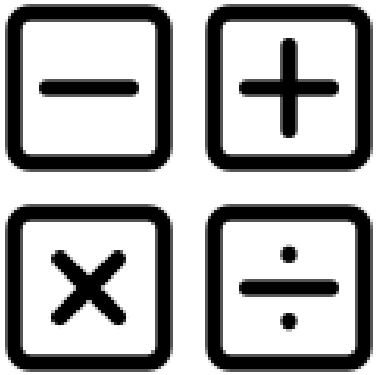
Функції для роботи з файлами

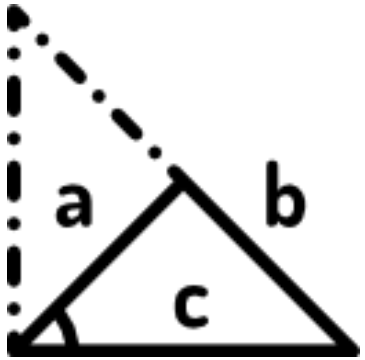
Функції для роботи з файлами в Python існують для обробки, читання, запису та керування файлами. Робота з файлами важлива для збереження та обробки даних.

1. **Читання файлів**: Ви можете використовувати функцію ***open()*** для відкриття файлів для читання. Потім ви можете використовувати функції, такі як ***read(), readline(), readlines(),*** для зчитування вмісту файлу в рядок або список рядків.
2. **Запис файлів**: Функція ***open()*** також використовується для відкриття файлів для запису. Ви можете використовувати функції, такі як ***write()***, для запису даних у файл. Запис може бути виконаний повністю або частково, заміщенням або дописуванням до вже існуючого вмісту файлу.
3. **Робота з багатофайловими операціями**: Python дозволяє виконувати операції з кількома файлами одночасно. Ви можете відкривати, читати і записувати файли одночасно за допомогою функцій, таких як ***'with'***, для автоматичного закриття файлів після завершення роботи з ними.

Загалом, функції для роботи з файлами є важливими для збереження та обробки даних у програмах Python і дозволяють взаємодіяти з файловою системою оптимальним способом.

Перелік основних функцій



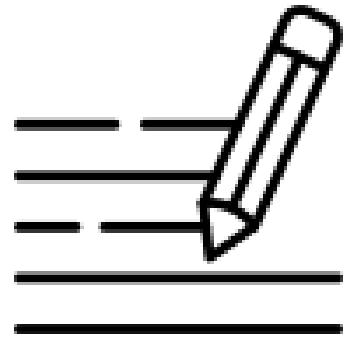


Функції для операцій з числами:

- *abs():* Повертає абсолютне значення числа.
- *round():* Округлює число до заданої кількості знаків після коми.
- *int():* Перетворює значення в ціле число.
- *float():* Перетворює значення в число з плаваючою комою.
- *max():* Повертає найбільше значення серед аргументів.
- *min():* Повертає найменше значення серед аргументів.

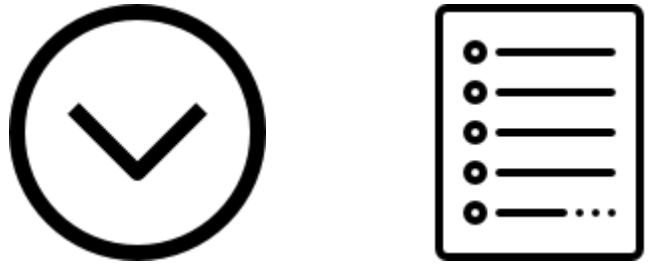
Функції для роботи з рядками:

- *len():* Повертає кількість символів у рядку.
- *str():* Перетворює значення в рядок.
- *split():* Розбиває рядок на підстроки за певним роздільником.
- *join():* Об'єднує список рядків в один рядок.
- *strip():* Видаляє пробіли і символи переносу рядка з початку і кінця рядку.
- *format():* Використовується для форматування рядків з підстановкою значень.



Функції для роботи з файлами:

- *open():* Відкриває файл для читання або запису.
- *read():* Зчитує вміст файлу.
- *write():* Записує дані у файл.
- *close():* Закриває файл.



Функції для роботи зі списками:

- *len():* Повертає кількість елементів у списку або кортежі.
- *append():* Додає елемент до кінця списку.
- *extend():* Розширює список іншим списком.
- *insert():* Додає елемент на задану позицію.
- *remove():* Видаляє перший зустрічний елемент зі списку.
- *sort():* Сортує список.

2.Передача параметрів

- До функції можуть передаватися аргументи. Аргументи – це змінювані чи незмінні об'єкти. У мові Python має значення, до якої категорії об'єктів (змінюваних чи незмінних) належить аргумент.
- Наприклад, до змінюваних об'єктів належать списки та словники. До незмінних об'єктів відносяться числа, рядки, кортежі.
- Якщо функцію передається незмінний об'єкт, цей об'єкт передається «за значенням». Це означає, що зміна цього об'єкта всередині функції не змінить його в коді, що викликає.
- Якщо в функцію передається об'єкт, що змінюється, то цей об'єкт передається за «вказівником». Зміна такого об'єкта всередині функції вплине на цей об'єкт у коді, що викликає.

Приклади передачі незмінних об'єктів у функцію

Передача числа у функцію.

У прикладі, в демонстраційних цілях реалізована функція Multi2(), яка отримує параметром число. У тілі функції це число подвоюється. Отриманий результат виводиться.

main.py

```
1  # Передача числа у функцію
2  #Число - це незмінний об'єкт, значить передається "за значенням"
3  # Оголосити функцію, яка отримує кілька і множить його на 2
4  def Mult2(number):
5      # помножити число на 2
6      number = number*2
7
8      # вивести значення числа у середені функції
9      print("Mult2.number = ", number)
10
11 # використати функцію
12 num = 25 # деяке число
13 Mult2(num) # викликати функцію
14
15 # вивести число num після виклику функції
16 print("num = ", num) # num = 25 - число не змінилось
```

```
Mult2.number =  50
num =  25


** Process exited - Return Code: 0 **
Press Enter to exit terminal
```

Як видно з результату, функції значення числа ***num*** множиться на 2 і становить 50. Однак, цей результат не передається в код, що викликає, тут значення ***num*** залишається рівним 25 як до виклику функції. Це підтверджує передачу «за значенням» незмінного об'єкта, яким є число.

Передача рядка у функцію

Демонструється робота функції ReverseStr(), яка реверсує рядок, одержуваний параметром.

main.py

```
1  # Оголосити функцію, яка отримує рядок,
2  # реверсує її та виводить на екран
3  def ReverseStr(text):
4      # Реверсувати рядок
5      txt = ''
6      #цикл перебору рядка
7      for c in text:
8          txt = c + txt
9      text = txt
10     # вивести реверсований текст
11     print('ReveseStr.text = ', text)
12 # Виклик функції ReverseStr()
13 Text = "Hello world!"
14 ReverseStr(Text)
15 print("Text = ", Text)
```

Код програми

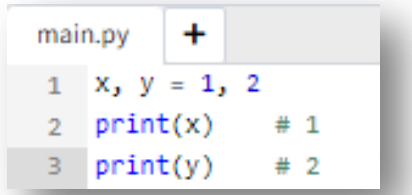
```
ReveseStr.text =  !dlrow olleH
Text =  Hello world!


** Process exited - Return Code: 0
Press Enter to exit terminal
```

Результат виконання програми

3.Упаковка і розпакування параметрів функції

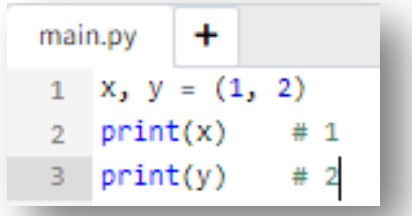
Розпакування (unpacking, також звана Деструктуризація) є розкладанням колекції (кортежу, списку тощо) на окремі значення.Так само, як і багато мов програмування, Python підтримує концепцію множинного присвоєння.

Наприклад:



В даному випадку надамо значення відразу двом змінним. Присвоєння йде за позицією: змінна x отримує значення 1, а змінна y - значення 2.

Даний приклад насправді вже представляє деструктуризацію чи розпакування. Значення 1, 2 фактично є кортежом, оскільки саме коми між значеннями говорять про те, що це кортеж. І ми також могли б написати так:



У будь-якому випадку ми маємо справу з деструктуризацією, коли перший елемент кортежу передається першою змінною, другий елемент - другою змінною і таке інше. Тобто розкладання йде за позицією.

Упаковка значень та оператор

Оператор упаковує значення в колекцію. Наприклад:

main.py

```
1  num1=1
2  num2=2
3  num3=3
4  *numbers,=num1,num2,num3
5  print(numbers)  #[1, 2, 3]
```

Тут ми упаковуємо значення з кортежу (num1,num2,num3) до списку numbers. Причому, щоб отримати список, після numbers вказується кома. Як правило, упаковка застосовується для збирання значень, що залишилися після присвоєння результатів деструктуризації. Наприклад:

main.py

```
1  head, *tail = [1, 2, 3, 4, 5]
2  print(head)  # 1
3  print(tail)  # [2, 3, 4, 5]
```

Тут змінна head відповідно до позиції отримує перший елемент списку. Всі інші елементи передаються змінну tail. Таким чином, змінна tail буде представляти список з елементів, що залишилися. Так же працює і навпаки, або елементи по середені використовуючі оператор ‘***middle’***, або всі крім першого та другого використавши оператори ‘***first’, ‘second’.***

Розпакування та оператори * та **

Оператор * разом із оператором ** також може застосовуватися для розпакування значень. Оператор * використовується для розпакування кортежів, списків, рядків, множин, а оператор ** - для розпакування словників. Особливо це може бути корисним, коли на основі одних колекцій створюються інші. Наприклад, розпакування кортежів та списків:

main.py

```
nums1 = [1, 2, 3]
nums2 = (4, 5, 6)
# роспаковка списку nums1 та кортежу nums2
nums3 = [*nums1, *nums2]
print(nums3)      # [1, 2, 3, 4, 5, 6]
```

Тут розпаковуємо значення зі списку nums1 та кортежу nums2 та поміщаємо їх у список nums3.

Подібно розкладаються словники, тільки застосовується оператор **:

main.py

```
dictionary1 = {"red":"червоний", "blue":"синій"}
dictionary2 = {"green":"зелений", "yellow":"жовтий"}
# роспаковка словників
dictionary3 = {**dictionary1, **dictionary2}
print(dictionary3)  # {'red': 'червоний', 'blue': 'синій', 'green': 'зелений', 'yellow': 'жовтий'}
```

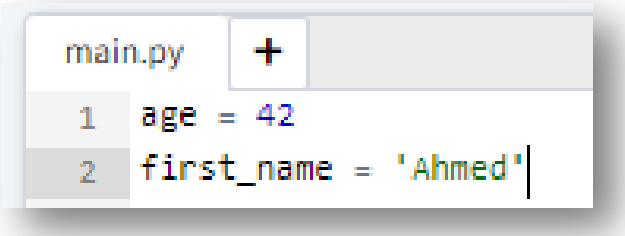
4.Функція як змінна.

Змінні - це назви значень.

У Python символ = використовується для присвоєння значення, яке знаходиться праворуч, до її назви, яка вказана ліворуч.

Змінна створена, коли ій присвоюється значення.

Нижче Python присвоює значення року змінній ***age*** та ім'я у лапках - змінній ***first_name***.



Назви змінних

- можуть складатися тільки з букв, цифр та підкреслення _ (яке звичайно використовується, щоб відокремити слова у довгих назвах змінних)
- не можуть починатися з цифри
- залежать від регістру (тобто age, Age та AGE - це три різні змінні)
- Назви змінних які починаються з підкреслення, як наприклад __alistairs_real_age, мають спеціальне значення, і тому ми не будемо використовувати їх, доки не зрозуміємо цього правила.

Рекурсія – це спосіб організації циклічного процесу шляхом виклику рекурсивної функції. ***Рекурсивна функція*** – це функція, яка містить код виклику самої себе з метою організації циклічного процесу. З допомогою рекурсивних функції можна з мінімальним об’ємом програмного коду розв’язувати деякі задачі, обійшовши використання (оголошення) зайвих структур даних. Рекурсію можна розглядати як альтернативу циклам та ітераціям.

За великим рахунком, є два алгоритми дій:

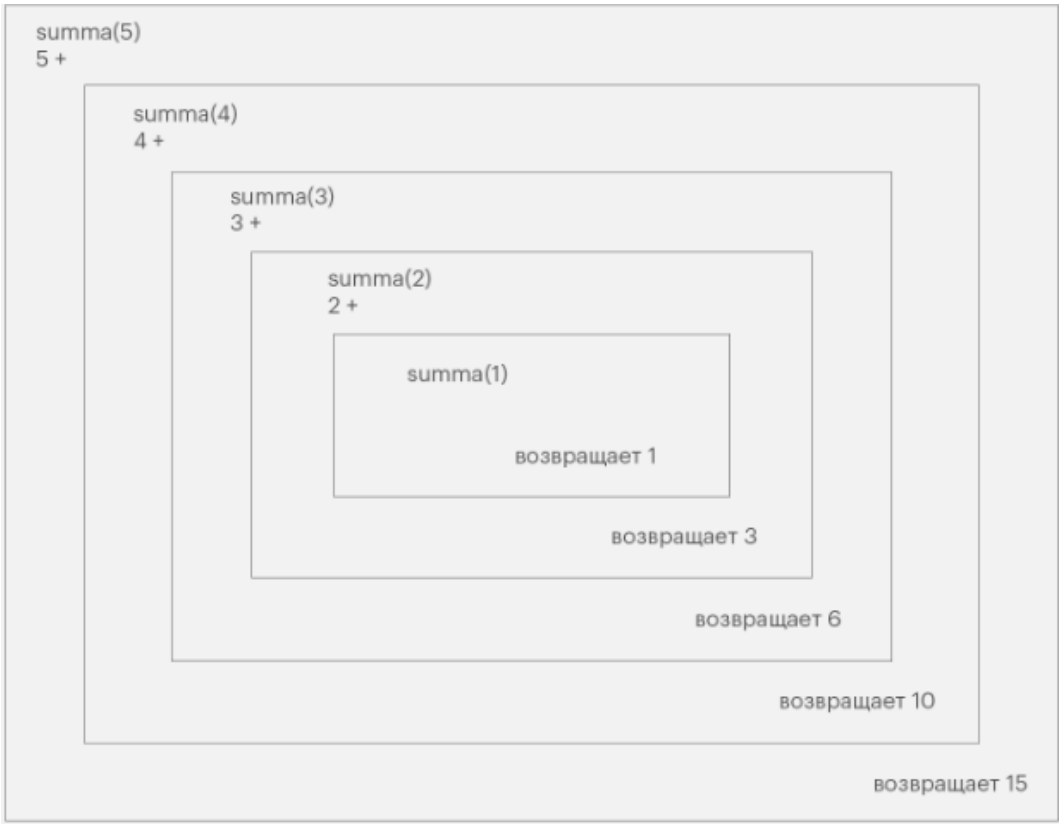
У першому випадку застосовується цикл while. Тобто ви берете коробку, відкриваєте її, потім наступну і так до того самого моменту, поки не знайдете потрібну вам коробку з ключем. Це так званий ітераційний (покроковий) метод. У Python він може виглядати так:

```
def summa(n):
    x = 0
    for n in range(1, n+1):
        x += n
    return x

summa(5)
>>> 15
```

Так як у нас стоїть завдання з повторюваними діями, але параметрами, що змінюються, ми можемо використовувати рекурсивний метод. Тобто, вирішити задачу, використовуючи її саму:

```
def summa(n):
    if n == 1:
        return 1
    return n + summa(n-1)

summa(5)
>>> 15
```

У цьому прикладі summa(1) – це 1, summa(2) – це 2 + summa(1) і так далі. Даний алгоритм простіше буде зрозуміти на наступній схемі:



І тут потрібно вказати важливу деталь. За великим рахунком, обидва алгоритми виконують одне й те саме завдання — шукають «коробку з ключем». Більше того, в даному випадку рекурсивний метод не матиме особливої переваги над ітераційним. Однак рекурсія функції краще підходить для вирішення складних багаторівневих завдань. Це буде швидше, простіше та зрозуміліше. Але що важливіше — рекурсивні рішення простіше підтримувати.

Навіщо рекурсія потрібна

Ось тут у вас може виникнути логічне питання:*«А чи взагалі потрібно використовувати рекурсію функції в Python, якщо по суті того ж результату можна досягти і ітеративними алгоритмами?»*. Питання абсолютно резонне!

У деяких ситуаціях цикли дійсно виявляються простішими і навіть ефективнішими за рекурсії. При цьому вони також можуть працювати з рекурсивними структурами даних. Більше того, часто рекурсивні функції вимагають більше місця та пам'яті, ніж ітеративні, адже вони постійно додають нові шари у стек пам'яті. Проте є нюанс — продуктивність рекурсивних функцій значно вища.

Але!

На освоєння рекурсивних функцій потрібен час та певні зусилля. І далеко не завжди вдається правильно їх реалізувати. А некоректна реалізація часто робить рекурсію досить повільною через те, що обчислення проводяться частіше, ніж це насправді необхідно. Тому дуже важливо починати практикуватися у реалізації рекурсивних функцій з максимально простих і навіть примітивних завдань, поступово підвищуючи складність та вкладеність.

З іншого боку, написання ітеративних функцій — це найчастіше громіздкіший і складніший код в той час як рекусивна функція може складатись з двох рядків.

@ZTUEDUUA

ЖИТОМИРСЬКА ПОЛІТЕХНІКА

ДЕРЖАВНИЙ УНІВЕРСИТЕТ

- **Розвиваємо лідерів**
- **Створюємо інновації**
- **Змінюємо світ на краще**