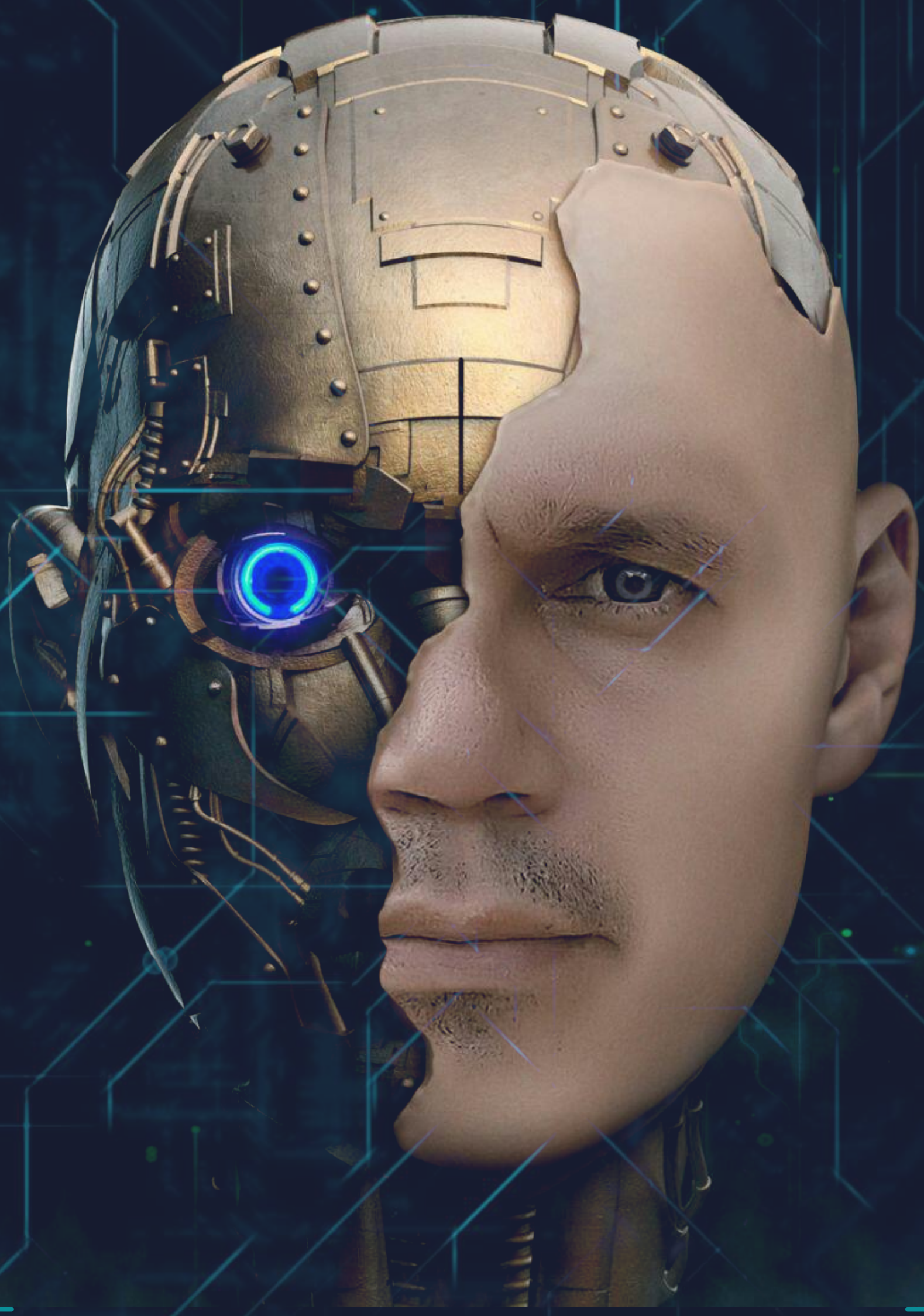
ECIT WILLIAMS DE SOUSA ARRUDA
COMPONENTE: MATEMÁTICA
PROFESSOR: PEDRO ROMÃO BATISTA



PROGRAMAÇÃO COM PYTHON: UMA INTRODUÇÃO

SUMÁRIO

Introdução

O Que é Python?

Qual Aplicativo Utilizar?

A Função Print

As Variáveis

Concatenação

Operadores Matemáticos

Ativ.: Seis Operações

Ativ.: Média

Ativ.: IMC

O Comando If

Operadores Relacionais

Operadores Lógicos

Laços de Repetição (For)

Ativ.: Números Ímpares

Ativ.: Números Triangulares

Ativ.: Soma de Ímpares

Ativ.: Tabuada Gigante

Entrada de Dados

Juros

SUMÁRIO

Listas

Ativ.: Divisores

Otimização

Mais Sobre Listas

Banco de Dados Com Listas

Laço de Repetição While

Ativ.: Mediana

Módulos

Seno, Cosseno e Tangente

Humanização da Máquina

Cara ou Coroa?

Lei dos Grandes Números

Ativ.: Simulações

Sorteando Grupos

Criando Funções

Ativ.: Área

O Módulo Turtle

Considerações Finais

INTRODUÇÃO

Todo programador sabe que programação e Matemática estão intimamente relacionados: não há como ser um programador competente sem se ter alguma relevante destreza em Matemática. O caminho inverso não deixa de ter razoável validade: quem tem facilidade com Matemática normalmente tem facilidade para aprender programação.

Além disso, com as recentes mudanças no Ensino Médio, introdução à programação agora faz parte do currículo de Matemática. Sendo assim, este material foi elaborado com o intuito de ser um guia introdutório de Python para ser utilizado na escola: a ideia, aqui, é a de relacionar Python com assuntos de Matemática que são explorados nos Ensinos Fundamental e Médio.

Dessa forma, não espere um curso completo de Python: foquei em comandos, funções e módulos que, ao mesmo tempo em que te guiarão nos primeiros passos pelo mundo da programação, explorarão vários assuntos de Matemática que na escola são estudados.

Bom proveito!

O QUE É PYTHON?

Python é uma linguagem de programação popular de alto nível lançada em 1991. Atualmente, é mantida por uma organização sem fins lucrativos chamada Python Software Foundation e tem o suporte de uma comunidade muito ativa e que produz vastos materiais.

Nos últimos anos, Python vem se destacando ferozmente em dois campos distintos: setor financeiro e Data Science (ciência de dados). É também bastante utilizada no meio educacional tendo em vista sua sintaxe simplificada, sua versatilidade e uma gama diversa de módulos (como no meu caso, que utilizo em aulas de geometria, probabilidade e estatística).

Imagem gerada no Bing. As próximas também

QUAL APLICATIVO UTILIZAR?

Não é comum programarmos no smartphone, porém é possível e a prática tem ficado cada vez mais popular. Em minhas aulas aqui no WSA, tenho sugerido a utilização do aplicativo Pydroid3, disponível na Google Play Store.

Logotipo do Pydroid3

O Pydroid3 possui uma versão gratuita e uma versão paga. Para nossos objetivos, a versão gratuita é mais do que suficiente.

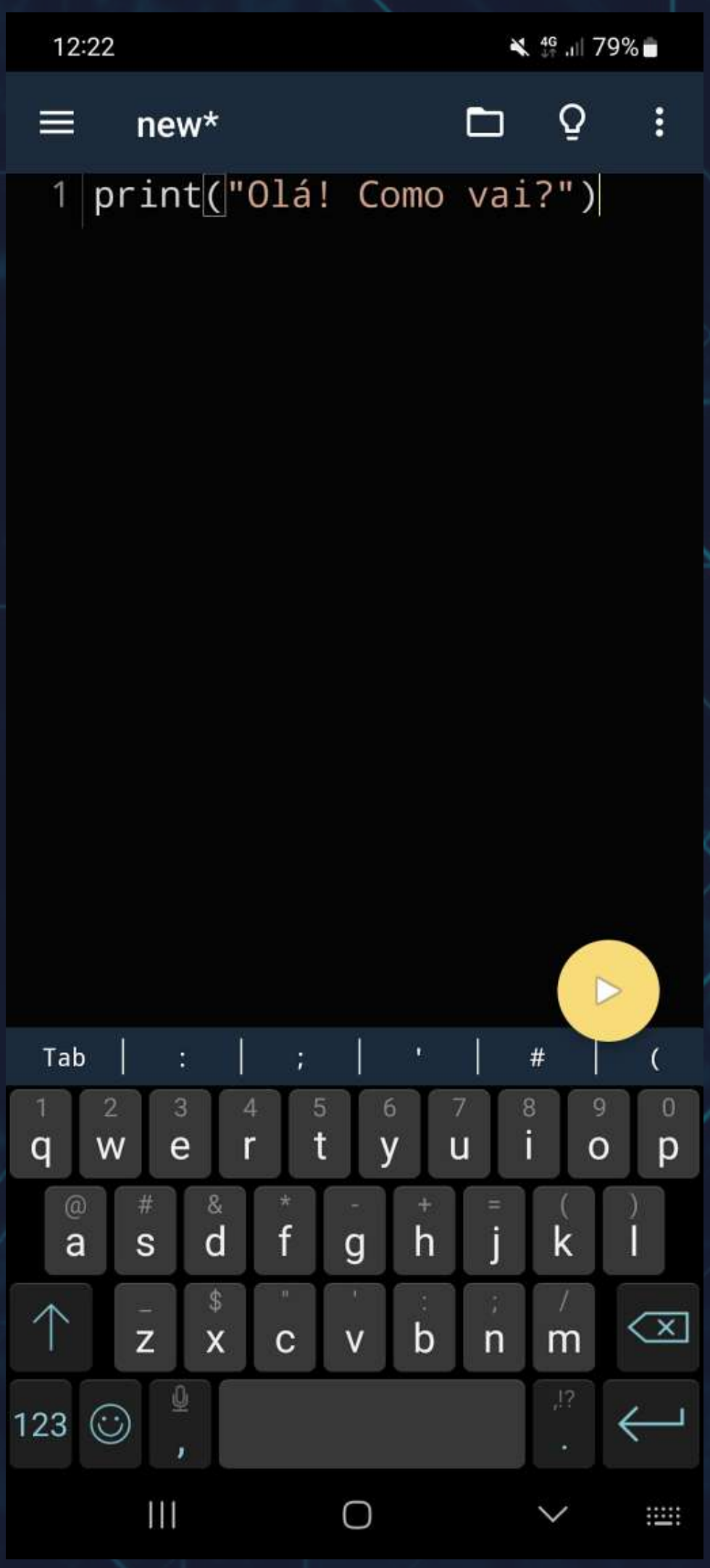
Para instalar, basta entrar na Play Store e buscar por "pydroid3". Instalação finalizada, o Pydroid3, ao ser iniciado pela primeira vez, fará algumas perguntas simples de múltipla escolha (tais como por que você está instalando, que tipo de projeto pretende fazer etc.). Finalizada essa parte, ele já fica pronto para uso: você já pode colocar os comandos e apertar no botão de Play (abaixo, à direita) para o programa ser executado.

A FUNÇÃO PRINT

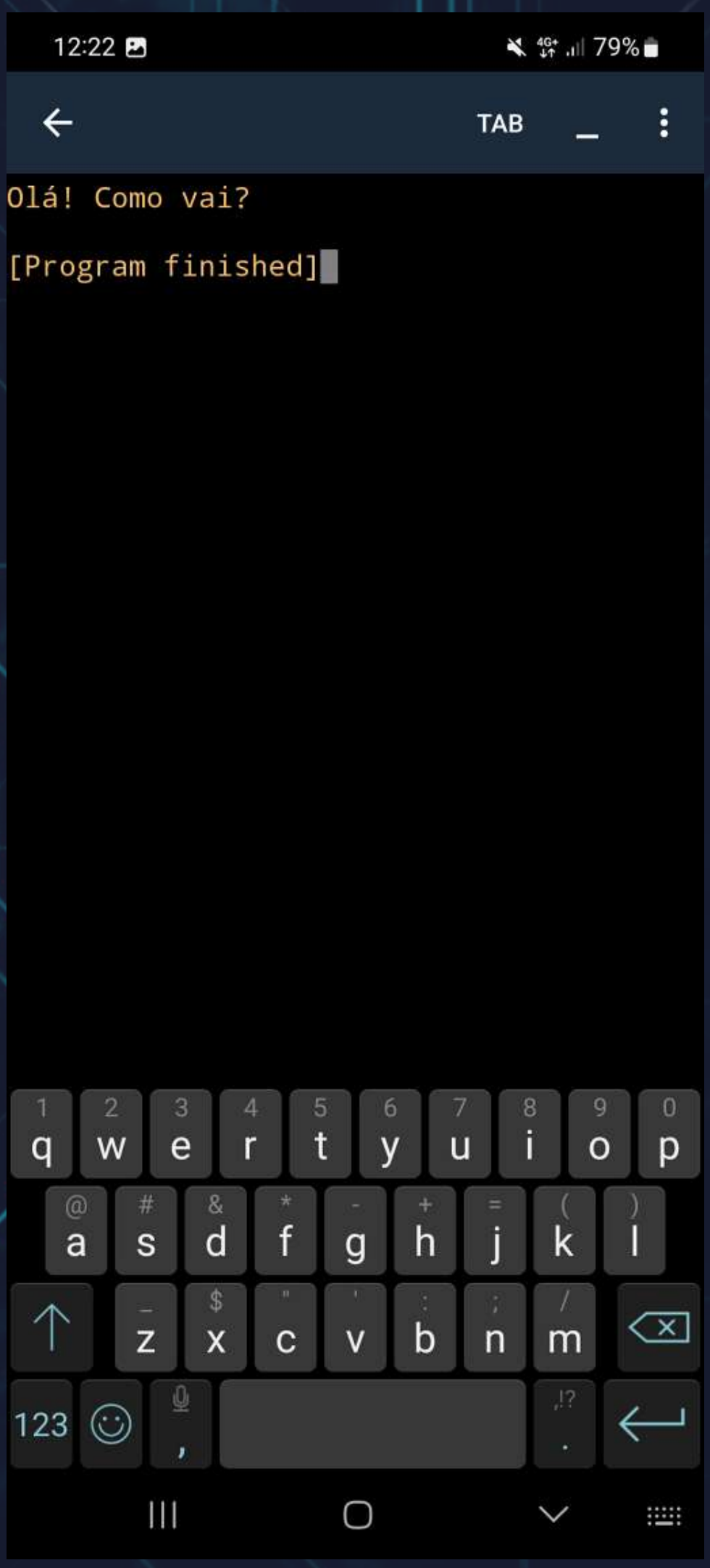
A função print() serve para exibir informações na tela. Estou iniciando por ela, pois a utilizaremos sempre. Sua sintaxe é:

```
print("texto a ser exibido")
```

Por exemplo:



Código



Execução

A FUNÇÃO PRINT

Além disso, por meio da função print, já é possível exibir números e realizar operações matemáticas.

```
print(5)
print(5+5)
print(85*123/3)
```

que trará como resultado:

```
5
10
3485
```

Tenha cuidado! Quando for trabalhar com números, as aspas não devem ser colocadas! Caso sejam colocadas, os números serão tratados como texto, como no exemplo a seguir:

```
print("5+5")
```

que trará como resultado:

```
5+5
```

AS VARIÁVEIS

Variáveis são imprescindíveis em programação: pode-se dizer que, sem elas e com um pouco de exagero, não há programas.

Em programação, o conceito de variável é parecido com o utilizado em Matemática: são letras ou nomes que recebem valores.

Se o valor recebido for texto, dizemos que trata-se de uma variável do tipo string; se for um número inteiro, será uma variável do tipo integer (abreviação: int); para números decimais, temos o float; e assim por diante. A seguir, podemos ver exemplos:

```
nome = "Pedro"
idade = 41
altura = 1.71
print(nome)
print(idade)
print(altura)
```

No programa acima, nós criamos três variáveis: a variável nome é do tipo string (texto); a variável idade recebeu um número inteiro (portanto, ela é int); já altura é um float. Ao executar o programa, teremos como resultado:

```
Pedro
41
1.71
```

CONCATENAÇÃO

No exemplo anterior, o programa exibiu o valor de três variáveis em três linhas diferentes. Caso quiséssemos juntar as informações numa frase só, estaríamos fazendo uma concatenação.

Podemos utilizar o comando print para isso, e há inúmeras formas de isso ser feito. Observe mais abaixo uma delas:

```
nome = "Pedro"
idade = 41
altura = 1.71
print(nome,"tem",idade,"anos","e mede",altura)
```

cujo resultado será:

```
Pedro tem 41 anos e mede 1.71
```

Há muitas outras maneiras de isso ser feito, mas, por enquanto, faremos deste modo.

Ademais, podemos utilizar operações com variáveis, claro:

```
preco = 10.55
print("Quinze produtos custarão",preco*15)
```

Resultado:

```
Quinze produtos custarão 158.25
```

OPERADORES MATEMÁTICOS

Os operadores matemáticos que nós vamos utilizar são:

TABELA DE OPERAÇÕES MATEMÁTICAS	
+	Adição
-	Subtração
*	Multiplicação
/	Divisão
//	Divisão inteira (retorna sempre um inteiro truncado para o valor imediatamente inferior)
%	Resto da divisão
**	Potenciação (também pode ser utilizado para calcular raízes)

Dessa forma, as quatro operações podem ser chamadas utilizando-se +, -, * e /. Em alguns poucos momentos precisaremos utilizar // quando necessitarmos de resultados inteiros de divisões. Sobre potência, sua utilização é assim:

```
numero = 5
cubo = numero**3
print(cubo)
```

Resultado:

125

Se o objetivo for apenas o resultado do cálculo, a mesma potência obtida anteriormente pode ser adquirida utilizando-se os números e a operação direto dentro do print:

```
print(5**3)
```

De um modo geral, no entanto, tendo em vista a necessidade de sempre estarmos utilizando variáveis, recomendo utilizá-las sempre que possível para se criar o hábito.

No que se refere às raízes, podemos calcular utilizando expoentes racionais: assim, 1/2 = 0.5 retorna a raiz quadrada; 1/3 retorna a raiz cúbica, e assim por diante.

```
n = 50
r = n**0.5
print("A raiz quadrada de",n,"é",r)
```

Cujo resultado será:

A raiz quadrada de 50 é 7.07106781186

Observe que, se quisermos a raiz quadrada de outro número, basta alterar o valor de "n" na primeira linha que todo o resto já será feito e impresso na tela - uma das vantagens do uso de variáveis.

ATIV.: SEIS OPERAÇÕES

O programa abaixo realiza as seis operações com os números 7 e 3.

```
a = 7
b = 3
print("Soma = ", a + b)
print("Diferença = ", a - b)
print("Produto = ", a * b)
print("Quociente = ", a / b)
print("Quociente inteiro = ", a // b)
print("Potência = ", a ** b)
print("Raiz cúbica = ", a ** (1/b))
```

Note que, para calcular a raiz cúbica de 5, coloquei um terço entre parênteses para evitar erros na ordem dos cálculos.

Os resultados são:

```
Soma =  10
Diferença =  4
Produto =  21
Quociente =  2.3333333333333335
Quociente inteiro =  2
Potência =  343
Raiz cúbica =  1.912931182772389
```

Atividade 01: É momento de começar a praticar: escolha dois números de sua preferência e faça um programa que calcule as seis operações mostradas acima.

ATIV.: MÉDIA

Um programa simples para calcular a média de um estudante em um componente de duas notas.

```
nome = "Pedro Romão"
nota1 = 8.5
nota2 = 7.7
media = (nota1 + nota2) / 2
print("O estudante",nome,"teve média igual a",media)
```

Atividade 02: Agora, faça um programa que calcule a média de um estudante num componente que possui três notas.

Observações:
- Em Matemática, nós utilizamos chaves, colchetes e parênteses para indicar ordem de operações. Em Python, só são utilizados parênteses, nada mais. No código acima, "(nota1 + nota2)" estão entre parênteses para que a soma das duas notas seja calculada antes da divisão - CUIDADO! Se colocar "nota1 + nota2/2", o cálculo ficará errado, pois a divisão "nota2/2" será feita antes da adição;
- Você já deve ter observado que o separador de decimal, em Python, é o ponto, e não a vírgula.

ATIV.: IMC

O IMC (Índice de Massa Corporal) pode ser calculado por meio da fórmula:

$$IMC = \frac{massa}{altura^2}$$

E, como você já deve saber, serve para determinar se uma pessoa está com *peso* normal, acima do *peso* ou abaixo do *peso*.

A massa deve ser dada em quilogramas (kg) e a altura, em metros (m). Assim, para uma pessoa que tem 1,66 m e massa 70 kg, teremos o seguinte cálculo:

$$IMC = \frac{70}{1{,}66^2} = 25{,}40$$

Depois disso, basta verificar numa tabela de IMC para saber a situação da pessoa (o que faremos mais à frente).

Atividade 03: Sendo assim, agora é com você: elabore um código Python que calcule o IMC de uma pessoa, dados seu nome, massa e altura.

O COMANDO IF

Um dos principais comandos de qualquer linguagem de programação, o comando if (se) é um controlador de fluxo, ou seja, por meio de sua utilização, o programador criará estruturas condicionais, fazendo com que o programa decida se vai ou não executar uma parte do código.

A sintaxe do if é bastante simples. Veja abaixo:

```
nota = 5
if nota == 5:
    print("A nota é igual a 5")
```

Com esse código, o programa irá verificar se a variável nota é igual a 5 - em caso afirmativo, o comando print escrito acima será executado.

Observe que, no final da linha do if, deve ser inserido os dois pontos (":"), símbolo utilizado em Python antes de um bloco de código, que por sua vez é um conjunto de comandos que será executado. O bloco de código é definido por um espaço extra em relação à margem esquerda do editor: todos os comandos alinhados horizontalmente serão executados.

Por exemplo:

```
nota = 5
if nota == 5:
    nota = 0
    print("A nota é igual a 5")
print("Até a próxima!")
```

Neste caso, os comandos

```
nota = 0
print("A nota é igual a 5")
```

serão executados como bloco de código (dentro do if), pois ambos possuem o mesmo alinhamento horizontal. Já o comando

```
print("Até a próxima!")
```

não faz parte do if, pois não está alinhado com os demais comandos. Dessa forma, ele será executado independentemente do if.

Nota: Esse alinhamento chama-se indentação. Por padrão, geralmente utilizamos 4 espaços (no caso do Pydroid3, quando você coloca os dois pontos e dá enter, ele já insere os espaços automaticamente).

OPERADORES RELACIONAIS

Para podermos utilizar o comando if em toda sua completude, precisamos conhecer os operadores relacionais bem como os operadores lógicos.

TABELA DE OPERADORES RELACIONAIS	
==	Igualdade
>	Maior que
<	Menor que
>=	Maior ou igual
<=	Menor ou igual
!=	Não é igual / diferente

Exemplos:

```
media = 8
if media >= 7:
    print("Aprovado!")
else:
    print("Reprovado!")
```

O resultado será "Aprovado!", pois a média é maior ou igual a 7.

Observe que utilizei também o comando else, que significa "caso contrário". Além do else, temos o elif, que significa "se não, se", e é utilizado quando queremos programar mais de duas condições.

OPERADORES LÓGICOS

Além disso, podemos utilizar os operadores lógicos and (e) e or (ou). Observe com atenção:

```
hora = 7
if hora >= 6 and hora < 12:
    print("Bom dia!")
elif hora >= 12 and hora < 18:
    print("Boa tarde!")
elif hora >= 18 and hora < 24:
    print("Boa noite!")
else:
    print("Boa madrugada!")
```

Ao ser executado, o programa irá retornar

Bom dia!

pois a hora informada (hora = 7) é maior ou igual a 6 e menor que 12. Como último comando condicional coloquei o else, que será executado como caso contrário, isto é, se todas as condições anteriores forem falsas (o que vai recair na madrugada).

Notas:
- O comando and (e) relaciona duas ou mais afirmações e só retorna verdadeiro (true) se todas forem verdadeiras; já o comando or (ou) é diferente: se apenas uma das afirmações presentes for verdadeira, então o retorno será verdade (true).

Essa diferenciação em programação é demasiadamente importante para que haja um correto controle de fluxo. Observe os exemplos:

Proposição 1: Pedro é professor e astronauta.

Proposição 2: Pedro é professor ou astronauta.

Cada proposição é composta de duas afirmações: "Pedro é professor"; "Pedro é astronauta". Como sabemos, Pedro (euzinho) não é astronauta. Logo, a primeira proposição é falsa (false). Já na segunda proposição, como Pedro é professor e foi utilizado um ou, então ela é verdadeira (true). Isso é... lógica!

Veja abaixo o que costumamos chamar de tabelas-verdade do "e" e do "ou".

p	q	p and q
V	V	V
V	F	F
F	V	F
F	F	F

p	q	p or q
V	V	V
V	F	V
F	V	V
F	F	F

Note que p e q são afirmações.

De posse de todas essas informações (comando if e operadores lógicos e relacionais), transforme-as em conhecimento:

Atividade 04: Crie um programa de IMC mais completo que o anterior (veja atividade 03), que dirá, ao final, se o usuário está abaixo do *peso*, com *peso* ideal e assim por diante. Para tanto, utilize a tabela abaixo.

IMC	Classificação
Abaixo de 18,5	Abaixo do peso
Entre 18,6 e 24,9	Peso ideal
Entre 25,0 e 29,9	Um pouco acima do peso
Entre 30,0 e 34,9	Obesidade grau I
Entre 35,0 e 39,9	Obesidade grau II
Acima de 40,0	Obesidade grau III

LAÇOS DE REPETIÇÃO (FOR)

O comando for (faça) permite que você repita um bloco de códigos quantas vezes você quiser. Há milhões de aplicações para isso, e laços de repetição são uma das grandes vantagens de se utilizar uma linguagem de programação para realizar cálculos e experimentos/simulações, como veremos a partir de agora.

Existem muitas formas de se utilizar o comando for, mas, por ora, utilizaremos a seguinte estrutura:

```
for x in range (número de repetições):
    comando
    comando
    comando
    ...
```

O "x" é uma variável que receberá um novo valor depois de cada repetição (e você poderá utilizar esse valor como bem quiser; você também pode trocar o "x" por outra letra ou por uma palavra). Já o "range" é uma função que gera um intervalo numérico: é esse intervalo que a variável "x" vai percorrer, valor por valor.

Vejamos abaixo um primeiro exemplo.

```
for x in range (10):
    print(x)
```

O resultado será:

```
0
1
2
3
4
5
6
7
8
9
```

Note que, na primeira repetição, o valor de "x" é 0 - importante lembrar disso: em programação, o primeiro elemento geralmente é indicado pelo zero. No mais, o comando print foi executado 10 vezes, e dentro dele poderíamos ter colocado texto, operações e assim por diante.

Na função range, nós também podemos colocar o início do intervalo, fazendo com que a primeira repetição não seja referenciada com o número 0. Se quisermos, por exemplo, fazer um programa que liste os 5 primeiros números quadrados, começando de 1 x 1, podemos escrever:

LAÇOS DE REPETIÇÃO (FOR)

```
for x in range (1,6):
    print(x*x)
```

Resultado:

```
1
4
9
16
25
```

Que são justamente os 5 primeiros números quadrados:

E para imprimir na tela os 100 primeiros, basta colocar 101 no lugar do 6.

Com isso, você já deve ter percebido que o limite superior colocado no range nunca é atingido. Assim, range(10) criará um intervalo numérico inteiro que vai de 0 até 9; range(50), que vai de 0 até 49; range(5,20), um intervalo que vai de 5 até 19).

ATIV.: NÚMEROS ÍMPARES

Segue abaixo um programa para escrever os 50 primeiros números pares (lembrando que, para se obter números pares, basta multiplicar a lista numérica por 2):

```
for numeros in range (50):
    print(numeros*2)
```

Atividade 05: Faça um programa que escreva:

a) Os 500 primeiros números ímpares.

b) Os 80 primeiros múltiplos de 5.

c) Os 300 primeiros múltiplos de 7.

ATIV.: NÚMEROS TRIANGULARES

Em Matemática, temos também os números triangulares:

A fórmula para se obter os números tringulares é:

$$T_x = \frac{x \cdot (x+1)}{2}$$

Onde o x representa a posição do número na fila. Assim, para obtermos o décimo número triangular, devemos colocar 10 no lugar do x e realizar os cálculos indicados na fórmula (na verdade, trata-se de uma equação do segundo grau).

ATIV.: NÚMEROS TRIANGULARES

Dessa maneira, podemos fazer:

Para x = 1:

$$T_1 = \frac{1\cdot(1+1)}{2} = 1$$

Para x = 2:

$$T_2 = \frac{2\cdot(2+1)}{2} = 3$$

Para x = 3:

$$T_3 = \frac{3\cdot(3+1)}{2} = 6$$

Atividade 06: Crie um programa que escreva os 1000 primeiros números triangulares.

ATIV.: SOMA DE ÍMPARES

Vamos agora usar laços de repetição para realizar múltiplos cálculos de uma forma muito simples. Normalmente, para esses casos, nós precisaremos de variáveis de suporte. Primeiramente, vamos ver um código para calcular a soma dos 100 primeiros números naturais:

```
soma = 0
for n in range (100):
    soma = soma + n
print("A soma é:",soma)
```

Observe que, a cada repetição, a variável "soma" adiciona a seu atual valor o número "n", que é alterado a cada repetição.

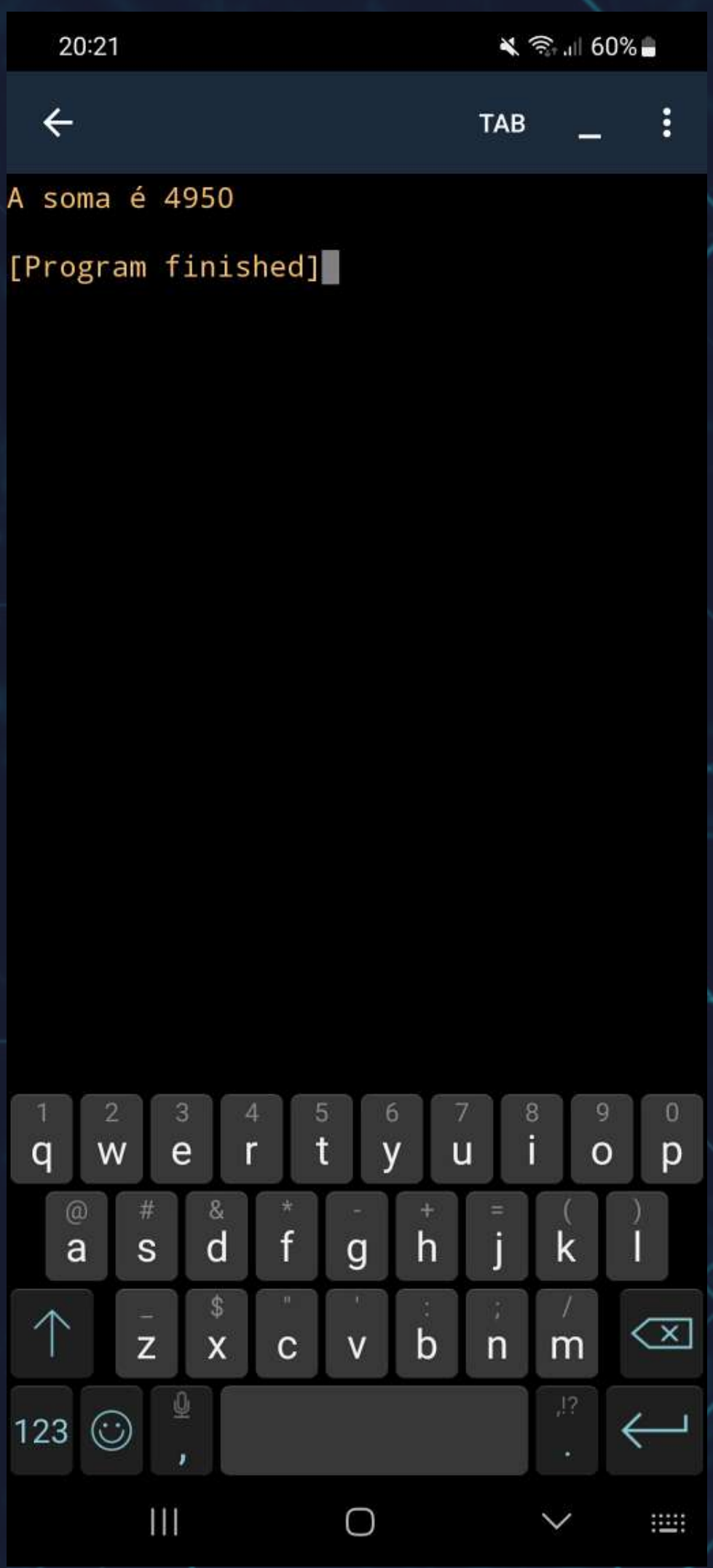
Abaixo, um código para adicionar os 20 primeiros números pares:

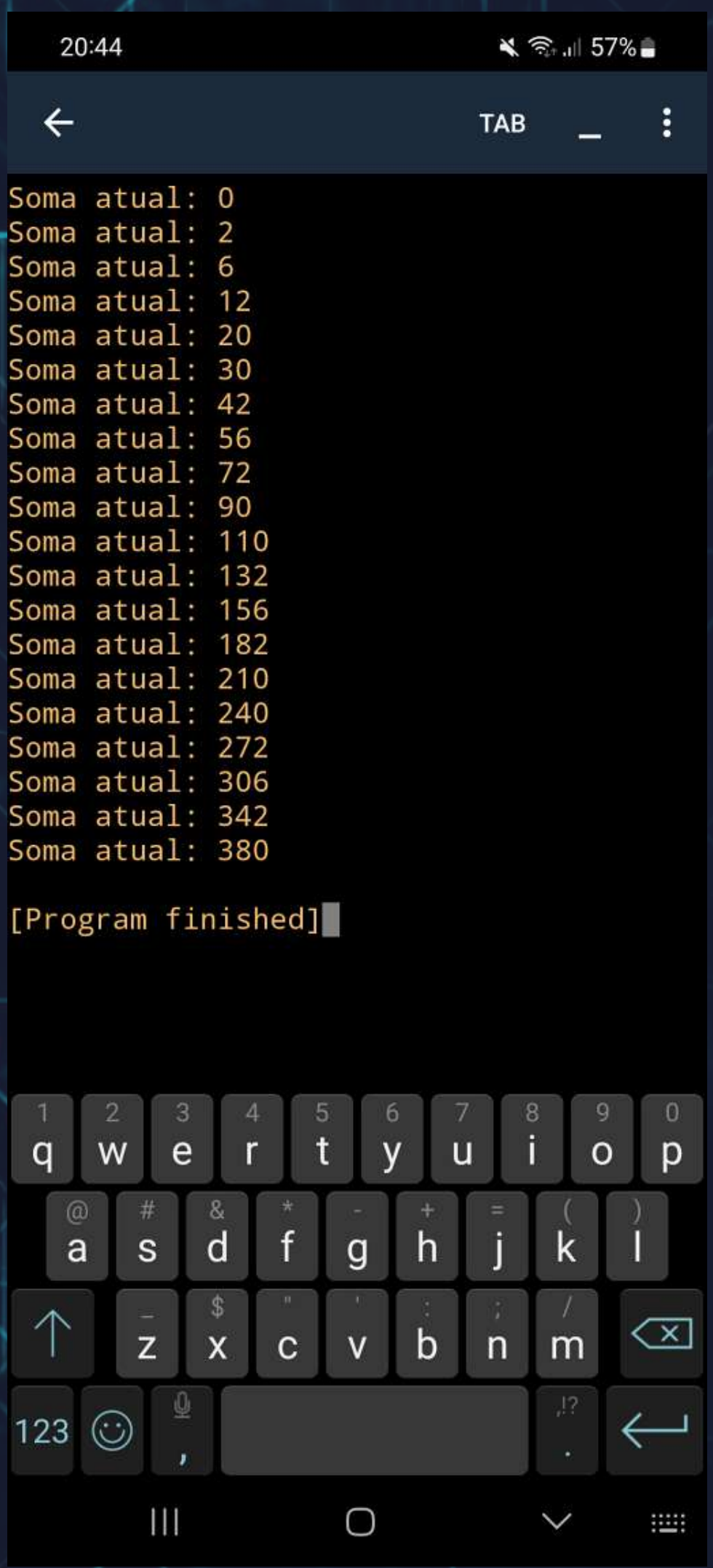
```
soma = 0
for n in range (20):
    soma = soma + n*2
    print("Soma atual:",soma)
```

Já no exemplo acima, note que o comando "print" está dentro do "for": com isso, ele será executado várias vezes junto com o bloco de repetição e o valor da variável "soma" será exibido 20 vezes (a cada atualização).

ATIV.: SOMA DE ÍMPARES

A seguir, temos os resultados dos dois códigos acima sendo executados no Pydroid3:





Atividade 07: Escreva um código para calcular a soma dos 50 primeiros números ímpares (coloque o "print" dentro do "for" para que a soma fique sendo mostrada a cada atualização). Depois de executar, observe se você nota alguma relação entre as somas e os números quadrados.

ATIV.: TABUADA GIGANTE

Podemos utilizar um for dentro do outro. Nesta atividade, faremos isso para construir uma tabuada gigante (ou o quão grande quisermos).

```
for num1 in range (10):
    for num2 in range (10):
        print(num1,"x",num2,"=",num1*num2)
```

Prontinho! O código acima irá gerar uma tabuada nos seguintes moldes:

```
0 x 0 = 0
0 x 1 = 0
0 x 2 = 0
...
0 x 10 = 0
1 x 0 = 0
1 x 1 = 1
1 x 2 = 2
...
2 x 0 = 0
2 x 1 = 2
2 x 2 = 4
...
...
9 x 9 = 81
```

Se quisermos tirar o zero da tabuada, basta ajustar o range e colocar "range(1, 10)". E alterando o 10, podemos criar uma tabuada gigante o quanto quisermos.

ATIV.: TABUADA GIGANTE

Observe que, o segundo "for" será executado várias vezes (um número de vezes indicado pelo primeiro "for"), todavia, ao se chegar nele, o primeiro "for" deve *aguardar* que ele faça suas repetições. Só depois que o segundo "for" é executado várias vezes, o processo volta para o primeiro "for".

Atividade 08: Crie uma tabuada gigante na qual o primeiro fator comece do 2 e o segundo fator, do 1 (isto é, a primeira linha da tabuada deve ser 2 x 1 = 2). A última linha da tabuada deve ser 100 x 100 = 10000.

ENTRADA DE DADOS

Até agora, os nomes ou números a serem utilizados pelo programa foram colocados dentro do próprio código. No entanto, em um programa profissional de fato, de um modo geral, é o usuário quem insere os dados enquanto o programa é executado. Vamos agora aprender a fazer isso por meio da utilização da função input().

Sua sintaxe é:

```
variavel = input("texto a ser exibido")
```

Dessa forma, uma caixa de texto será aberta durante a execução do programa e o usuário poderá digitar. No caso acima, o que o usuário digitar será recebido pela variável "variavel". Por exemplo, o código

```
nome = input("Escreva seu nome: ")
```

abrirá uma caixa de texto para o usuário digitar seu nome. Depois que ele concluir, então o nome por ele digitado será capturado pela variável "nome", que depois poderá ser manipulada como bem quisermos.

A função "input", por padrão, cria um texto (string). Se quisermos inserir um número inteiro, por exemplo, devemos especificar com a função int():

```
num1 = int(input("Digite sua idade: "))
```

Caso deseje abrir um input para um número decimal, deve ser utilizada a função float():

```
altura = float(input("Sua altura: "))
```

Importante:
- Se for digitado letras para o caso acima, teremos um erro na execução do programa;
- Observe que, se dois parênteses forem abertos, dois parênteses precisam ser fechados;
- Cuidado nas aspas! O Pydroid3, bem como outras IDEs de Python, normalmente (para facilitar a digitação) inserem duas aspas quando você coloca apenas uma; o mesmo pode ocorrer com os parênteses. Em caso de erro de execução, verifique se não tem aspas ou parênteses a mais.

O seguinte código cria um programa que recebe as três notas de um estudante, calcula sua média e a exibe.

```
nome = input("Digite seu nome: ")
n1 = float(input("Nota 1: "))
n2 = float(input("Nota 2: "))
n3 = float(input("Nota 3: "))
media = (n1+n2+n3)/3
print("A média de",nome,"foi",media)
```

Em programação é muito comum fazermos polimentos para evitar erros e também para ajudar o usuário a não cometer erros. Por exemplo, para o caso anterior das médias, se por acidente o usuário digitar uma nota negativa ou acima de dez, é importante que o programa detecte o erro e avise ao usuário. Para isso, vamos utilizar o comando "if" bem como operadores lógicos e operadores relacionais:

```
nome = input("Digite seu nome: ")
n1 = float(input("Nota 1: "))
n2 = float(input("Nota 2: "))
n3 = float(input("Nota 3: "))
if n1 < 0 or n1 > 10 or n2 < 0 or n2 > 10
or n3 < 0 or n3 > 10:
      print("Uma das notas foi digitada de
forma incorreta. Por favor, verifique.")
else:
    media = (n1+n2+n3)/3
    print("A média de",nome,"foi",media)
```

Com isso, o programa alertará o usuário se ele errar. Caso esteja tudo em ordem com as notas, o "else" deixará o fluxo seguir, a média será calculada e depois exibida na tela.

Atividade 09: Escolha duas das atividades anteriores e, para a entrada de dados, reescreva o código utilizando a função input().

JUROS

A função input nos permite também criar menus para que o usuário possa escolher entre dois ou mais caminhos diferentes. Vamos exemplificar utilizando os juros simples e os juros compostos.

Para relembrar, a fórmula para os juros simples é:

$$J = C \cdot i \cdot t$$

E a dos juros compostos:

$$M = C \cdot (1 + i)^t$$

Onde "J" são os juros, "C" é o capital, "i" é a taxa, "t" é o período e "M", o montante (juros + capital).

Observe que, para o cálculo dos juros, seja simples ou composto, o usuário precisará informar o capital, a taxa e o período. Vamos criar um programa que perguntará ao usuário que tipo de juros ele quer calcular.

JUROS

Inicialmente, será exibido um menu para a escolha entre juros simples ou compostos; posteriormente, os dados numéricos serão solicitados.

```
print("Digite 1 para juros simples")
print("Digite 2 para Juros compostos")
escolha = int(input("Faça sua escolha: "))
c = float(input("Capital: "))
i = float(input("Taxa: "))
t = float(input("Período: "))
juros = 0
montante = 0
if escolha == 1:
    juros = c * i/100 * t
    print("Juros simples =", juros)
elif escolha == 2:
    montante = c * (1 + i/100)**t
    print("Montante =",montante)
    print("Juros =",montante - c)
else:
    print("Escolha inválida!")
```

Note que dividi o i por 100, pois a taxa é dada em porcentagem.

LISTAS

Em Python, existe um tipo de estrutura de dados que pode receber inúmeros valores: são as listas. Para declarar uma lista, devemos utilizar colchetes. Seu funcionamento lembra um pouco a indexação vista no estudo de matrizes na escola.

Por exemplo, é possível colocar três nomes numa lista:

```
nomes = ["Ana", "Beto", "Yasmin"]
```

Para darmos um print na variável "nome", fazemos:

```
print(nomes)
```

Resultado:

```
['Ana','Beto','Yasmin']
```

E se quisermos referenciar apenas um elemento da lista? Neste caso, utilizamos índices entre colchetes depois do nome da lista (o 0 será o primeiro elemento; o 1 será o segundo; etc.).

nomes[0] -> Primeiro elemento da lista.
nomes[1] -> Segundo elemento.
nomes[2] -> Terceiro elemento.

Veja:

```
cores = ["Azul", "Verde", "Vermelho"]
print(cores[0])
print(cores[1])
print(cores[2])
```

Resultado:

```
Azul
Verde
Vermelho
```

Para adicionarmos um elemento a uma lista, um comando muito utilizado é o "append". Para o caso acima, se quisermos adicionar uma nova cor, fazemos:

```
cores.append("Ciano")
```

Agora, se dermos um print na lista:

```
print(cores)
```

O resultado será:

```
['Azul','Verde','Vermelho','Ciano']
```

E podemos acessar o "Ciano" utilizando cores[3].

Listas tabém podem receber outras listas.

ATIV.: DIVISORES

Vamos agora aplicar as informações vistas sobre listas para criar um programa que identifica os divisores de um número e os exibe. Para guardar esses divisores, vou utilizar uma lista.

Além disso, vou utilizar o operador matemático %, visto na tabela de operadores matemáticos. Esse símbolo retorna o resto de uma divisão. Por exemplo:

```
dividendo = 10
divisor = 3
print(dividendo % divisor)
```

Retornará:

```
1
```

Pois 10 dividido por 3 tem como resto 1.

Ademais, vamos utilizar repetições: a ideia é que o usuário digite um número; depois disso, o nosso programa deverá pegar esse número e dividí-lo por todos os números naturais menores que ele: aquelas divisões que deixarem resto zero são especiais, pois mostra que aquele número que está dividindo é, de fato, um divisor do número informado pelo usuário - então, quando isso ocorrer, devemos guardar o referido número numa lista.

A lista, por sua vez, começará vazia: devemos então declará-la sem valores no início do código. Vamos lá?

```
lista_div = []
numero = int(input("Informe o número: "))
for div in range (1, numero):
    if numero % div == 0:
        lista_div.append(div)
print("A lista de disivores é:",lista_div)
```

Observe que, no range, nós colocamos o 1 para iniciar o intervalo, justamente porque não há divisão por zero (se colocarmos apenas range(numero) haverá esse erro e o programa não será executado).

Com o código acima, se você executar e colocar o número 10, o resultado será:

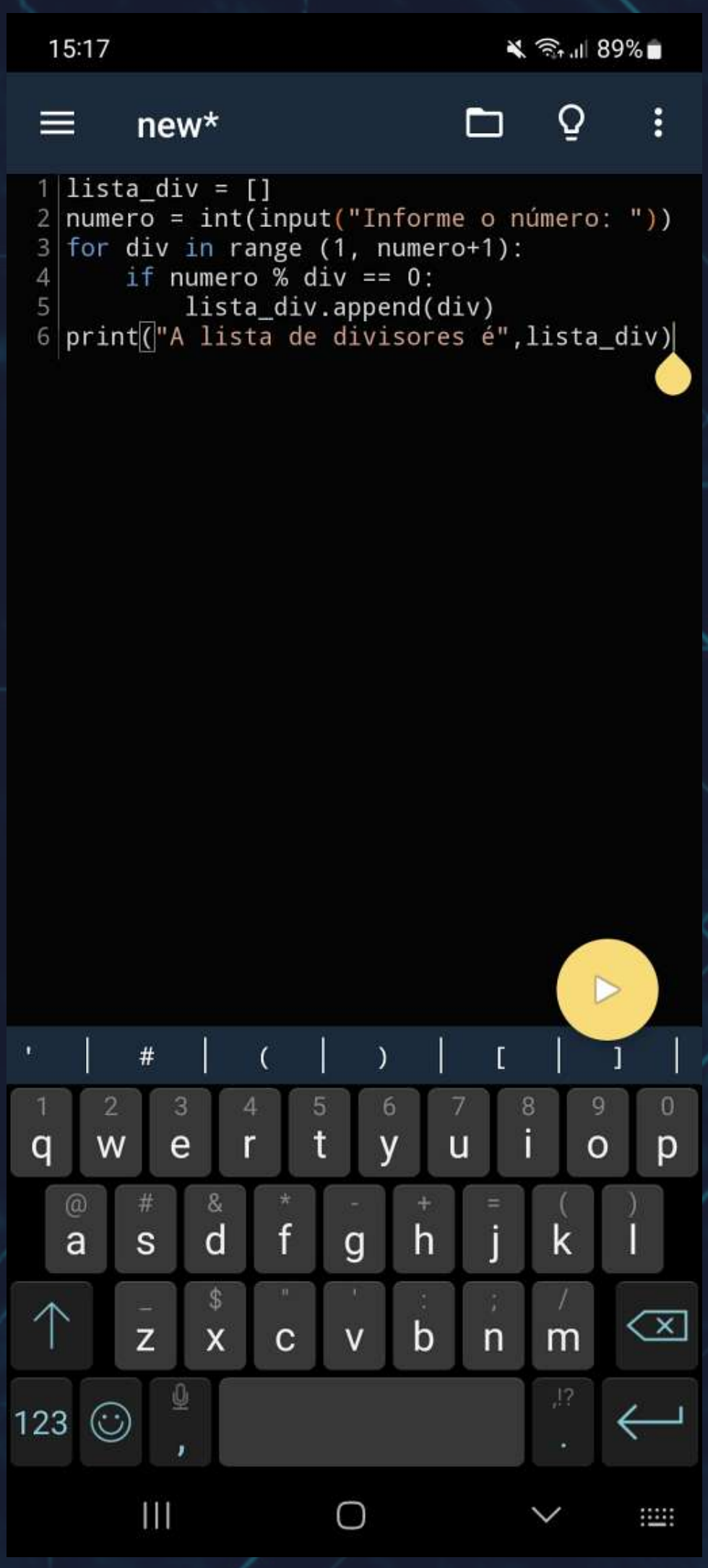
A lista de divisores é: [1, 2, 5]

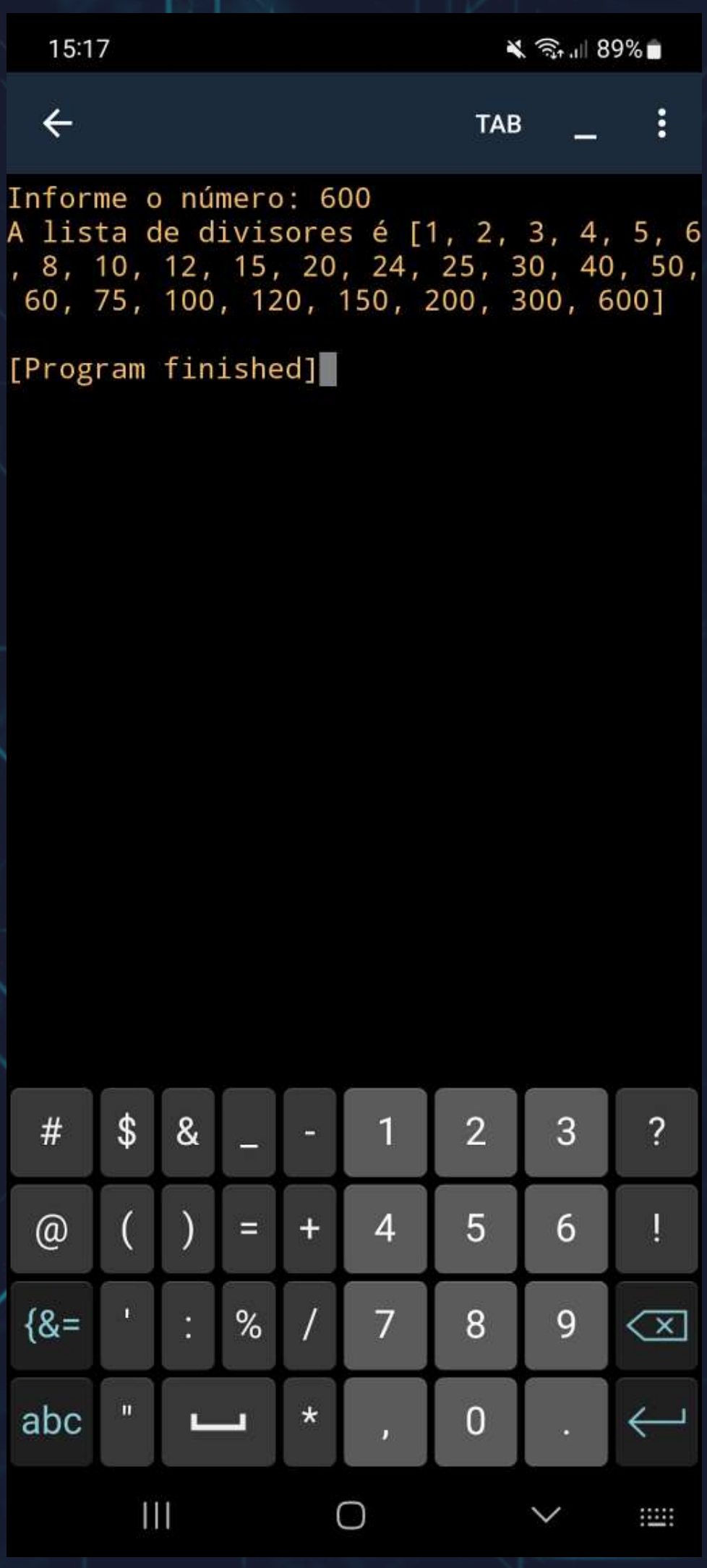
Portanto, ainda há um problema: o número 10 é divisor de si mesmo e ficou de fora da lista. Isso ocorre porque, ao colocarmos range (1, numero), na última repetição, a variável div recebe o valor numero - 1. Logo, no caso do 10, o último resto calculado foi de 10 / 9. Para corrigir isso, devemos colocar numero + 1 dentro do range. O que faremos a seguir.

ATIV.: DIVISORES

```
lista_div = []
numero = int(input("Informe o número: "))
for div in range (1, numero+1):
    if numero % div == 0:
        lista_div.append(div)
print("A lista de disivores é:",lista_div)
```

Prontinho!





OTIMIZAÇÃO

O programa dos divisores me faz sempre pensar no termo "otimização", muito importante em programação. Você já deve ter notado que o smartphone faz milhões de cálculos por segundo: ele determina os divisores de 500, por exemplo, em uma fração de segundo. Mas e se quisermos os divisores do número 147.000.000.000.848? Neste caso, o processamento será bem mais demorado. É aí que entra a otimização: tem como ajustar o código para que esses divisores sejam encontrados de um modo mais rápido? Ou, em outras palavras: tem como otimizar o programa?

Em programação, a resposta para esse tipo de pergunta é quase sempre "sim", pois normalmente há inúmeras maneiras de se realizar uma mesma coisa - essa é uma das belezas da programação!

No caso dos divisores, vamos tomar como exemplo o 10: nós não precisamos dividir o 10 por todos os números naturais menores ou iguais a 10, pois depois de sua metade (o 5), não há mais divisores de 10 (a não ser ele próprio). Isso ocorre com todos os números maiores que 2. Assim, podemos reescrever o nosso código para que, ao chegar na metade do número, o laço de repetição `for` seja quebrado - com isso, vamos economizar muito, mas muito processamento (e também tempo!)

OTIMIZAÇÃO

O comando para quebrar um laço de repetição é o break. O novo código:

```
lista_div = []
numero = int(input("Informe o número: "))
for div in range (1, numero+1):
    if div > numero//2:
        break
    if numero % div == 0:
        lista_div.append(div)
print("A lista de disivores é:",lista_div)
```

Agora, assim que div ficar maior que a metade do número informado (if div > numero//2), o for é quebrado (break). O último ajuste que falta é adicionar o próprio número à lista de divisores. Então, ficamos finalmente com:

```
lista_div = []
numero = int(input("Informe o número: "))
for div in range (1, numero+1):
    if div > numero//2:
        break
    if numero % div == 0:
        lista_div.append(div)
lista_div.append(numero)
print("A lista de disivores é:",lista_div)
```

Testei aqui o número 135.848.900: sem otimização, o processo demorou cerca de 17 segundos; com otimização, 13 segundos aproximadamente.

MAIS SOBRE LISTAS

Para contar a quantidade de elementos de uma lista, basta usar a função len. No nosso programa anterior, poderíamos ter escrito no final:

```
print("Nº de divisores:",len(lista_div))
```

Isso vai retornar o número de elementos da lista.

Atividade 10: Ajuste o código anterior para que, no final, o programa diga se o número informado é primo ou composto (lembre-se de que um número é primo se possuir exatamente dois divisores; será composto se possuir mais de 2).

Além disso, podemos ordenar os elementos de uma lista com a função sort().

```
numeros = [3, 1, 5, 9]
print(numeros)
numeros.sort()
print(numeros)
```

Resultado:

```
[3, 1, 5, 9]
[1, 3, 5, 9]
```

BANCO DE DADOS COM LISTAS

Listas são objetos que podem armazenar textos, números e até mesmo outras listas: quando este último ocorre, temos um conceito muito parecido com o de matrizes em Matemática. Veremos isso agora e também como fazer um banco de dados simples por meio de listas.

Para declararmos uma lista dentro de outra, fazemos assim:

```
dados = [["Pedro","Renato"],[58,13]]
```

Alternativamente, podemos utilizar mais linhas para visualizarmos melhor:

```
dados = [
["Pedro","Renato"],
[58,13]
]
```

Agora, podemos utilizar índices para acessar as informações: o índice 0 (primeira lista dentro da lista) nos dá acesso aos nomes "Pedro" e "Renato"; o índice 1, aos números 58 e 13.

```
dados[0] -> ["Pedro","Renato"]
dados[1] -> [58, 13]
dados[0][0] -> Pedro
dados[0][1] -> Renato
dados[1][0] -> 58
dados[1][1] -> 13
```

BANCO DE DADOS COM LISTAS

De posse dessas informações, agora vamos montar nosso banco de dados para posteriormente manipulá-lo.

O banco de dados será composto pelo nome de um cliente, endereço e dívida (como um banco de dados de uma empresa).

```
dados = [
["Ana", "José", "Pedro"],
["Vig Calixto","Prof Balbino","Pau Maia"],
["11","25","74"],
[123,942,650]
]
```

Observe que a primeira lista contém os nomes dos clientes; a segunda, a rua; a terceira; o número da casa e, por fim, a última lista contém as dívidas de cada um.

Agora vamos escrever um código para exibir essas informações de uma maneira organizada. Lembrando que:

```
dados[0] -> lista de clientes
dados[1] -> lista de ruas
dados[0][1] -> nome do segundo cliente
dados[1][1] -> rua do segundo cliente
dados[3][2] -> dívida do terceiro cliente
```

E assim por diante.

Agora, nós podemos utilizar laços de repetição para manipular o nosso banco de dados conforme nossas necessidades.

Por exemplo, se escrevermos:

```
for i in range (len(dados[0])):
    print(dados[0][i])
```

O resultado será:

```
Ana
José
Pedro
```

```
print("As dívidas são:")
for i in range (len(dados[3])):
    print(dados[3][i])
```

O resultado será:

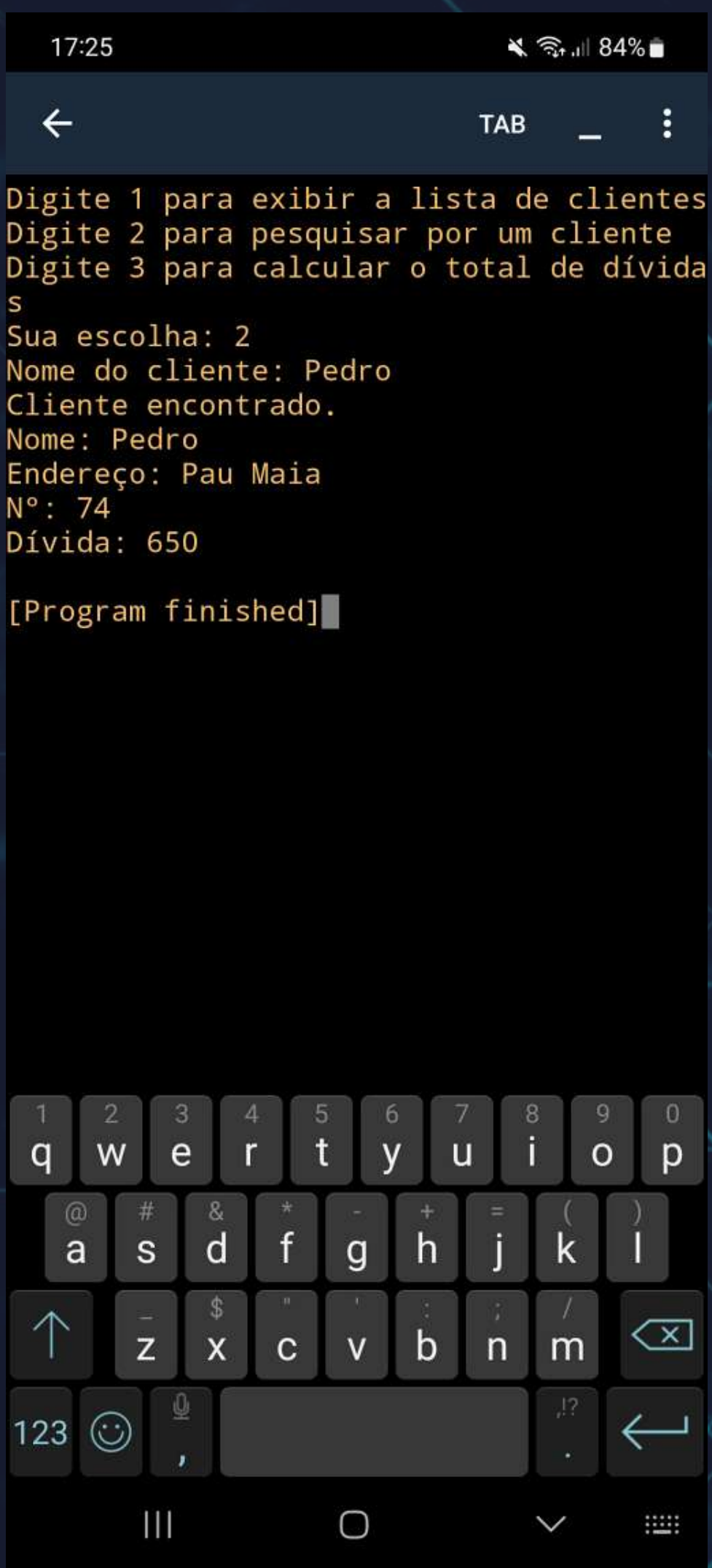
```
123
942
650
```

Agora vamos escrever nosso programa.

BANCO DE DADOS COM LISTAS

```
dados = [
["Ana", "José", "Pedro"],
["Vig Calixto","Prof Balbino","Pau Maia"],
["11","25","74"],
[123,942,650]
]
print("Digite 1 para exibir a lista de clientes")
print("Digite 2 para pesquisar por um cliente")
print("Digite 3 para calcular o total de dívidas")
escolha = int(input("Sua escolha: "))
if escolha == 1:
    for i in range(len(dados[0])):
        print(dados[0][i])
elif escolha == 2:
    cliente = input("Nome do cliente: ")
    for i in range (len(dados[0])):
        if dados[0][i] == cliente:
            print("Cliente encontrado.")
            print("Nome:",dados[0][i])
            print("Endereço:",dados[1][i])
            print("Nº:",dados[2][i])
            print("Dívida:",dados[3][i])
elif escolha == 3:
    dividas = 0
    for i in range (len(dados[3])):
        dividas = dividas + dados[3][i]
    print("Total:",dividas)
```

Abaixo, prints de seu funcionamento.



Atividade 11: Como se vê, nosso programa ainda precisa de polimentos: por exemplo, deve-se incluir o nome "reais" na exibição de dívidas; ademais, quando a opção 2 for utilizada e o cliente não for localizado, o que deve ocorrer? Fazer esses polimentos constitui-se a atividade 11.

LAÇO DE REPETIÇÃO WHILE

Já vimos o laço de repetição for. Agora veremos o while (enquanto). A principal diferença entre ambos é que, com o while, você precisa definir uma condição para a quebra da repetição. Isso é útil em diversas situações.

A sintaxe do while que utilizaremos é:

```
while True:
    comandos
    comandos
    condição para a quebra do while
```

Se nenhuma condição for estabelecida, o código se repete para sempre, o famoso loop infinito. Por exemplo:

```
n = 0
while True:
    print(n)
    n = n + 1
```

O código acima vai escrever os números 0, 1, 2, 3, 4, 5, ..., 1.000.000 e assim por diante, sem nunca parar, pois não estabelecemos nenhuma condição de quebra da repetição.

No entanto, com o ajuste abaixo:

```
n = 0
while True:
    print(n)
    n = n + 1
    if n > 1000:
        break
```

Ele vai printar na tela de 0 até 1000. Quando n for 1001, o break será ativado e o laço de repetição será quebrado.

E qual a vantagem disso? Imagine, por exemplo, que você deseja criar um programa de média onde o usuário vai inserindo quantas notas quiser. Neste caso, você pode usar o while. Vejamos:

```
notas = []
nota = 0
media = 0
soma = 0
print("Digite s quando finalizar")
while True:
    nota = input("Digite a nota: ")
    if nota == "s":
        break
    notas.append(float(nota))
for n in range (len(notas)):
    soma = soma + notas[n]
media = soma / len(notas)
print("A média foi", media)
```

Ao ser executado, o programa ficará solicitando as notas ao usuário até que ele digite "s". Enquanto vai digitando, as notas vão sendo armazenadas na lista "notas". O for que vem logo depois do while é para calcular a soma de todas as notas que estão dentro da lista. Ademais, note que utilizei a função float dentro do comando append: isso foi necessário, pois o campo de entrada das notas está como texto (string), já que o usuário poderá digitar "s" para sair. Dessa forma, a função float converte o texto em número antes do append adicioná-lo à lista.

Um dos polimentos que falta nesse programa é o seguinte: se ele digitar "s" logo de primeira, sem inserir nota alguma, haverá um erro no cálculo da média. Com um if adicional é possível corrigir isso (deixo para você fazer).

ATIV. MEDIANA

Dada uma lista de números ordenada, a mediana é o número que ocupa a posição central na lista (caso a lista tenha um número ímpar de elementos) ou a média aritmética dos dois números centrais (caso a lista tenha um número par de elementos). Por exemplo:

Considere a lista (1, 8, 2, 9, 7). Vamos determinar a mediana.

O primeiro passo é ordenar a lista. Em ordem crescente, teremos a lista (1, 2, 7, 8, 9). Agora, como há um número ímpar de elementos, então existe um elemento central, que é o número 7. Assim, a mediana é 7.

Agora, considere a lista (7, 3, 1, 4).

Vamos ordenar: (1, 3, 4, 7). Como há um número par de elementos, não existe um número do meio, mas dois: o 3 e 4. Vamos fazer a média aritmética deles dois: (3 + 4) / 2 = 3,5. Dessa forma, a mediana será 3,5.

Atividade 12: Elabore um programa que receba do usuário uma lista qualquer de números e depois retorne a mediana. Dicas: [1] A função para ordenar listas é o sort(); [2] Um número é par se o resto da divisão por 2 for igual a zero.

MÓDULOS

Você pode importar para seu projeto bibliotecas/módulos, tanto seus como de outrem. Não ensinarei como criá-los, mas como utilizar algumas das várias bibliotecas disponíveis para Python.

Ao importar bibliotecas, você passa a ter novos comandos e funções para utilizar, o que nos dá muitas possibilidades. Por exemplo, na biblioteca time temos comandos temporais, incluindo comando para dar pausa; na biblioteca math, temos seno, cosseno, tangente, fatorial e vários outros; na biblioteca numpy, temos comandos para se trabalhar com matrizes matemáticas; na pylab, temos comandos para construir gráficos; e assim por diante.

Para importar uma biblioteca, nós utilizamos o comando import. Como exemplo, vou importar o comando sleep, que serve para dar pausa. O comando sleep está na biblioteca time, então:

```
import time
print("Bom dia!")
time.sleep(2)
print("Como vai?")
```

O programa acima vai exibir a mensagem "Bom dia!"; vai aguardar 2 segundos (comando sleep) e, só depois, a pergunta "Como vai?" aparecerá.

Ao usar a sintaxe acima para o comando import, é assim que funciona os comandos: "biblioteca.comando". No entanto, Python é muito flexível e há outras maneiras de importar um comando ou funções:

```
from time import sleep
print("Bom dia!")
sleep(2)
print("Como vai?")
```

No caso acima, nós utilizamos o from time import sleep (de time traga sleep): neste caso, o comando pode ser utilizado sem o nome da biblioteca. Aqui, nós importamos apenas o comando sleep.

E se você quiser renomear o comando ao importá-lo, você digita:

```
from time import sleep as pausa
print("Bom dia!")
pausa(2)
print("Como vai?")
```

Há ainda outra forma muito utilizada de importação, não obstante, por ora, ficaremos com estas.

Importante: Cada forma de importação tem suas vantagens e desvantagens. Com a primeira, você precisa ficar digitando o nome da biblioteca antes dos comandos (mas isso evita a confusão entre comandos de mesmo nome em diferentes bibliotecas); a principal vantagem da segunda e da terceira é que você está trazendo apenas um comando (e não a biblioteca inteira), no entanto, deve tomar cuidado ao importar, de bibliotecas distintas, comandos que possuem a mesma escrita.

SENO, COSSENO E TANGENTE

Vamos explorar um pouco o módulo math.

```
import math
n = 10
print(math.sqrt(n))
print(math.sin(n))
print(math.factorial(n))
```

O programa acima retornará a raiz quadrada de 10; o seno de 10 e 10 fatorial. Bastante simples, não?

Para o caso do seno, a entrada, por padrão, é em radianos. Caso queiramos em graus, colocamos:

```
import math
n = math.radians(10)
print(math.sin(n))
```

Alternativamente, se sabemos que vamos utilizar apenas esses dois comandos do módulo math, podemos fazer:

```
from math import sin, radians
n = radians(10)
print(sin(n))
```

Atividade 13: Crie um programa que peça ao usuário a medida de um ângulo, que pergunte se será em grau ou radiano e que, por fim, exiba o seno (sin), o cosseno (cos) e a tangente (tan) do ângulo.

HUMANIZAÇÃO DA MÁQUINA

Existe um comando muito utilizado para se criar comportamentos aleatórios em games e programas, dando-lhes um aspecto mais humano: é o comando random, que serve para gerar números aleatórios. Em Python, está no módulo random. Abaixo, já trago um exemplo:

```
import random
numero = random.randint(0,10)
print(numero)
```

Ao executar a primeira vez, o resultado aqui foi:

1

Depois de mais uma execução, o resultado foi:

5

Ou seja: o programa acima gera um número aleatório dentro do intervalo fechado que vai de 0 até 10. O comando utilizado foi o randint (gerar número inteiro aleatório).

As aplicações disso são inúmeras: desde importantes simulações no campo da probabilidade até o desenvolvimento de programas que devem fazer tabelas ou horários, passando pela programação do comportamento de carros ou inimigos em games - é uma espécie de humanização da máquina.

Vejamos um outro exemplo. Queremos que nosso aplicativo, ao ser iniciado, cumprimente o usuário - mas queremos programar vários cumprimentos possíveis.

```
import random
cumprimentos = ["Olá!","Como vai?","Que dia lindo!","Eae","Seja bem-vindo!"]
pega_cump = random.randint(0,4)
print(cumprimentos[pega_cump])
```

Pronto! Toda vez que o programa acima for executado, um número aleatório será gerado; esse número, por sua vez, será aplicado como índice à lista "cumprimentos", pegando um dos cumprimentos escritos e imprimindo na tela.

CARA OU COROA?

Voltando à Matemática, vamos utilizar o módulo random para fazer simulações envolvendo probabilidade. A vantagem de se utilizar programação para isso é que, para algumas simulações, nós perderíamos muito tempo, talvez anos, para podermos realizá-las na prática. Aqui nós levaremos segundos!

Simulação envolvendo lançamento de moedas:

```
import random
caras = 0
coroas = 0
lancar = int(input("Digite o número de lançamentos: "))
for i in range (lancar):
    cara_coroa = random.randint(0,1)
    if cara_coroa == 0:
        caras = caras + 1
    else:
        coroas = coroas + 1
print("Caras:", caras)
print("Coroas:", coroas)
```

Ao ser executado, o programa perguntará ao usuário o número de lançamentos. Por exemplo, vou executar aqui e colocar 500. O resultado foi:

```
Caras: 257
Coroas: 243
```

CARA OU COROA?

Ou seja, facilmente nós fizemos a simulação de 500 lançamentos de moeda! Esse tipo de simulação é feita (não apenas com moedas, claro) para determinar se um certo negócio (cassinos, por exemplo) ou jogo de azar poderão ou não retornar lucro a longo prazo e quais ajustes você deve fazer no negócio ou no jogo para conseguir um bom lucro sem necessariamente espantar os jogadores.

Com simulações em Python, vou agora lhe explicar um teorema fundamental da teoria das probabilidades: a Lei dos Grandes Números.

LEI DOS GRANDES NÚMEROS

Sabemos que a probabilidade de dar cara (ou coroa) no lançamento de uma moeda é de 50%. Isso não quer dizer, claro, que, ao jogarmos 10 vezes uma moeda, teremos 5 caras e 5 coroas. Vamos, todavia, realizar simulações.

Para tanto, precisaremos trabalhar com porcentagem. Vou adaptar o nosso código anterior para que, além dele exibir o número de caras e o número de coroas, ele também mostre a porcentagem de cada. O ajuste é simples, basta acrescentar duas linhas:

```
import random
caras = 0
coroas = 0
lancar = int(input("Digite o número de lançamentos: "))
for i in range (lancar):
    cara_coroa = random.randint(0,1)
    if cara_coroa == 0:
        caras = caras + 1
    else:
        coroas = coroas + 1
print("Caras:", caras)
print("Coroas:", coroas)
print("% de caras:", caras/lancar*100)
print("% de coroas:", coroas/lancar*100)
```

Farei agora simulações. Primeiro, vou colocar 10 lançamentos:

Caras: 6
Coroas: 4
% de caras: 60.0
% de coroas: 40.0

10 novamente:

Caras: 4
Coroas: 6
% de caras: 40.0
% de coroas: 60.0

Agora 100 lançamentos:

Caras: 50
Coroas: 50
% de caras: 50.0
% de coroas: 50.0

100 novamente:

Caras: 44
Coroas: 56
% de caras: 44.0
% de coroas: 56.00000000000001

Observe que, até agora, os resultados são muito discrepantes quando consideramos os 50% *que deveriam sair*. Vamos continuar!

500 lançamentos:

Caras: 256
Coroas: 244
% de caras: 51.2
% de coroas: 48.8

1000 lançamentos:

Caras: 499
Coroas: 501
% de caras: 49.9
% de coroas: 50.1

50.000 lançamentos:

Caras: 25026
Coroas: 24974
% de caras: 50.052
% de coroas: 49.948

Observe que, claramente, quando aumentamos o número de lançamentos, a porcentagem tende a 50% (e tanto mais se aproximará de 50% quanto mais lançamentos forem feitos). Essa é, com efeito, a Lei dos Grandes Números! É ela quem garante o lucro de um cassino a longo prazo.

ATIV.: SIMULAÇÕES

Atividade 14: Crie um programa para fazer simulações, idênticas as que acabamos de fazer, só que usando dados de 6 faces.

Atividade 15: Crie um código e faça simulações para resolver o seguinte problema: num jogo de sorteio aleatório composto por 10 números (1 até 10), se sair de 1 a 6, você ganha; se sair de 7 até 10, o seu adversário ganha. Já vimos que, em poucas rodadas, os resultados serão um tanto aleatórios e você não terá garantias de ser o vencedor. A partir das simulações que você fizer, responda: com 5 rodadas, você sempre será o vencedor? E com 25 rodadas? E com 5000? Estabeleça um número mínimo de rodadas para que você sempre saia o vencedor nesse jogo.

SORTEANDO GRUPOS

Há muitas outras aplicações para os números aleatórios além das vistas até aqui. Por exemplo, vamos agora criar um programa no qual você insere alguns nomes e ele os distribui em equipes aleatoriamente.

Utilizaremos bastante listas neste caso: precisamos de uma lista que receba os nomes; precisamos de uma outra lista que receberá os nomes embaralhados; uma última lista organizará os grupos e será printada na tela. A primeira parte ficará assim:

```
import random
nomes = ["a","b","c","d"]
qtd = len(nomes)
embaralha = []
equipes_finais = []
membros = 2
for i in range (qtd):
    pos = random.randint(0,len(nomes)-1)
    embaralha.append(nomes[pos])
    nomes.remove(nomes[pos])
```

O código acima pega os elementos da lista "nomes" e os embaralha, adiciondo-os na lista "embaralha". A variável "pos" recebe sempre um número aleatório (dentro da quantidade de elementos de "nomes"). O comando remove é utilizado para retirar da lista "nomes" aqueles que já foram sorteados.

Além disso, dentro do randint tive que colocar len(nomes)-1, pois, no referido comando, o limite superior informado pode ser atingido (diferentemente do range).

Agora, precisamos separar os grupos e jogá-los na lista "equipes_finais". A quantidade por grupo é informada na variável "membros". A segunda parte ficará assim:

```
for i in range (0, qtd, membros):
    if membros == 2:

equipes_finais.append([embaralha[i],
embaralha[i+1]])
    elif membros == 3:

equipes_finais.append([embaralha[i],
embaralha[i+1],
embaralha[i+2]])
    elif membros == 4:

equipes_finais.append([embaralha[i],
embaralha[i+1],
embaralha[i+2],
embaralha[i+3]])
print("As equipes são:")
print(equipes_finais)
```

Prontinho! Caso queira grupos com 5 ou 6 membros, basta adicionar mais elif seguindo o padrão acima.

SORTEANDO GRUPOS

Sobre o comando range, tivemos uma variação nele: range(0, qtd, membros). Os dois primeiros valores numéricos são o início e o fim do intervalo; o terceiro valor é o passo, o *pulo* - assim, se colocarmos range(0,10,2), ele vai gerar um intervalo de 0 até 9 pulando de dois em dois. Abaixo, prints do funcionamento desse programa.

```
import random
nomes = ["Ana","Beatriz","Carla",
"José","Maria","Neuza","Samara",
"Silmara","Zenildo"]
qtd = len(nomes)
embaralha = []
equipes_finais = []
membros = 3
for i in range (qtd):
    pos = random.randint(0,len(nomes
    embaralha.append(nomes[pos])
    nomes.remove(nomes[pos])
for i in range (0, qtd, membros):
    if membros == 2:
        equipes_finais.append([embar
embaralha[i+1]])
    elif membros == 3:
        equipes_finais.append([embar
embaralha[i+1],
embaralha[i+2]])
    elif membros == 4:
        equipes_finais.append([embar
embaralha[i+1],
embaralha[i+2],
embaralha[i+3]])
print("As equipes são:")
print(equipes_finais)
```

```
As equipes são:
[['Silmara', 'Maria', 'Zenildo'], ['Beat
riz', 'Ana', 'Samara'], ['Carla', 'José'
, 'Neuza']]

[Program finished]
```

CRIANDO FUNÇÕES

O conceito de função em programação não é o mesmo que o utilizado em Matemática. Em programação, podemos criar funções para resolvermos muitos problemas, dentre os quais citarei dois: organizar códigos grandes e otimizar o tempo do programador - não se restringindo apenas a isso, porquanto as funções possuem vastas aplicações e são importantíssimas em programação.

Imagine que um determinado bloco de código apareça múltiplas vezes ao longo do seu programa. Para estes casos, você pode simplesmente criar uma função e chamar esse código por meio da função criada.

O comando para se criar função é o def:

```
def nome_da_função(argumentos):
    código
    código
    código
    return valor
```

Sobre argumentos e return, falarei daqui a pouco. Vamos começar com um exemplo mais simples.

CRIANDO FUNÇÕES

```
from time import sleep as delay
def cumprimento():
    print("Olá!")
    delay(1)
    print("Bom dia!")
    delay(1)
    print("Como vai?")

cumprimento()
```

Com o código acima criamos uma função chamada "cumprimento": toda vez que a chamarmos digitando cumprimento(), aparecerá "Olá!", "Bom dia!" e "Como vai?", com uma pausa entre eles.

Podemos criar funções de utilidades variadas. A imaginação é o limite.

```
from random import randint
def sorteio():
    num = randint(0, 10)
    print(num)

sorteio()
sorteio()
```

O programa acima printará na tela dois números aleatórios de 0 a 10.

```
def impares():
    for i in range (100):
        print(i*2 + 1)
print("Os 100 primeiros ímpares são:")
impares()
```

Já este acima printará na tela os 100 primeiros números ímpares. E toda vez que digitarmos impares(), ele fará isso.

É possível enviar valores para as funções e obter valores delas: é aí onde entram os argumentos e o comando return.

Na função impares() acima, imagine que você queira definir, ao chamar a função, a quantidade de números a ser impressa na tela - você então usa argumentos:

```
def impares (quantidade):
    for i in range (quantidade):
        print(i*2 + 1)

print("Os 100 primeiros ímpares são:")
impares(100)
print("Os 300 primeiros ímpares são:")
impares(300)
```

Aqui, o número que você coloca dentro dos parênteses ao chamar a função "impares" é recebido pelo argumento "quantidade". Com isso, você manipula esse número dentro da função.

E o comando return?

Se sua função precisar retornar um valor numérico, por exemplo, você pode utilizá-lo.

No programa abaixo, a função "quadrado" retorna o quadrado do número informado como argumento:

```
def quadrado(num):
    return num**2
a = quadrado(6)
print("O quadrado de 6 é:")
print(a)
```

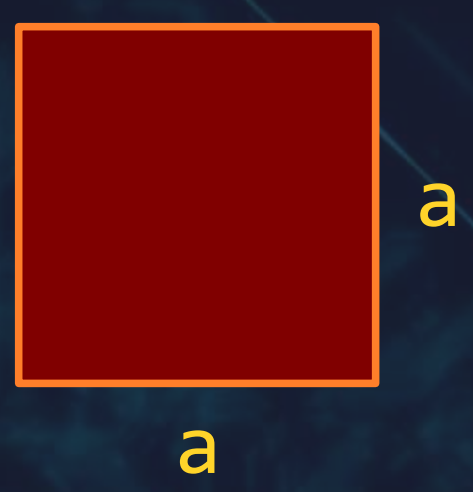
E este aqui calcula e retorna a média de dois números ao chamarmos a função "media" com dois números dentro:

```
def media(n1, n2):
    return (n1 + n2) / 2
print(media(15, 4))
print(media(11, 12))
c = media(20, 30)
print(c)
```

ATIV.: ÁREA

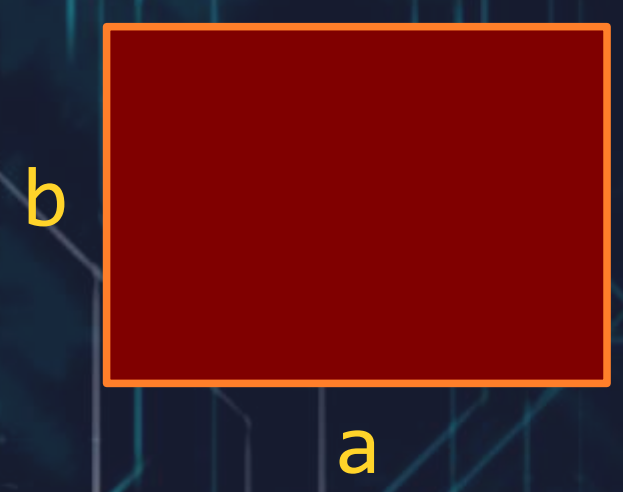
Mais adiante, na exploração do módulo turtle (tartaruga), nós utilizaremos bastante funções. Por ora, para encerrarmos este capítulo, vamos realizar algumas atividades envolvendo funções e geometria. Abaixo, temos algumas fórmulas para se calcular área de figuras planas.

Quadrado



$$A = a \cdot a = a^2$$

Retângulo

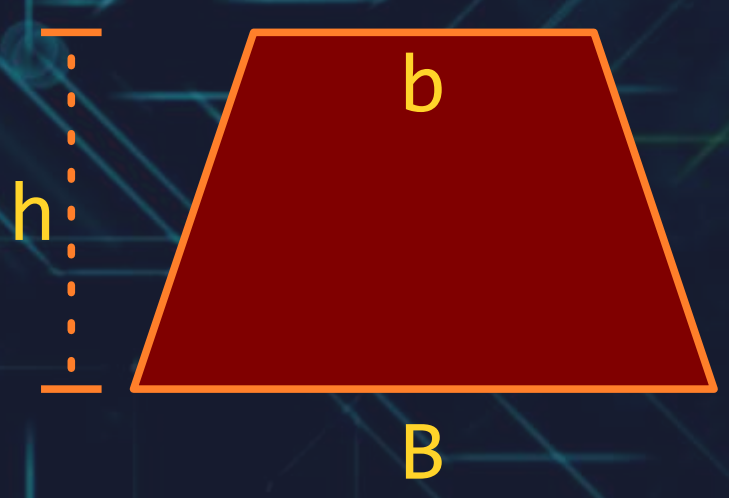


$$A = a \cdot b$$

Losango

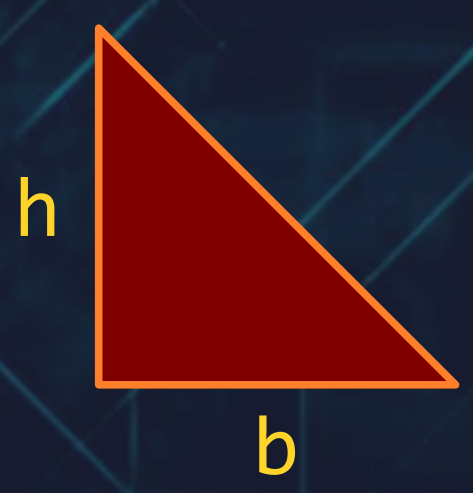
$$A = \frac{a \cdot b}{2}$$

Trapézio



$$A = \frac{(B+b) \cdot h}{2}$$

Triângulo



$$A = \frac{b \cdot h}{2}$$

Círculo

$$A = \pi \cdot r^2$$

ATIV.: ÁREA

Uma função para retornar a área do losango poderia ser:

```
def area_losango(a, b):
    return (a * b) / 2

print(area_losango(4, 9))
```

Atividade 16: Crie uma função pra retornar a área do:

a) Retângulo.

b) Trapézio.

c) Triângulo.

d) Círculo (utilize pi = 3.14).

O MÓDULO TURTLE

Existe uma famosa linguagem de programação criada há algumas décadas para ser utilizada nas escolas: é a linguagem LOGO. Trata-se de uma tartaruga que deve ser controlada: por onde ela passa, deixa um risco - com isso, é possível aprender alguns princípios de programação criando desenhos.

Disponível em
https://www.pngegg.com/pt/png-cjpam

A tartaruga obedece a alguns comandos simples, tais como "vá para frente 60 passos" (forward), "vire à esquerda 45 graus" (left) e assim por diante. A partir dessa simplicidade, você consegue criar figuras complexas.

Python possui um módulo, chamado turtle, com o qual é possível criar figuras com a tartaruga, e esse é o importante tema desta parte.

O MÓDULO TURTLE

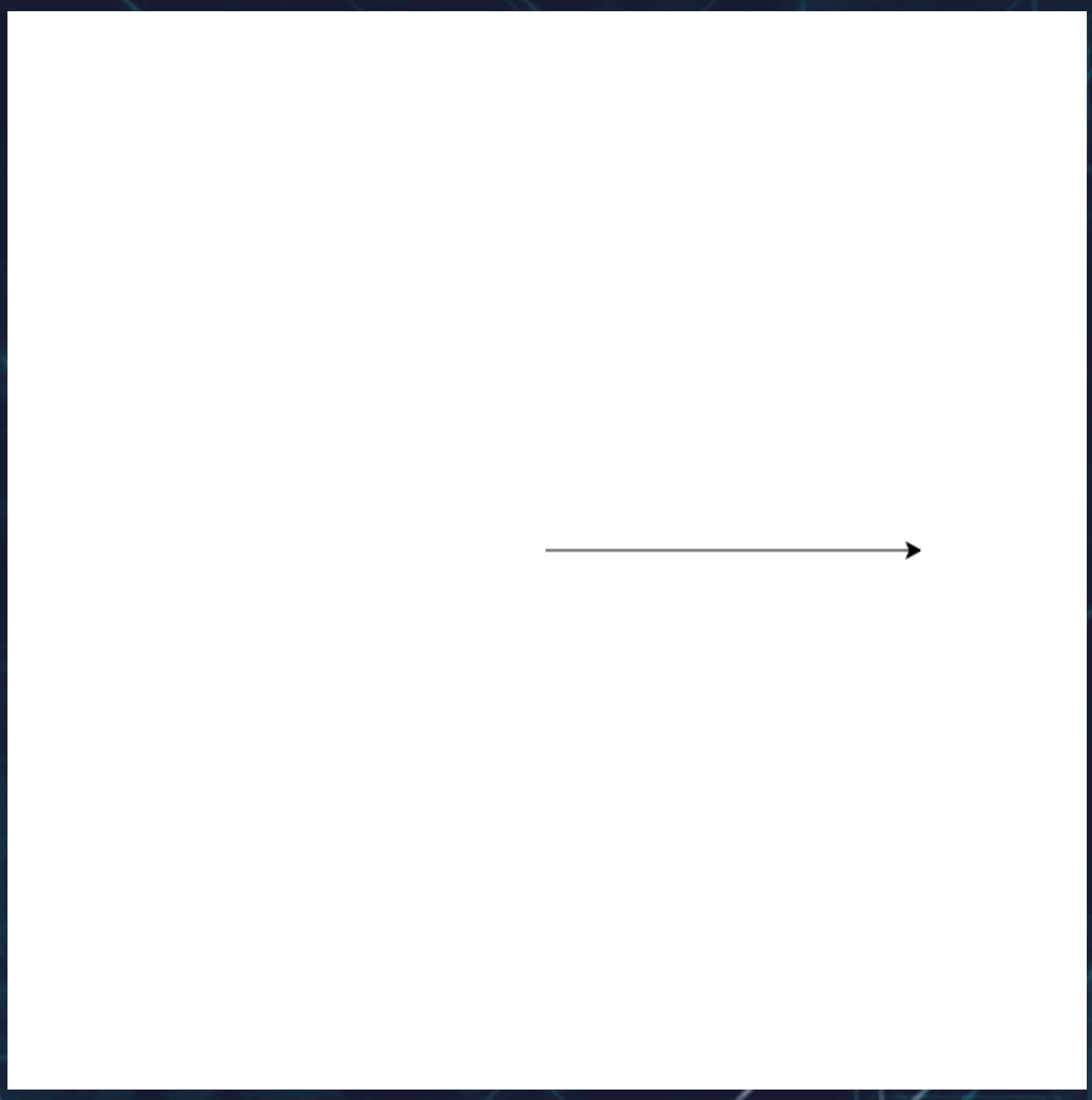
Para importar o módulo, basta escrever:

```
import turtle
```

Depois disso, você já pode criar uma tartaruga e começar a movimentá-la pela tela via comandos. A criação da tartaruga é feita por meio do comando "turtle.Turtle()". Abaixo, já mostro como importar, como criar uma tartaruga chamada "pedro" e como movimentá-la para frente 250 passos:

```
import turtle
pedro = turtle.Turtle()
pedro.forward(250)
turtle.done()
```

O comando final "turtle.done()" serve para não sair do programa despois que a tartaruga se movimenta, ou seja, para podermos ver o desenho. A seguir, você pode ver um print do resultado.



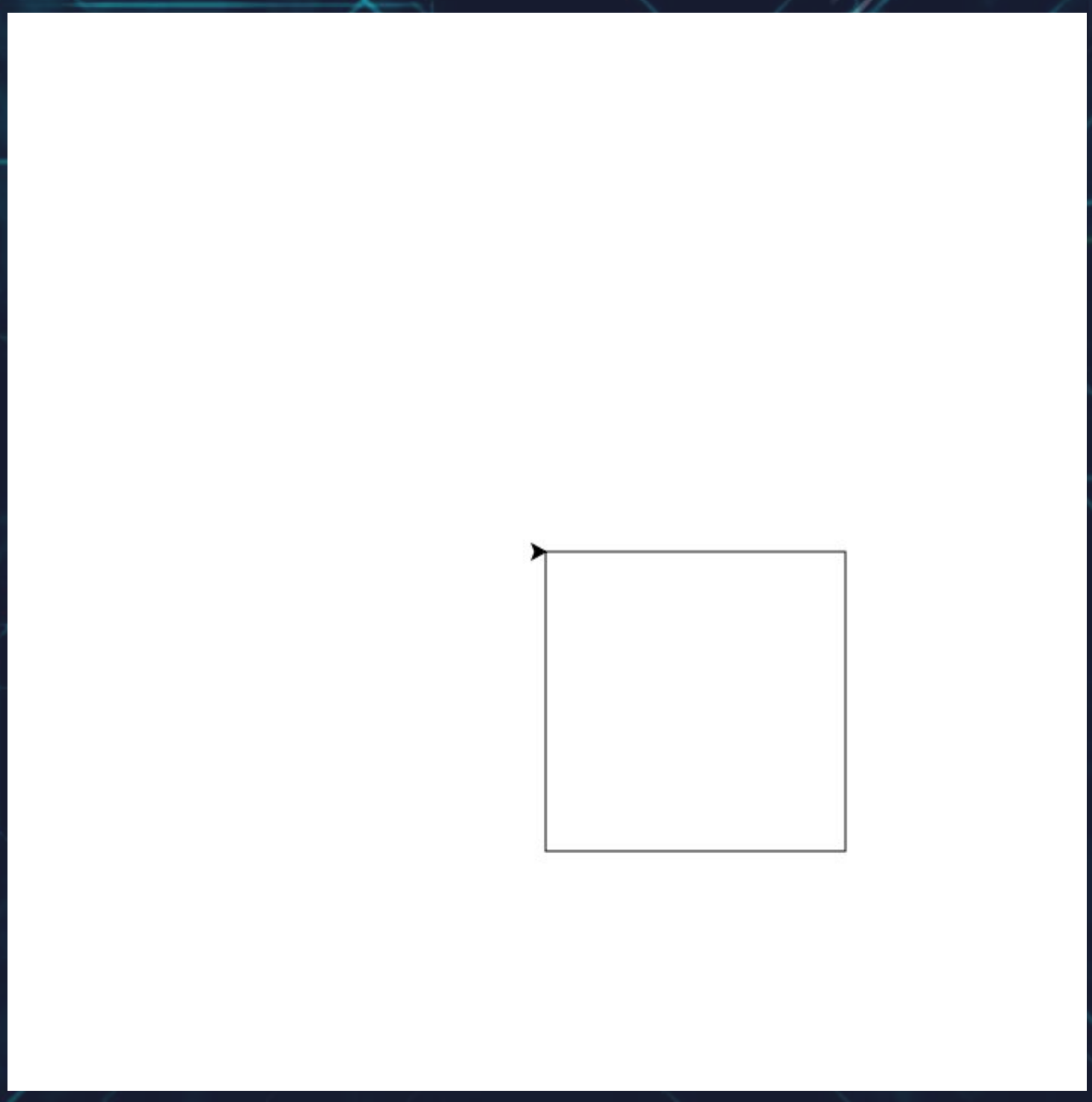
Note que os 250 passos que colocamos dentro do comando forward significaram 250 pixels na tela, que por sua vez funciona como um Plano Cartesiano, no qual a tartaruga inicia no centro, no ponto (0,0). Assim, a unidade de medida dos passos é pixels. Além disso, no lugar da tartaruga, há uma seta: daqui a pouco aprenderemos a configurar a parte visual do nosso programa.

Já usamos o comando forward. Para movimento, ainda temos como comandos básicos o backward (para trás), o right (virar para a direita) e o left (virar para a esquerda). O código a seguir cria um quadrado de lado 200 pixels.

O MÓDULO TURTLE

```
import turtle
pedro = turtle.Turtle()
pedro.forward(200)
pedro.right(90)
pedro.forward(200)
pedro.right(90)
pedro.forward(200)
pedro.right(90)
pedro.forward(200)
pedro.right(90)
turtle.done()
```

Lembrando que um quadrado possui lados de mesma medida e quatro ângulos internos medindo 90 graus. O resultado:



Atividade 17: Crie um triângulo equilátero com lado medindo 250 pixels.

Atividade 18: Crie um pentágono regular com lado medindo 150 pixels.

Antes de continuarmos, vamos ver como configurar o ambiente à nosso gosto.

ESTILO, COR E TAMANHO

Seja uma tartaruga chamada "eu", temos:

```
eu.shape("forma") -> altera o estilo
eu.color("cor") -> altera a cor
eu.pencolor("cor") -> cor do traço
eu.turtlesize(tamanho) -> altera o tamanho
eu.width(espessura) -> espessura do traço
turtle.bgcolor("cor") -> cor do fundo
```

No lugar de "forma", devemos colocar "square", "circle" ou "turtle" - para termos um quadrado, um círculo ou uma tartaruga.

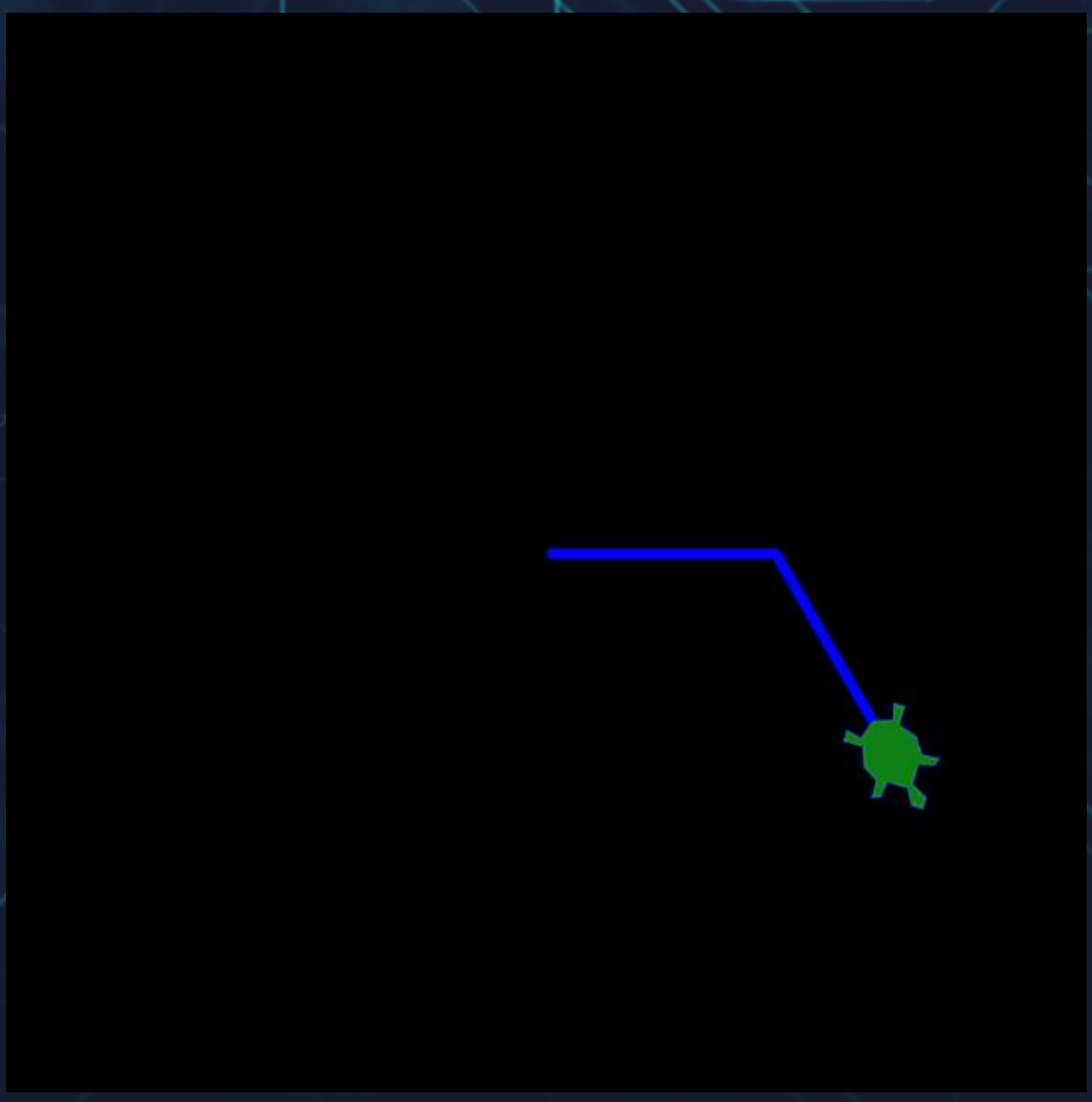
Já no lugar de "cor", devemos colocar "blue", "red", "green", "light blue", "dark green", "orange" etc - para termos a cor que quisermos.

O MÓDULO TURTLE

Pois bem, vamos criar o início de um paralelogramo com uma tartaruga verde de tamanho 4, que desenha com linhas azuis de espessura 10 em um fundo preto:

```
import turtle
roman = turtle.Turtle()
roman.shape("turtle")
roman.turtlesize(4)
roman.color("green")
roman.pencolor("blue")
roman.width(10)
turtle.bgcolor("black")

roman.forward(300)
roman.right(60)
roman.forward(200)
turtle.done()
```



Atividade 19: Termine o paralelogramo que comecei a fazer na página anterior.

A grande jogada da linguagem LOGO é que ela, de fato, permite introduzir lógica de programação com atividades envolvendo geometria. Por exemplo, já vimos que, ao fazer um quadrado, um determinado bloco de comandos se repete 4 vezes. Dessa forma, podemos utilizar o comando for:

```
import turtle
pedro = turtle.Turtle()
for i in range (4):
    pedro.forward(220)
    pedro.right(90)
turtle.done()
```

Prontinho! Um quadrado está feito! Para fazermos um triângulo, basta trocarmos o 4 por um 3 e ajustarmos o ângulo de giro (360 graus / 3 lados = 120 graus):

```
import turtle
pedro = turtle.Turtle()
for i in range (3):
    pedro.forward(220)
    pedro.right(120)
turtle.done()
```

Além disso, uma abertura para variáveis quase sempre se apresenta. Ainda sobre o exemplo anterior, podemos declarar uma variável que vai receber o número de lados do polígono e realizar os ajustes necessários:

```
import turtle
lados = 3
pedro = turtle.Turtle()
for i in range (lados):
    pedro.forward(220)
    pedro.right(360/lados)
turtle.done()
```

Com esse código, podemos desenhar qualquer polígono regular, bastando para isso mudarmos o valor de "lados".

Atividade 20: Desenhe:

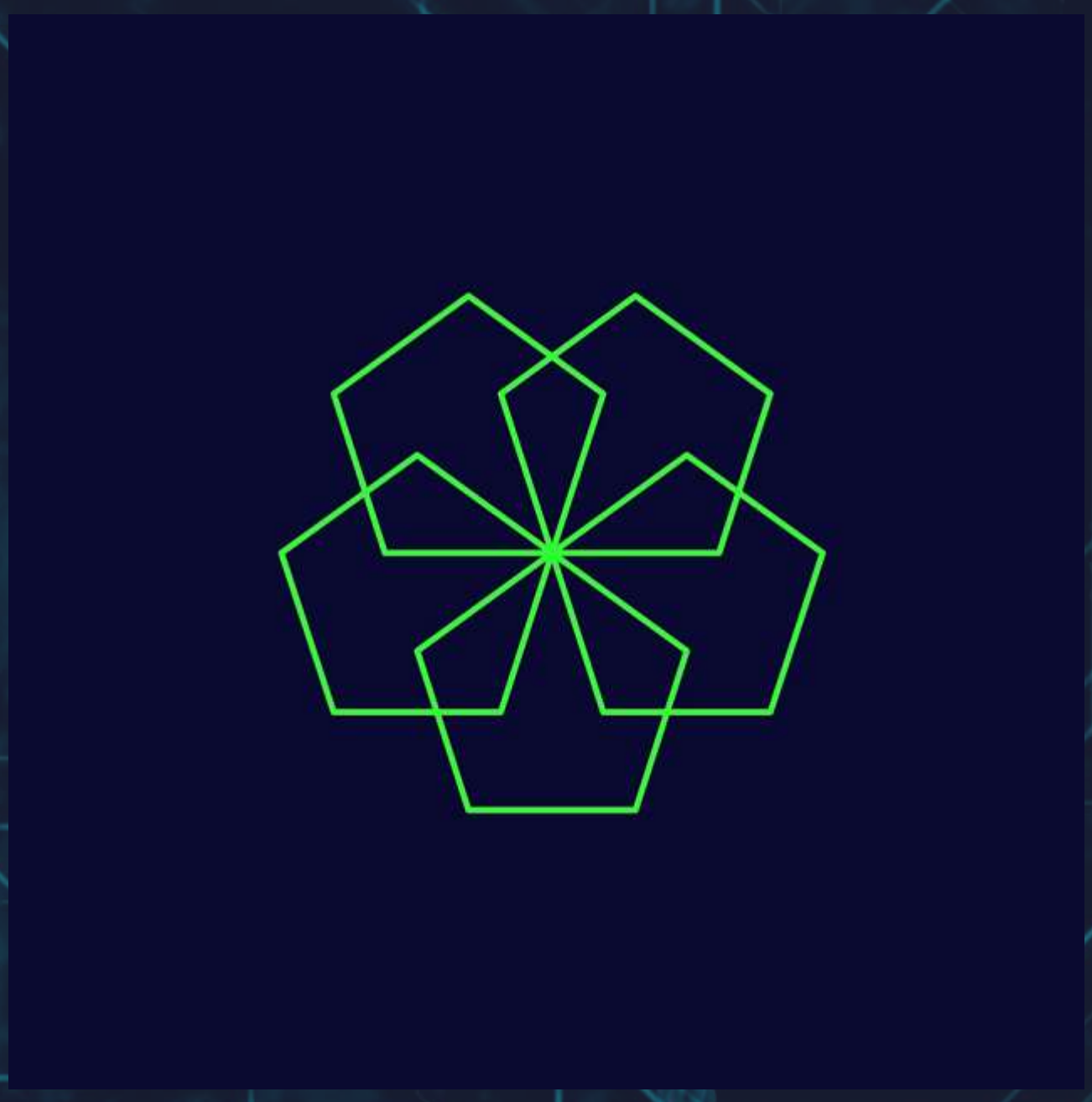
a) Um pentágono azul em um fundo preto.

b) Um octógono verde em um fundo branco, com uma linha de espessura 5.

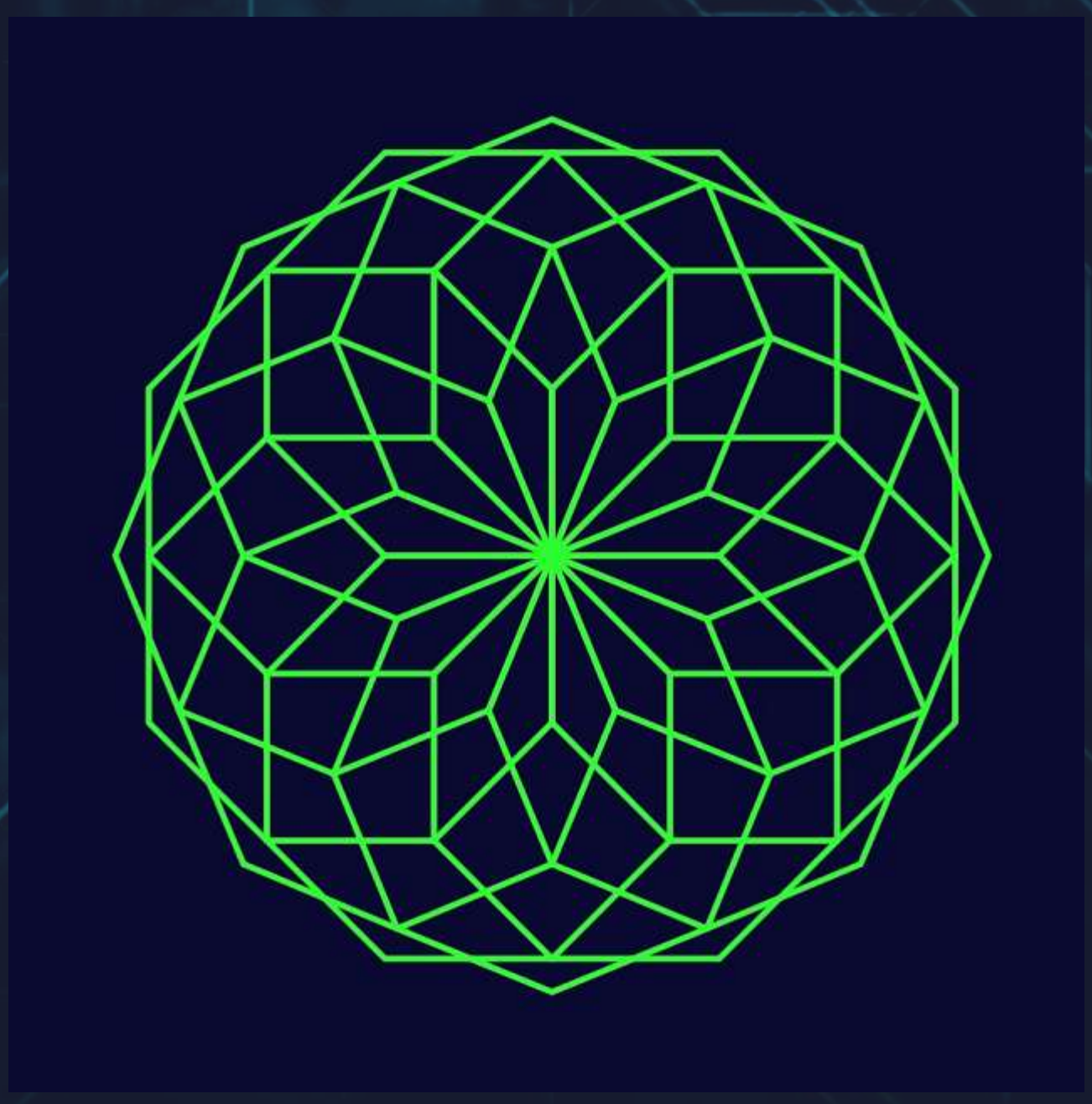
c) Um hexágono amarelo, de lado 150 pixels, com espessura da linha igual a 10, em um fundo preto com uma tartaruga azul.

Atividade 21: Utilizando um "for" dentro do outro, crie um programa que possa desenhar facilmente as seguintes figuras:

5 pentágonos:



16 octógonos:



No módulo turtle, como já deve ter ficado claro, a criação de funções é muito útil. O esquema para criarmos é o mesmo já visto no capítulo anterior. Abaixo, mostro como criar uma função chamada "quadrado".

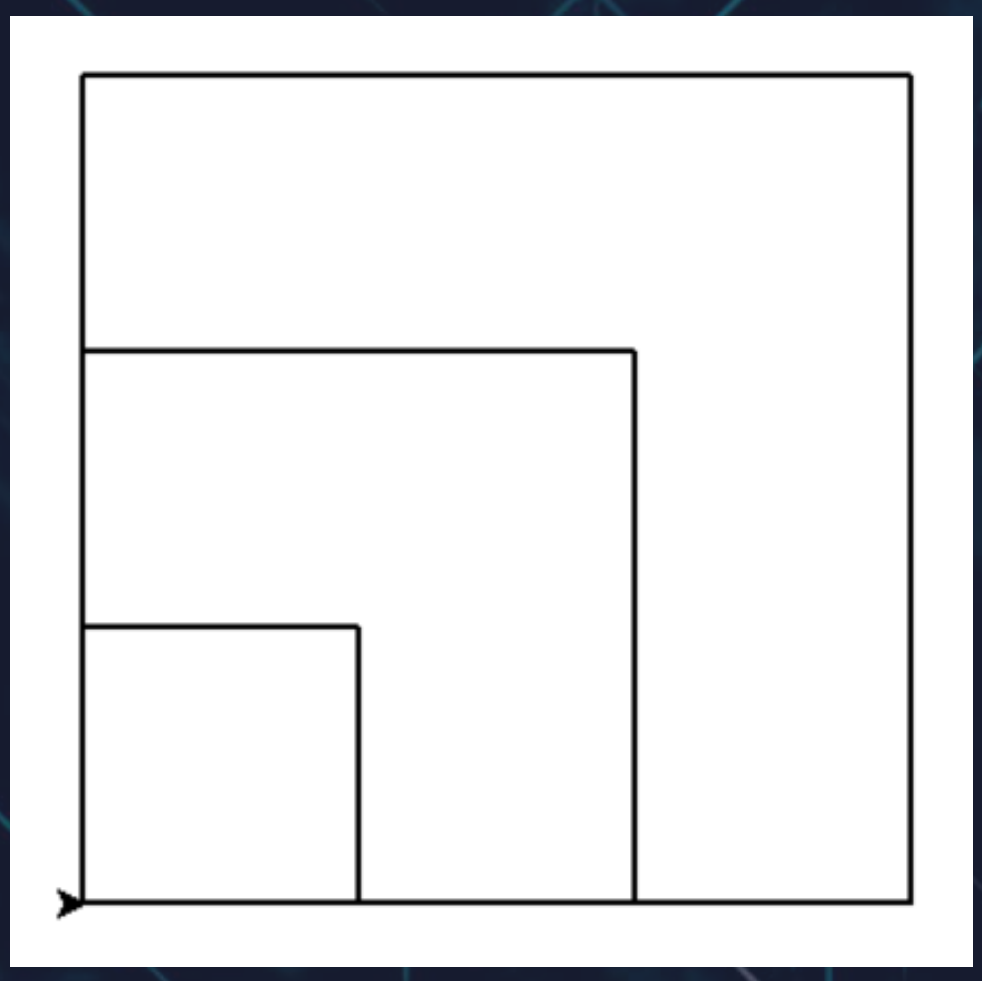
```
import turtle
pedro = turtle.Turtle()
def quadrado():
    for i in range(4):
        pedro.forward(100)
        pedro.left(90)
quadrado()
turtle.done()
```

E se quisermos utilizar argumentos para uma função mais completa:

```
import turtle
pedro = turtle.Turtle()
def quadrado(tam_lado):
    for i in range(4):
        pedro.forward(tam_lado)
        pedro.left(90)
quadrado(100)
quadrado(200)
quadrado(300)
turtle.done()
```

O programa acima desenha três quadrados: um de lado 100, um de lado 200 e um terceiro de lado 300.

Abaixo, print:



Atividade 22: Crie uma função de nome "regular" que desenhe qualquer polígono regular. A função deve possuir argumentos, quais sejam: número de lados, tamanho do lado e cor. Assim, se for colocado regular(6, 200, "blue"), deve ser desenhado um hexágono, com lado medindo 200 e na cor azul.

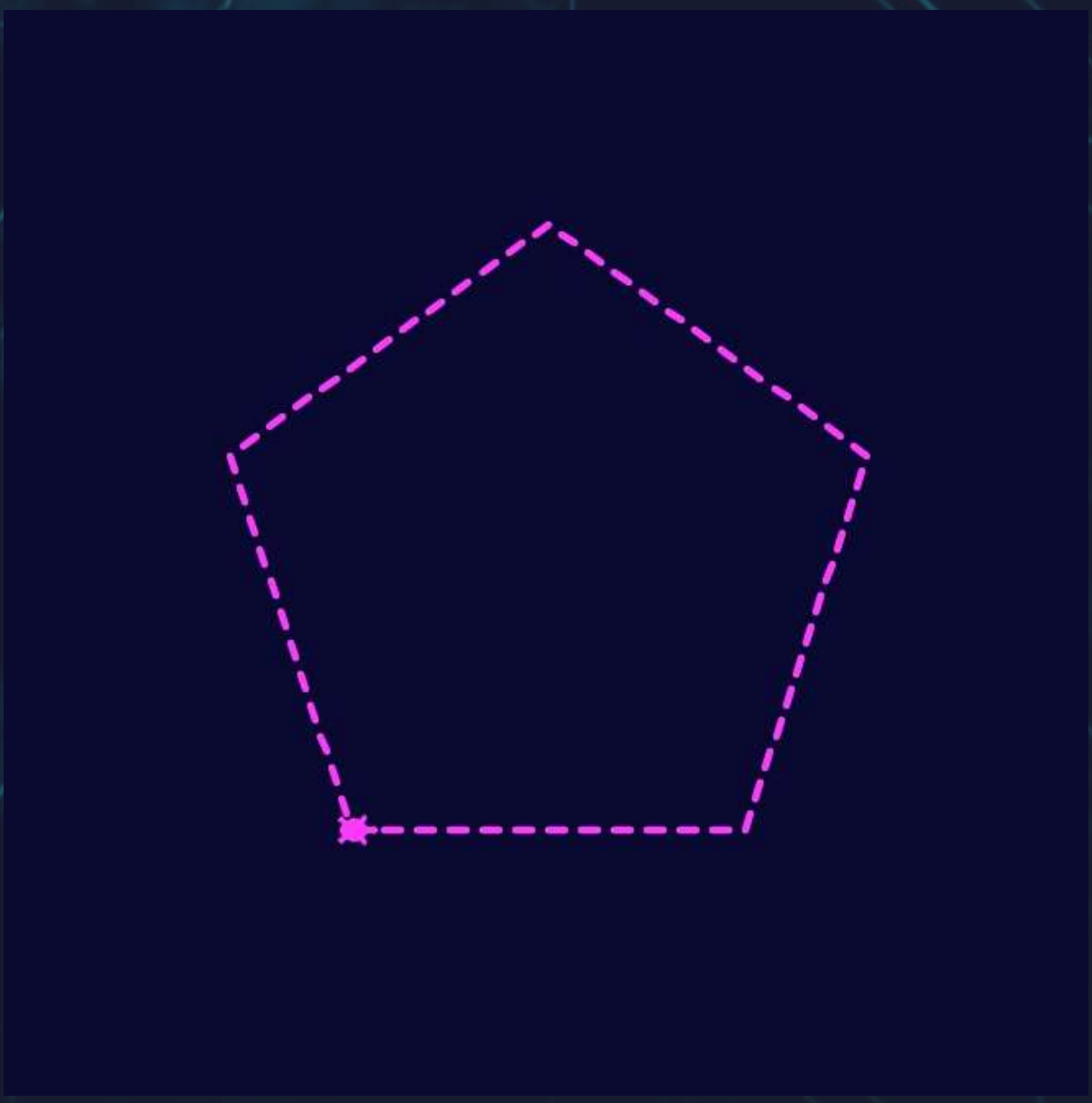
Há variados comandos no módulo turtle, muitos dos quais não veremos aqui. Porém, ainda quero explorar mais três comandos.

Com a função penup(), você determina que a tartaruga levante o lápis, isto é, ela se moverá sem riscar; já o comando pendown() faz com que ela baixe o lápis, fazendo com que o traço volte a aparecer na tela conforme a tartaruga se movimenta.

Para exemplificar, o código abaixo cria dois traços de 100 pixels separados por um espaço vazio de 50 pixels:

```
import turtle
pedro = turtle.Turtle()
pedro.width(2)
pedro.forward(100)
pedro.penup()
pedro.forward(50)
pedro.pendown()
pedro.forward(100)
turtle.done()
```

Atividade 23: Crie uma função que, informado um determinado tamanho como argumento, faça linhas tracejadas. Depois, utilize-a para desenhar o seguinte pentágono:



O terceiro comando que ainda quero explorar é o goto(): com ele, é possível posicionar a tartaruga. Sua sintaxe é:

```
tartaruga.goto(x, y)
```

Onde x e y são as coordenadas x e y do Plano Cartesiano (tela). Lembrando que a tartaruga começa, por padrão, na posição (0, 0), que é o centro.

```
import turtle
tartaruga = turtle.Turtle()
tartaruga.width(2)
tartaruga.speed(1)
tartaruga.penup()
tartaruga.goto(100,100)
tartaruga.pendown()
tartaruga.circle(100)
turtle.done()
```

O programa leva a tartaruga até a posição (100, 100), sem criar linha na tela, e, dessa posição, começa a fazer um círculo de raio 100 à velocidade 1 (um pouco mais lenta que a usual).

Com a função goto é possível fazer a tartaruga desenhar gráfico de funções matemáticas. Vejamos abaixo um exemplo:

```
import turtle
from math import sin
euzinho = turtle.Turtle()
euzinho.color("blue")
euzinho.shape("circle")
euzinho.width(2)
turtle.bgcolor("black")
euzinho.speed(2)
for x in range (200):
    euzinho.goto(x, sin(x))
turtle.done()
```

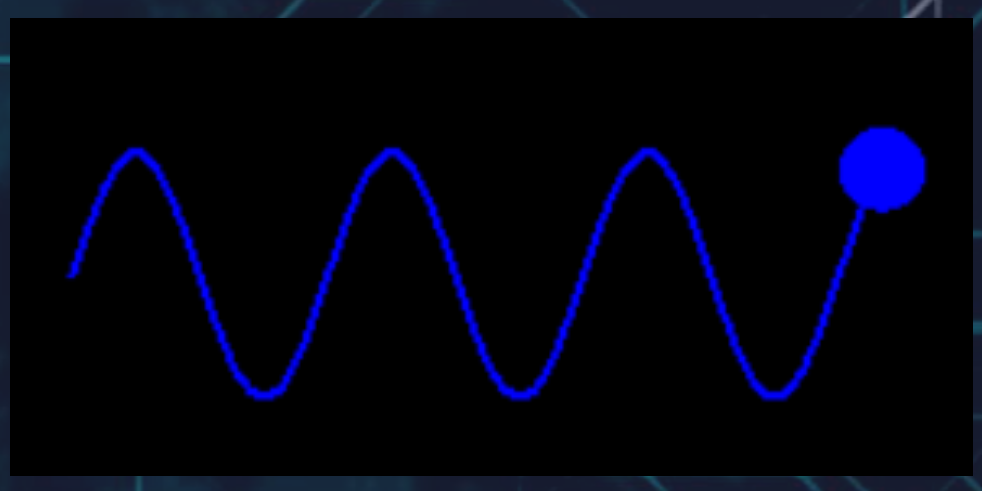
Que tratá como resultado:

Observe que as curvas do gráfico do seno ficaram muto pequenas: isso ocorre pois o Plano Cartesiano que estamos utilizando (tela) é muito grande (observe no código que o x variou de 0 até 199). Podemos ajustar a curva como quisermos utilizando multiplicações ou divisões.

Observe a seguir:

```
import turtle
from math import sin
euzinho = turtle.Turtle()
euzinho.color("blue")
euzinho.shape("circle")
euzinho.width(2)
turtle.bgcolor("black")
for x in range (200):
    euzinho.goto(x, 30*sin(x/10))
turtle.done()
```

Cujo resultado será:



Atividade 24: Crie os gráficos das funções:

a) $y = x^2$

b) $y = sen(x^2)$

O MÓDULO TURTLE

É possível criar muitas formas de arte complexa com o módulo turtle utilizando poucos comandos, desde que você utilize comandos de repetição e funções:

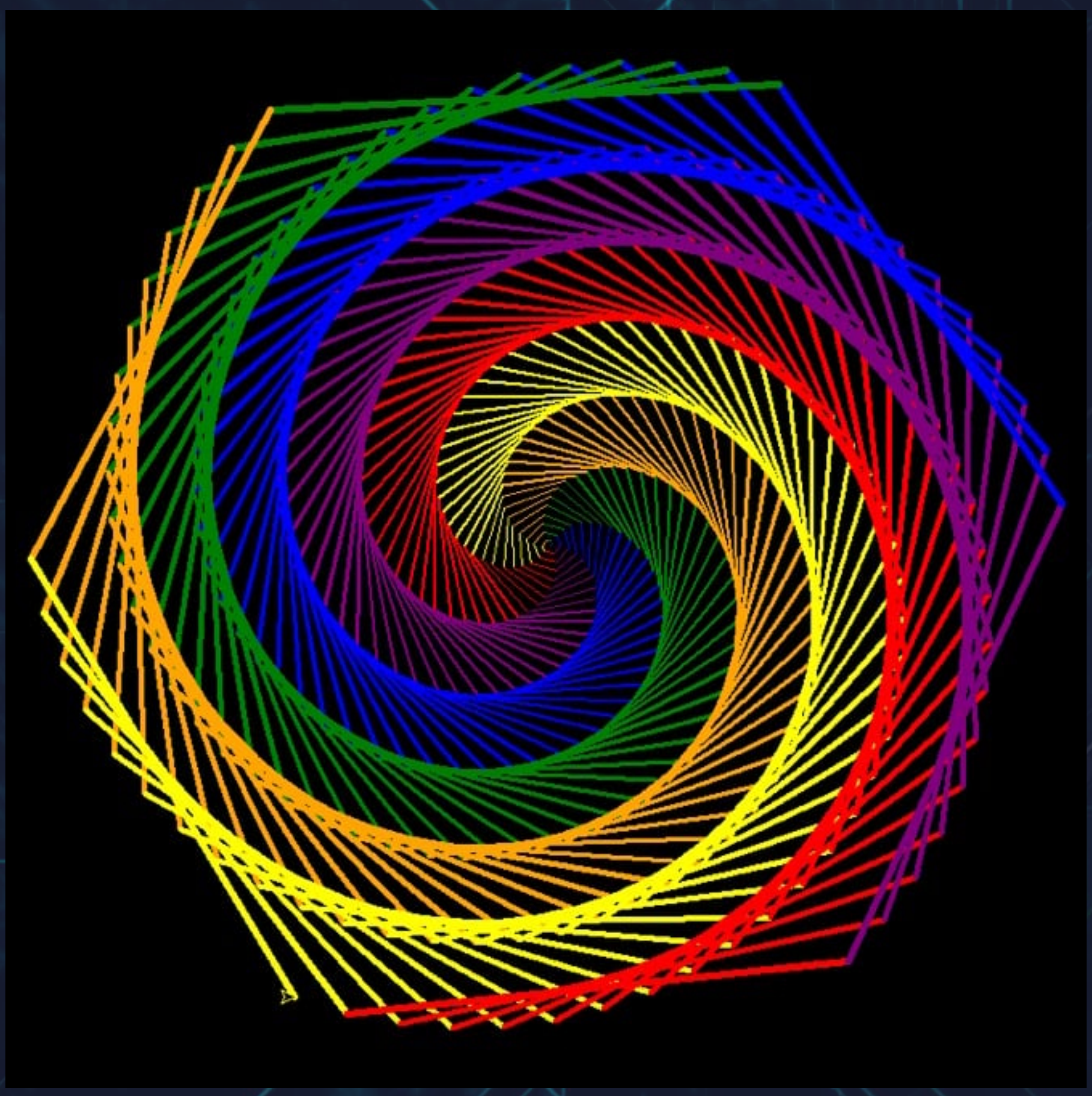
Tart_ladrilho.py
/storage/emulated/0...

```python
import turtle
eu = turtle.Turtle()
turtle.colormode(255)
turtle.Screen().bgcolor(10, 10, 20)
eu.color(255, 255, 0)
eu.speed(10)
eu.width(5)
cor = 250
for i in range(40):
    eu.forward(20 + (i*20))
    eu.left(90)
    eu.color(200, cor, 0)
    cor = cor - 200//40
turtle.done()
```

Note que, acima, com o comando colormode, passo a criar cores utilizando RGB, o que me dá muito mais controle sobre o desenho:

O MÓDULO TURTLE

Em diversos sites você pode encontrar muitos exemplos desse tipo de arte, como o mostrado abaixo:



Use sua criatividade!

CONSIDERAÇÕES FINAIS

Termino aqui o que desejava expor neste primeiro momento. Alguns pequenos ajustes e refinamentos serão feitos em uma segunda versão deste ebook, e isso tanto em partes já existentes (vou mostrar mais opções da função print, incluindo mais opções de concatenação e exibição de números; mostrarei também como criar tabelas para uma melhor exibição de certos resultados; como escrever comentários dentro do código e assim por diante) como também na inserção de novos capítulos, dentre eles: como resolver equações, matrizes, sistemas lineares; como construir gráficos estatísticos e gráfico de funções matemáticas (sem o módulo turtle).

Possíveis erros, dúvidas ou sugestões, por favor encaminhar para meu email profpedroromao@gmail.com.