UNIVERSIDADE FEDERAL DO PARÁ
INSTITUTO DE CIÊNCIAS EXATAS E NATURAIS
MESTRADO PROFISSIONAL EM MATEMÁTICA EM REDE NACIONAL

Marcelo da Silva Araújo

A experiência da criação de um aplicativo que gera gráficos de funções e o aprendizado da Matemática no ensino médio

Belém
2019

Marcelo da Silva Araújo

A experiência da criação de um aplicativo que gera gráficos de funções e o aprendizado da Matemática no ensino médio

Dissertação apresentada como requisito parcial para obtenção do título de Mestre em Matemática, pelo programa de Mestrado Profissional em Matemática em Rede Nacional – PROFMAT, do Instituto de Ciências Exatas e Naturais da Universidade Federal do Pará, sob orientação do Prof. Dr. Arthur da Costa Almeida.

Belém
2019

Dados Internacionais de Catalogação na Publicação (CIP) de acordo com ISBD
Sistema de Bibliotecas da Universidade Federal do Pará
Gerada automaticamente pelo módulo Ficat, mediante os dados fornecidos pelo autor

---

A658e Araújo, Marcelo da Silva.
A experiência da criação de um aplicativo que gera gráficos de funções e o aprendizado da Matemática no ensino médio / Marcelo da Silva Araújo, . — 2019.
52 f. : il.

Orientador: Prof. Dr. Arthur da Costa Almeida
Dissertação (Mestrado) – Programa de Pós-Graduação em Matemática em Rede Nacional, Instituto de Ciências Exatas e Naturais, Universidade Federal do Pará, Belém, 2019.

1. Matemática. 2. programação. 3. Python. 4. tecnologia. 5. ensino. I. Título.

CDD 510

Marcelo da Silva Araújo

A experiência da criação de um aplicativo que gera gráficos de funções e o aprendizado da Matemática no ensino médio

Dissertação apresentada como requisito parcial para obtenção do título de Mestre em Matemática, pelo programa de Mestrado Profissional em Matemática em Rede Nacional – PROFMAT, do Instituto de Ciências Exatas e Naturais da Universidade Federal do Pará, sob orientação do Prof. Dr. Arthur da Costa Almeida.

Data da Aprovação: 25/03/2011

Banca examinadora:

Prof. Dr. Arthur da Costa Almeida (Orientador)
Faculdade de Matemática – UFPA, Castanhal-PA

Prof. Dr. Edilberto Oliveira Rozal
Faculdade de Matemática – UFPA, Castanhal-PA

Prof. Dr. Samuel Levi Freitas da Luz
Faculdade de Matemática – UFPA, Castanhal-PA

Prof. Dr. João dos Santos Protázio
Faculdade de Estatística, UFPA, Belém-PA
Examinador externo

Dedico este trabalho à razão do meu viver, meu amor, Eliana Araújo Teran.

Agradecimentos

A Deus, por sua misericórdia, perdão e proteção.
A minha amada esposa, Eliana, por todo o seu apoio, compreensão e amor.
Aos meus pais, pela educação que me deram.
Ao meu orientador, professor Arthur, pela prontidão em mais uma vez me orientar em uma atividade tão importante.
Aos colegas do PROFMAT, pela amizade.
Aos professores do PROFMAT, por suas aulas.
A CAPES, pelo suporte financeiro.
A UFPA, pela minha formação profissional.

Sempre me pareceu estranho que todos aqueles que estudam seriamente esta ciência acabam tomados de uma espécie de paixão pela mesma. Em verdade, o que proporciona o máximo de prazer não é o conhecimento e sim a aprendizagem, não é a posse, mas a aquisição, não é a presença, mas o ato de atingir a meta.

Carl Friedrich Gauss

RESUMO

Este trabalho tem como objetivo descrever uma experiência de estudo de Matemática, auxiliado com noções de programação, em escolas da Rede Estadual. O estudo está em consonância com as habilidades descritas na Competência Específica 4, da área de Matemática e suas Tecnologias, da BNCC do Ensino Médio. A linguagem de programação escolhida é a Python, por ser uma das mais fáceis de aprender, de fácil aquisição e instalação e por dispor de vários módulos e bibliotecas de software voltados para soluções matemáticas. São abordadas brevemente as sequências recursivas, devido a grande importância no raciocínio computacional e na Matemática. Depois, mostramos como a linguagem de programação implementa estas sequências. Mais adiante, são revistos os conceitos de função, de plano cartesiano e da construção e análise de gráficos. Paralelo a isso, mostramos os conceitos gerais da programação, e as particularidades básicas da linguagem Python. São apresentados os módulos *math* e *turtle* que proveem, respectivamente de funções matemáticas e gráficas. Com isso, mostramos como o processo ensino/aprendizagem de matemática pode ser beneficiado com a adição de aulas de noções de programação utilizando a linguagem Python, e como os assuntos da matemática podem estar presentes nas atividades de programação.

Palavras-chave: programação, tecnologia, Python, Matemática

ABSTRACT

This work aims to describe an experience of Mathematics study, aided by programming notions, in schools of the State Network. The study is in line with the skills described in Specific Competence 4, Mathematics and its Technologies, from the BNCC of High School. The chosen programming language is Python, because it is one of the easiest to learn, easy to acquire and install and has several modules and software libraries for mathematical solutions. Recursive sequences are briefly discussed, due to their great importance in computational reasoning and mathematics. Then we show how the programming language implements these sequences. Further on, the concepts of function, Cartesian plane, and the construction and analysis of graphs are reviewed. Parallel to this, we show the general concepts of programming, and the basic peculiarities of the Python language. The *math* and *turtle* modules which provide respectively mathematical and graphical functions are presented. Thus, we show how the teaching / learning process of mathematics can benefit from the addition of classes of programming notions using the Python language, and how mathematical subjects can be present in programming activities.

Keywords: programming, technology, Python, Mathematics

LISTA DE FIGURAS

Figura 1: IDLE, o ambiente de desenvolvimento integrado do Python 16
Figura 2: Representação de listas no Python 17
Figura 3: Alguns exemplos simples de operações lógicas 17
Figura 4: Atribuição "=" e comparação "==" 18
Figura 5: Exemplo de uso de estruturas de controle if e for.....19
Figura 6: Exemplo de uso dos comando if e else.....20
Figura 7: Sequência de Fibonacci 21
Figura 8: Função sqrt (raiz quadrada).....22
Figura 9: Exemplo de solução computacional da recorrência 24
Figura 10: Exemplo de função definida recursivamente 26
Figura 11: Gráfico de funções 27
Figura 12: Gráfico de função 28
Figura 13: O objeto Screen 30
Figura 14: Trajeto formado pela sequência de comandos 31
Figura 15: Octógono desenhado no Turtle Graphics 33
Figura 16: Círculo desenhado no Turtle Graphics 34
Figura 17: Definindo o tamanho da tela 36
Figura 18: Quadriculado 38
Figura 19: Desenho a ser gerado na atividade 11 40
Figura 20: Eixos graduados.....42
Figura 21: Gráfico ocupando com exatidão a região definida 43
Figura 22: Mudança de coordenadas 45
Figura 23: Diferentes resoluções do gráfico 47
Figura 24: Gráfico de $f(x) = x^2 - x - 2$ 49
Figura 25: Gráfico de $(x + 1)^{(x + 1)} - 1$, com 49
Figura 26: Gráfico de $f(x) = \text{sen}(x)$, com 50

SUMÁRIO

1. CONSIDERAÇÕES INICIAIS.....12
2. A LINGUAGEM DE PROGRAMAÇÃO PYTHON.....16
Expressões e operadores booleanos.....17
Variáveis e atribuições.....17
Strings.....18
Listas.....18
2.1. ESTRUTURAS DE CONTROLE DE EXECUÇÃO.....19
2.2. MÓDULO *MATH*.....21
3. ESTUDO DE RECORRÊNCIAS AUXILIADO COM PYTHON.....24
3.1. SEQUÊNCIAS RECURSIVAS.....24
3.2. FUNÇÕES RECURSIVAS.....25
3.3. MÉTODO SIMPLES DE INTEGRAÇÃO GRÁFICA.....26
4. OBJETOS *TURTLE GRAPHICS*.....29
4.1. ATIVIDADES GEOMÉTRICAS NO TURTLE.....30
5. CONSTRUÇÃO DO APLICATIVO.....45
5.1. MUDANÇA DE SISTEMA DE COORDENADAS.....45
5.2. ESCOLHA DA RESOLUÇÃO DO GRÁFICO.....46
6. CONSIDERAÇÕES FINAIS.....51
REFERÊNCIAS.....52

1. CONSIDERAÇÕES INICIAIS

Existem muitos fatores que podem ser apontados como causa do desinteresse e, consequentemente, do baixo rendimento dos estudantes em matemática. Um destes fatores é a pouca habilidade de converter a linguagem natural para a linguagem matemática, e vice-versa. Feio (2009, p. 63), em sua dissertação de Mestrado, verificou que esta inabilidade resulta de quatro dificuldades básicas com as quais os estudantes se deparam:

> A primeira [dificuldade] apontou para o fato de existirem em cada registro de representação de um mesmo objeto matemático, diferentes conteúdos a serem mobilizados; a segunda mostrou que os alunos fracassam ao realizar a conversão da língua natural para a linguagem matemática quando não interpretam corretamente as regras matemáticas implícitas no enunciado de uma situação problema; a terceira surgiu do fato de existirem no texto de uma situação problema, palavras que os alunos não compreendiam o seu significado ou que geravam ambiguidade de sentidos; a quarta surgiu a partir do fato dos alunos não conseguirem compreender o significado matemático das letras utilizadas nos enunciados dos problemas (FEIO, p. 10).

A Base Nacional Comum Curricular (BNCC), para assegurar aos estudantes o direito à aprendizagem e ao desenvolvimento, categorizou as aprendizagens essenciais em dez **competências**[1] **gerais**, e as grandes áreas de conhecimento em **competências específicas**, que devem ser alcançadas com um conjunto de habilidades, por competência. A proposta deste trabalho atende, em maior grau, a competência específica 4 da Matemática e suas tecnologias no Ensino Médio, desenvolvendo a habilidade EM13MAT406: *Utilizar os conceitos de uma linguagem de programação na implementação de algoritmos escritos em linguagem corrente e/ou matemática* (BNCC do Ensino Médio, página 531).

> COMPETÊNCIA ESPECÍFICA 4: Compreender e utilizar, com flexibilidade e fluidez, diferentes registros de representação matemáticos (algébrico, geométrico, estatístico, computacional etc.), na busca de solução e comunicação de resultados de problemas, de modo a favorecer a construção e o desenvolvimento do raciocínio matemático (BNCC, pág. 530).

1 Na BNCC, competência é definida como a mobilização de conhecimentos, habilidades, atitudes e valores para resolver demandas complexas da vida cotidiana (BNCC do Ensino Médio, pág. 8).

A prática da programação aproxima a linguagem matemática da linguagem natural. Para que os estudantes se sintam mais motivados a aprender, muitas vezes, é necessário que lhes seja mostrado uma aplicação imediata daquilo que lhes é ensinado. Este trabalho descreve uma experiência com oficinas de programação e algoritmo para estudantes do 2º ano do ensino médio de duas escolas estaduais de Castanhal. As aulas iniciais mostraram que o estudo de programação desenvolve habilidades de resolução de problemas, incentiva os alunos a serem mais atentos ao seu trabalho e faz com que sejam mais perseverantes na solução de problemas difíceis. Com este experimento, oferecemos mais uma "camada" de soluções, pois faz com que os estudantes aprendam a traduzir os problemas (ou ideias matemáticas) para executá-los em códigos interpretados pelo computador. Isso exige que os alunos aprofundem seu entendimento matemático do problema, para que encontrem uma maneira de representar em uma linguagem de programação. A programação exige que os alunos sejam explícitos em suas estratégias de solução de problemas.

Desta forma, o estudo de programação enriquece os ambientes tradicionais de aprendizagem, pois o aluno terá a chance de construir o seu conhecimento. "Um outro nível de construção do conhecimento existe quando o aluno desenvolve um objeto de seu interesse, como um produto, serviço ou programa de computador" (ALMEIDA e FREITAS, 2015, apud, PAPERT, 1986, p. 55). Os conhecimentos matemáticos reforçados durante a prática de programar se tornam mais significativos, porque o aluno fica mais motivado. A BNCC do Ensino Médio faz várias recomendações visando "consolidar, aprofundar e ampliar a formação integral dos estudantes", sendo o ensino de Matemática auxiliado pelo uso de tecnologias, entre as quais a computacional.

> No Ensino Médio, na área de **Matemática e suas Tecnologias**, os estudantes devem utilizar conceitos, procedimentos e estratégias não apenas para resolver problemas, mas também para formulá-los, descrever dados, selecionar modelos matemáticos e desenvolver o pensamento computacional, por meio da utilização de diferentes recursos da área (BNCC, p. 470).

A prática da programação no contexto do ensino de Matemática, tal como é proposta neste trabalho, dá aos estudantes a oportunidade de desenvolver o

pensamento algébrico. No processo de criação de nossos scripts[2] de computação gráfica surgem várias demandas por identificar a relação de dependência entre duas grandezas e comunicá-las utilizando diferentes escritas algébricas. Em vários fragmentos de nosso código, funções e inequações serão implementadas. Computadores fazem exatamente o que lhes são instruídos a fazer, dependendo de suas capacidades de hardware e software, portanto este é um instrumento perfeito para se verificar o quanto um conceito de geometria, por exemplo, foi verdadeiramente assimilado. Mais uma vez, a proposta principal deste projeto está em sintonia com o que estabelece a BNCC:

> Em relação ao pensamento geométrico, eles [estudantes] desenvolvem habilidades para interpretar e representar a localização e o deslocamento de uma figura no plano cartesiano, identificar transformações isométricas e produzir ampliações e reduções de figuras. Além disso, são solicitados a formular e resolver problemas em contextos diversos, ampliando os conceitos de congruência e semelhança (BNCC, p. 517).

Python é a linguagem de programação escolhida para o desenvolvimento deste projeto, devido a sua simplicidade de codificação, facilidade de leitura e escrita, por ser de fácil acesso (livre[3] e multiplataforma[4]), e aprendizagem, apesar de não ser uma linguagem especializada para fins educacionais. O produto deste projeto será um aplicativo que gera gráficos de funções. Esta atividade estava em desenvolvimento na Escola Estadual Lameira Bittencourt, no 2º semestre de 2018, até o final do ano letivo. Seus primeiros resultados foram apresentados na I Mostra de Robótica e Objetos Digitais de Aprendizagem, promovida pelo NTE de Castanhal, no dia 13 de novembro de 2018. Esta experiência também foi feita na Escola Estadual Profª. Clotilde Pereira, e teve culminância no evento Mostra Científica e Cultural 2018, nos dias 06 e 07/12/2018. Para a maioria dos alunos, foi a primeira experiência em programação. Há muita matemática envolvida na construção deste aplicativo, tais como sequências numéricas, sistemas de coordenadas cartesianas, transformações lineares (escala e translação), além do estudo de funções. A experiência de programar soluções para problemas

2 Conjunto de instruções para que uma função seja executada em determinado aplicativo.

3 Software livre é aquele que concede ao usuário liberdade para executar, acessar, modificar e redistribuir. Geralmente, são gratuitos.

4 Neste contexto, nos referimos a programas multiplataforma como aqueles que são disponíveis para vários sistemas operacionais (Linux, Windows).

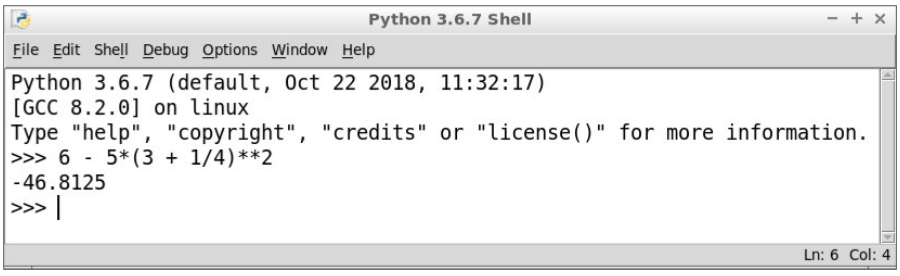
matemáticos tem contribuído bastante para o ensino e aprendizagem de Matemática no ensino básico. As atividades de programação propostas estimularam os estudantes a buscarem adquirir aprofundamento nos assuntos da Matemática.

2. A LINGUAGEM DE PROGRAMAÇÃO PYTHON

Python é uma linguagem de uso geral, projetada especificamente para tornar os programas bastante legíveis. Também possui uma rica biblioteca[5], tornando possível criar aplicações sofisticadas usando código de aparência relativamente simples. Python é um software livre, gratuito e pode ser baixado no site https://www.python.org/ para vários SO[6], entre os quais, Windows, Linux/Unix, Mac OS X e outros. Geralmente, já vem, por padrão, instalado nas distribuições Linux. Este projeto foi desenvolvido nos ambientes Linux Fedora, Mint e Lubuntu. Não descreveremos aqui o processo de instalação, porque isso varia de SO, e porque essa informação é fácil de ser obtida na internet.

O ambiente de desenvolvimento foi a própria IDE[7] padrão do Python, a IDLE. Para começar, abrimos uma janela do shell interativo do Python (Figura 1). Expressões matemáticas foram as primeiras experimentações que os alunos fizeram no ambiente. As operações de adição, subtração, multiplicação e divisão são efetuadas como os caracteres +, –, * e /, respectivamente. Para efetuarmos, por exemplo, a potenciação 2^{10}, escrevemos 2**10; por exemplo, para calcular o quociente de 20 por 8, escrevemos 20//8; para calcular o resto da divisão euclidiana de 20 por 8, escrevemos 20%8. O Python entende a ordem de precedência dos operadores tal como ensinamos nas aulas de Matemática.

Figura 1: IDLE, o ambiente de desenvolvimento integrado do Python, mostrando como calculamos, por exemplo, $6 - 5\left(3 + \frac{1}{4}\right)^2$.



Fonte: Autor

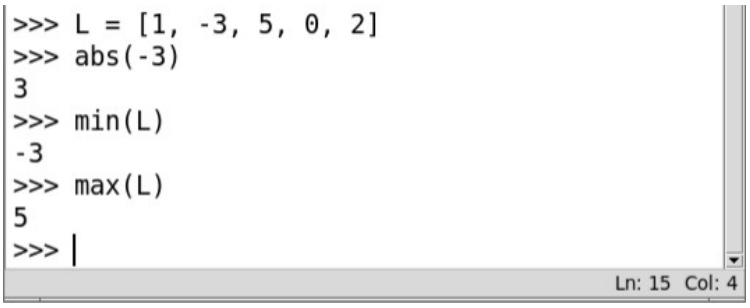
5 Assim como outras linguagens, o Python consiste em um pequeno conjunto de instruções de propósito de uso geral. Para instruções mais específicas, são usadas bibliotecas de software, separadas do núcleo da linguagem (PERKOVIC).

6 Sistema Operacional.

7 Ambiente de desenvolvimento integrado.

Além destas operações, Python também tem suporte a outras funções matemáticas interessantes tais como `abs`, que retorna o valor absoluto de um número dado, e `min` e `max` que retornam o menor e o maior valor entre uma lista de números dados (Figura 2).

Figura 2: No Python, representamos listas com seus elementos entre colchetes, separados por vírgula.

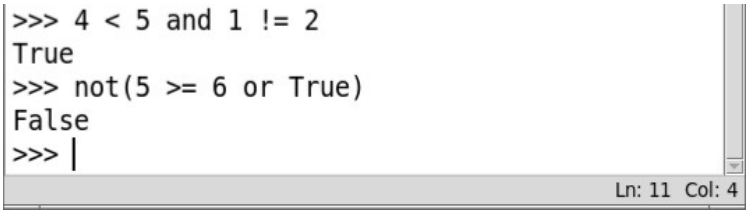


Fonte: Autor

Expressões e operadores booleanos

Também podemos usar os operadores de comparação $>$, $<$, $\geq$, $\leq$, $=$ e $\neq$, os conectivos lógicos $\wedge$ e $\vee$, indicados, respectivamente, por `and` (e) e `or` (ou), a negação `not` e as constantes `True` (Verdadeiro) e `False` (Falso) (Figura 3).

Figura 3: Alguns exemplos simples de operações lógicas.



Fonte: Autor.

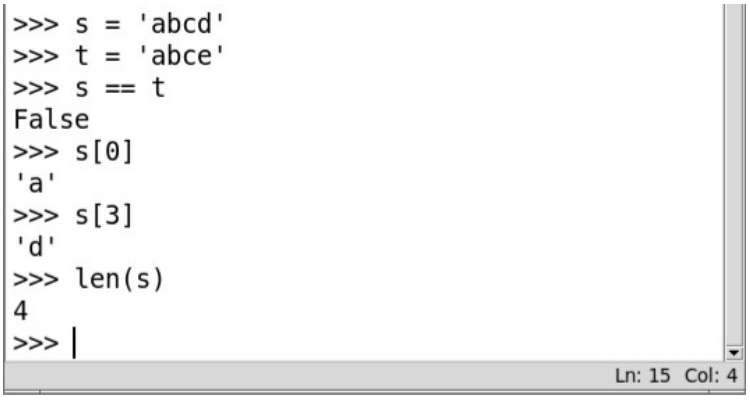
Variáveis e atribuições

Em programação, variável tem a mesma ideia que tem em álgebra. Por exemplo, o número 1 pode ser atribuído à variável x em um problema de álgebra, fazendo $x = 1$. A instrução de atribuição em Python é feita da mesma forma. Não entraremos mais em detalhes sobre isso, por ser algo muito básico em Matemática.

Strings

O tipo *string*, denominado `str`, é usado para representar uma cadeia de caracteres, palavras e textos. Assim como os dados numéricos, as strings podem ser comparadas usando os operadores == (igual), != (diferente), > (maior) ou <.
O operador + concatena duas strings. Os caracteres de uma string podem ser acessados usando o operador de indexação [ ]. O índice de um caractere de uma string é um inteiro que indica a posição (ordem) do caractere na string. O primeiro caractere tem índice 0, o segundo tem índice 1, o terceiro tem índice dois e assim por diante até o último, que tem índice $n - 1$, em uma string com n caracteres. A função `len()` retorna o número de caracteres da string (Figura 4).

Figura 4: Observe que usamos "=" para atribuir valor a uma variável, e "==" para comparar dois valores. A expressão `x[i]` retorna o valor do caractere da string de posição $i + 1$.



Fonte: Autor

Listas

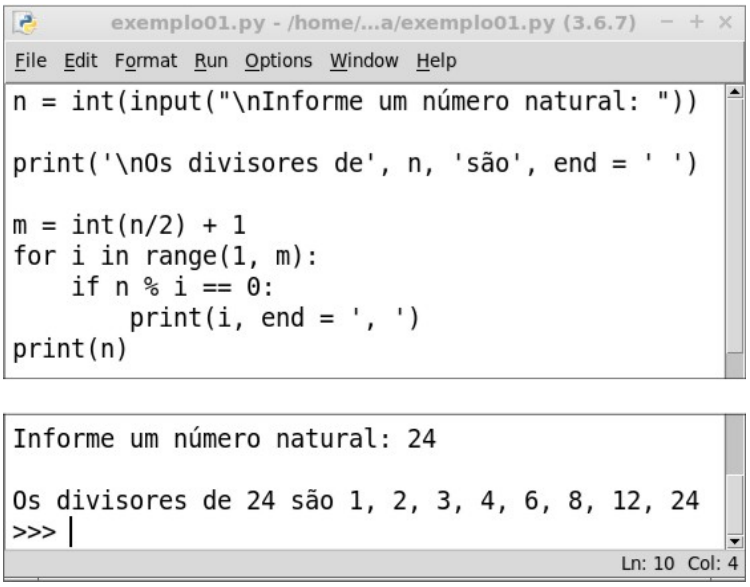
Em muitas situações, precisamos organizar os dados em listas (sequências). Em Python, as listas são armazenadas em um tipo de objeto denominado `list`. Os elementos da lista podem ser de qualquer tipo: números, strings (palavras), ou até mesmo outras listas. Uma lista é representada como uma sequência de objetos separados por vírgula, dentro de colchetes. Uma lista vazia é representada por [ ].
A maioria dos operadores de string também são válidos para listas. Os itens de uma lista podem ser acessados usando o operador de indexação. O comprimento de uma lista (número de elementos) é obtido usando a função len().

Como exemplo, seja dada a lista L = [10, -5, "xyz", [20, 30]]. $L[0]$ = 10, $L[2][1]$ = "y", $L[3]$ = [20, 30], `len`(L) = 4 e `len`($L[3]$) = 2.

2.1. ESTRUTURAS DE CONTROLE DE EXECUÇÃO

Um programa é uma sequência de instruções executadas em sucessão. Os programas de computador normalmente realizam coisas diferentes, dependendo dos valores informados. Por exemplo, vamos escrever um script que receba um número natural n, e retorne todos os divisores naturais deste número (Figura 5). Sabemos que 1 e o próprio n são os divisores triviais de n. Então, seria suficiente avaliar apenas todos os números entre 2 e $n/2$. Esta forma ainda não é a mais eficiente, em termos computacionais, porém é a que tem um algoritmo mais simples.

Figura 5: Exemplo de uso de estruturas de controle `if` e `for`.



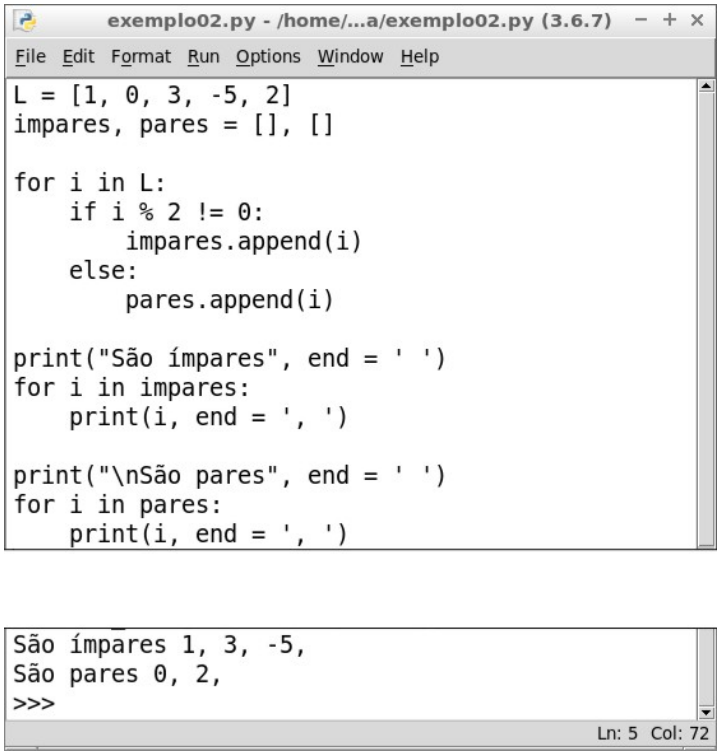
Fonte: Autor

O comando `input` imprime uma instrução ao usuário e espera que ele escreva algo, que, em seguida, será armazenado na variável n, na forma de string. Como queremos que n seja um número inteiro, precisamos convertê-lo em inteiro com o comando `int()`. A variável m armazena o quociente da divisão de n por 2 mais 1. O comando `range`(1, m) gera a sequência dos números naturais de 1 a m – 1. O comando `for i in range`(1, m) atribui à variável i os valores da sequência gerada pelo range, uma a cada iteração. Em cada iteração, o comando `if n % i == 0`

avalia se o resto da divisão de n por i (n % i) é igual a zero, ou seja, se i é divisor de n. Se sim, o valor de i é impresso. Após a última iteração, n é impresso.

O escopo da função `if` informa o que fazer se o valor lógico da sentença avaliada for verdadeiro. No nosso exemplo, não instruímos o programa sobre o que fazer se a sentença for falsa. Quando isso for necessário, usamos o comando `else` (Figura 6).

Figura 6: Exemplo de uso dos comando if e else.

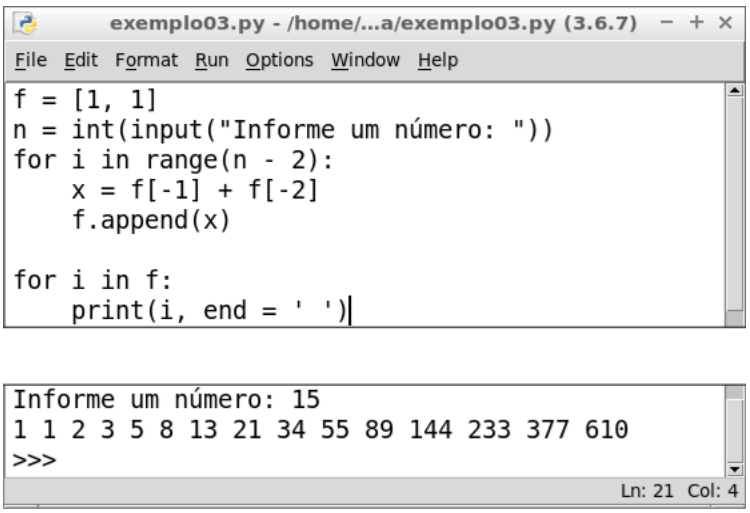


Fonte: Autor

Neste script, criamos uma lista de nome L, com cinco elementos, e duas listas vazias, de nomes *impares* e *pares*. O comando `for i in L` percorre os elementos da lista, um por um. Se o elemento for ímpar (`if i % 2 != 0`, ou seja, se o resto de sua divisão por 2 for diferente de 0) é adicionado à lista *impares*, com o comando `append`. Caso contrário (else), é adicionado a lista *pares*.

Para encerrar essa seção reforçando um pouco mais os conceitos de lista e estruturas de controle de execução, vamos criar um script que gere os números de Fibonacci, até uma posição informada pelo usuário (Figura 7). Desse modo, se o número informado fosse 8, isso retornaria 1, 1, 2, 3, 5, 8, 13, 21.

Figura 7: Sequência de Fibonacci. Mais um exemplo de uso da função `append`, e do operador de indexação.



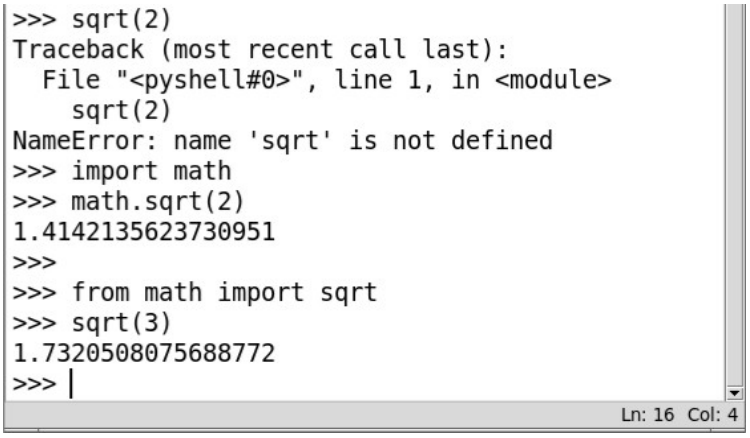
Fonte: Autor

Já sabemos que para acessar o elemento de ordem k de uma lista L, escrevemos $L[k-1]$, porque os elementos de uma lista em Python são ordenados a partir de 0. Mas e se quisermos acessar o último elemento de L sem especificar o número de elementos da lista? Para isso, escrevemos $L[-1]$. O penúltimo elemento da lista será $L[-2]$, e assim sucessivamente. Todo elemento n_k da sequência de Fibonacci é obtido somando $n_{k-1} + n_{k-2}$. No nosso script, em cada iteração, fazemos a variável x receber a soma do último com o penúltimo elemento de L. Em seguida, colocamos x no final de L.

2.2. MÓDULO *MATH*

Para ter acesso a outras funções e constantes matemáticas, temos que importar o módulo `math`, que faz parte da biblioteca padrão do Python. Importamos um módulo escrevendo o comando `import math` (Figura 8). O comando import coloca à disposição todas as funções matemáticas definidas no módulo. Se quisermos importar, não todo o módulo math, mas somente as funções que usaremos no programa, por exemplo seno e cosseno, escrevemos `from math import sin, cos`.

Figura 8: Observe que a função `sqrt` (raiz quadrada) só fica disponível depois que importamos o módulo math. Para usá-la, escrevemos com a sintaxe *módulo.função*().



Fonte: Autor

Na tabela abaixo, temos algumas funções do módulo math.

Função	Descrição	Exemplos

ceil(x)	Retorna o maior inteiro maior ou igual a x.	`>>> ceil(-3.01)`
		`-3`
		`>>> ceil(3.01)`
		`4`

floor(x)	Retorna o inteiro menor ou igual a x.	`>>> floor(-3.01)`
		`-4`
		`>>> floor(3.01)`
		`3`

fabs(x)	Retorna o valor absoluto de x.	`>>> fabs(-4)`
		`4.0`
		`>>> fabs(4) == fabs(-4)`
		`True`

factorial(x)	Retorna x fatorial.	`>>> factorial(5)` `120`
gcd(a, b)	Retorna o MDC dos inteiros a e b.	`>>> gcd(12, 18)` `6`
modf(x)	Retorna as partes fracionária e inteiro de x.	`>>> modf(7/3)` `(0.333333333333333, 2.0)`

log(x [, b])	Com um argumento, retorna o logaritmo natural de x. Com dois, retorna o logaritmo de x na base b.	`>>> log(e)`
		`1.0`
		`>>> log(81, 3)`
		`4.0`

Função	Descrição	Exemplos
exp(x)	Retorna e^x.	`>>> exp(1)` `2.718281828459045`
log10(x)	Retorna o logaritmo de x na base 10.	`>>> log10(.1)` `-1.0`
pow(x, y)	Retorna x^y.	`>>> pow(2, 10)` `1024.0`
sqrt(x)	Retorna a raiz quadrada de x.	`>>> sqrt(.36)` `0.6`
cos(x)	Retorna o cosseno de x em radianos.	`>>> cos(pi)` `-1.0`
acos(x)	Retorna o arco cosseno de x, em radianos.	`>>> acos(1)` `0.0`
degrees(x)	Converte o ângulo x de radianos em graus.	`>>> degrees(pi/2)` `90.0`
radians(x)	Converte o ângulo x de graus em radianos.	`>>> radians(180)` `3.141592653589793`
pi	Retorna a constante π = 3,14159...	`>>> pi` `3.141592653589793`

Fonte: Adaptado de https://docs.python.org/3/library/math.html

3. ESTUDO DE RECORRÊNCIAS AUXILIADO COM PYTHON

A recursão é objeto de estudo da Matemática, e uma técnica de solução de problemas muito usada na programação. As funções recursivas são funções de domínio $\mathbb{N}$, e solucionam um problema recursivamente chamando a si mesmas. Podemos entender as funções recursivas como aquelas que têm como imagem de n o enésimo termo de uma sequência recursiva. Então, vamos começar com o estudo das sequências recursivas. Neste trabalho, não discutiremos as recorrências com relação a ordem (recorrências de 1ª ordem, de 2ª ordem...).

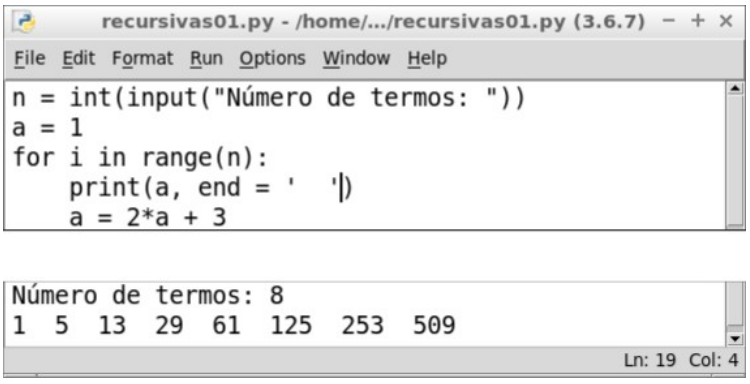
3.1. SEQUÊNCIAS RECURSIVAS

Consideremos uma sequência definida por $a_n = 2a_{n-1} + 3$, com $a_0 = 1$ (Neste trabalho, adotaremos como 1º termo da sequência a_0 e não a_1, para melhor aproveitar a função `range()` do Python). Seus primeiros termos são:

$$1, 5, 13, 29, 61, 125, \ldots$$

Uma solução computacional de encontrar os n primeiros termos desta sequência está expressa abaixo (Figura 9).

Figura 9: Exemplo de solução computacional da recorrência $a_n = 2a_{n-1} + 3$, com $a_0 = 1$.



Fonte: Autor

Uma análise na sequência sugere que seus termos podem ser expressos da forma:

$$4 - 3,\ 8 - 3,\ 16 - 3,\ 32 - 3,\ \ldots,\ 4 \cdot 2^n - 3$$

Lembrando, que definimos o 1º termo como a_0, então $a_n = 4 \cdot 2^n - 3$, ou $a_n = (a_0 + 3) \cdot 2^n - 3$.

3.2. FUNÇÕES RECURSIVAS

Antes de mostrarmos a implementação de uma função recursiva, vamos entender como as funções são definidas no Python. Por exemplo, se quisermos fazer a função $f(x) = 2x - 1$, usamos o comando `def` pra definir uma função em Python. Em seguida, escrevemos a expressão da função, precedida da palavra `return`, como abaixo:

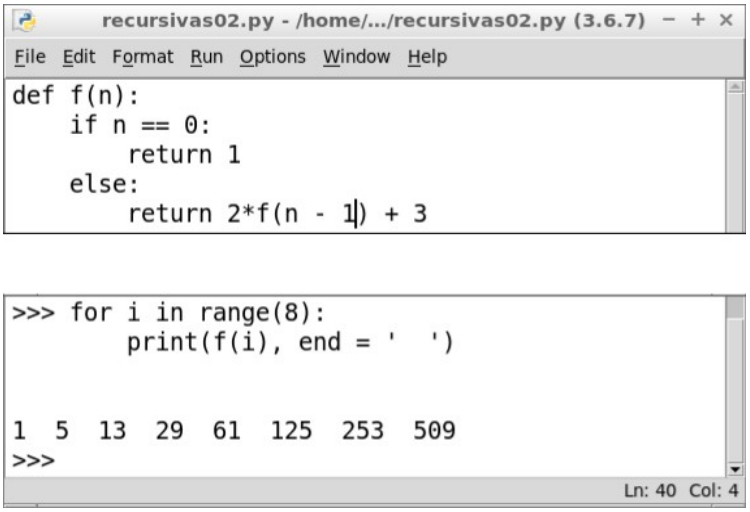
```
>>> def f(x):
        return 2*x - 1
>>> f(10)
19
>>>
```

Voltando a sequência recursiva anterior, podemos testar o termo geral, $a_n = 4 \cdot 2^n - 3$:

```
>>> def g(n):
        return 4*2**n - 3
>>> g(7) # Que na verdade é o 8o termo.
509
>>>
```

Desse modo, podemos definir uma função recursiva pra encontrar os n primeiros termos da sequência $a_n = 2a_{n-1} + 3$, com $a_0 = 1$ do seguinte modo (Figura 10):

Figura 10: Exemplo de função definida recursivamente. No shell, usamos a função for para calcular os oito primeiros termos da sequência.



Fonte: Autor

3.3. MÉTODO SIMPLES DE INTEGRAÇÃO GRÁFICA

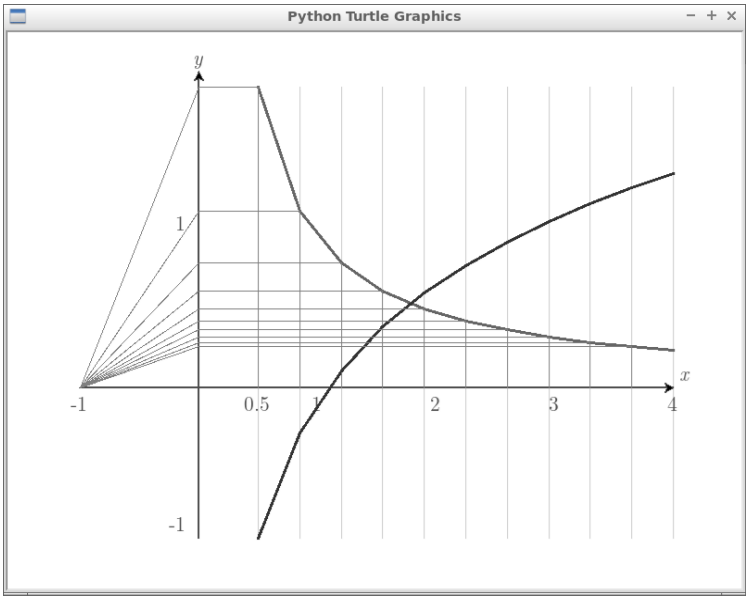
Terminamos este capítulo mostrando um último exemplo de solução recursiva implementada com programação Python. Trata-se de um procedimento gráfico descrito por Richard Courant, no vol. I do livro Cálculo Diferencial e Integral, que detalha a construção da curva da integral de uma função. Primeiro, vamos a definição de integral:

> Uma integral definida ou função primitiva de $f(x)$ é uma função $y = F(x)$ que pode ser considerada, não somente como área, mas, como qualquer outra função, pode, também, ser representada graficamente por uma curva. A definição sugere a possibilidade imediata de se construir tal curva aproximadamente, obtendo-se, assim, o gráfico da função integral (COURANT, página 119).

Inicialmente, vamos descrever o método, fazendo algumas adaptações. Seja dada uma função $f(x)$. Queremos representar em uma mesma tela, o seu gráfico e o gráfico de sua integral $F(x)$ em um dado domínio $[a, b]$. Vamos estabelecer que a curva integral passa pelo ponto (a, k), arbitrariamente escolhido. Para cada valor de x, a curva fica determinada pela direção igual ao valor de $f(x)$. No nosso experimento, dividiremos o intervalo $[a, b]$ por meio dos pontos $x = a, x_1, x_2, ...,$

$x_n = b$ em n partes iguais (geralmente, diz partes não necessariamente iguais, mas faremos assim para facilitar a programação). Por estes pontos de divisão elevaremos paralelas ao eixo dos y. Tracemos pelo ponto (a, k), a linha reta, cuja inclinação é igual a $f(a)$; pela interseção desta com a linha $x = x_1$ traçaremos outra linha com a inclinação $f(x_1)$; pela interseção desta com a linha $x = x_2$, uma linha com inclinação $f(x_2)$, e assim sucessivamente. Na prática, cada um dos n segmentos do gráfico poligonal que representará $F(x)$ terá extremidades $(x_i, f(x_i))$ e $(x_{i+1}, f(x_{i+1}))$ e sua inclinação será numericamente igual a $f(x_i)$. A construção deste gráfico fica mais didática, se projetarmos no eixo dos y as ordenadas $f(x_i)$, e traçarmos, segmentos com extremidades $(-1, 0)$ e $(0, f(x_i))$ (Figuras 11 e 12). A inclinação de cada um destes segmentos será igual à inclinação dos segmentos do contorno poligonal que resultará no gráfico. Quanto maior o valor de n, maior será a aproximação do contorno poligonal com a curva de $F(x)$.

Figura 11: Gráfico das funções $f(x) = \frac{1}{x}$, e de sua integral, no domínio [0,5; 4].



Fonte: Autor

Podemos compreender as ordenadas $F(x_i)$, com $0 \leq i \leq n$, como elementos de uma sequência recursiva, começando com $F(x_0) = F(a) = k$. Para expressar $F(x_{i+1})$ em função de $F(x_i)$, resolvemos a proporção:

$$\frac{F(x_{i+1}) - F(x_i)}{x_{i+1} - x_i} = f(x),$$

mas repartimos $[a,\ b]$ de modo que $x_{i+1} - x_i = \dfrac{b-a}{n}$, então

$F(x_{i+1}) - F(x_i) = \dfrac{b-a}{n} f(x)$. Fazendo $\dfrac{b-a}{n} = h$, chegamos a expressão

$F(x_{i+1}) = F(x_i) + hf(x)$.

Para escrever um script que desenha estes gráficos, de $f(x)$ e $F(x)$, criamos três listas (sequências) de $n + 1$ elementos,

$X = \{a,\ x_1,\ x_2,\ ...,\ b\}$,

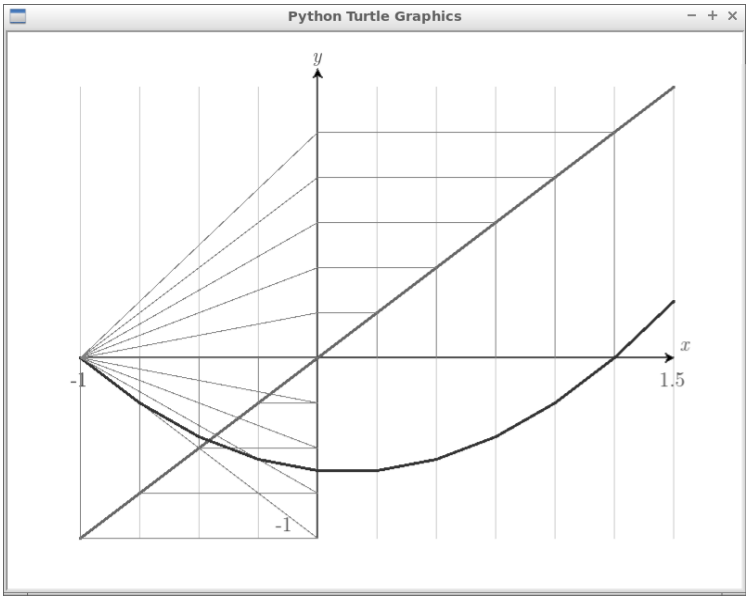
$Y = \{f(a),\ f(x_1),\ f(x_2),\ ...,\ f(b)\}$ e

$F = \{k,\ F(x_1),\ F(x_2),\ ...,\ F(x_n)\}$.

Atribuindo a notação $X_i = x_i$, $Y_i = f(x_i)$ e $F_i = F(x_i)$, os segmentos que formam os contornos poligonais de $f(x)$ terão extremidades $(X_i,\ Y_i)$ e $(X_{i+1},\ Y_{i+1})$, e os que formam o de $F(x)$ terão extremidades $(X_i,\ F_i)$ e $(X_{i+1},\ F_{i+1})$.

Omitiremos, aqui, os códigos Python que geram os gráficos, porque estas construções só serão abordadas nos capítulos seguintes.

Figura 12: Gráfico da função $f(x) = x$, e de sua integral no domínio [-1; 1,5].



Fonte: Autor

4. OBJETOS *TURTLE GRAPHICS*

Disponível através do módulo `turtle`, a ferramenta `turtle graphics` dispõe de recursos para desenhar linhas e outras formas na tela do computador. Não é o objetivo deste trabalho examinar todos os recursos do módulo `turtle`, mas mostraremos os suficientes para implementar o programa gráfico que propomos.

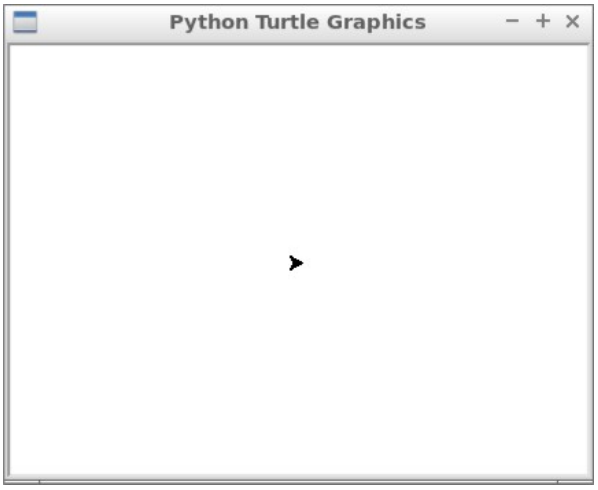
> Turtle graphics tem uma longa história, desde a época em que o ramo da ciência da computação estava sendo desenvolvido. Ele faz parte da linguagem de programação Logo, desenvolvida por Daniel Bobrow, Wally Feurzig e Seymour Papert em 1966. A linguagem de programação Logo e seu recurso mais popular, turtle graphics, foi desenvolvida para fins de ensino de programação. A tartaruga (turtle) era originalmente um robô, ou seja, um dispositivo mecânico controlado por um operador de computador. Uma caneta era presa ao robô e deixava um rastro na superfície enquanto o robô se movia de acordo com as funções inseridas pelo operador (PERKOVIC, pág. 42).

Começaremos importando o módulo turtle e instanciando um objeto Screen, que chamaremos de *s*. Em seguida, iniciaremos a tartaruga (ou turtle), que funciona como uma caneta, instanciando um objeto Turtle que chamaremos de *t*.

```
>>> import turtle
>>> s = turtle.Screen()
>>> t = turtle.Turtle()
```

Surge no centro do monitor uma janela com fundo branco, com largura e altura iguais a 50% e 75%, respectivamente, da largura e altura do monitor (Figura 13). No centro desta janela, ponto de coordenadas (0, 0), acionado pelo comando `t = turtle.Turtle()`, temos o cursor, ou tartaruga t.

Figura 13: O objeto `Screen` é a tela na qual iremos desenhar. No centro, ponto de coordenadas (0, 0), temos o objeto `Turtle`.



Fonte: Autor

Diferente da maneira como apresentamos as funções do módulo math, na próxima sessão, vamos mostrar as funções do módulo turtle com uma sequência de atividades práticas com complexidade crescente.

4.1. ATIVIDADES GEOMÉTRICAS NO TURTLE

As atividades a seguir (algumas das mais relevantes, propostas nas oficinas) devem ser executadas no shell do Python.

Atividade 01

(1) Importe o módulo turtle; (2) instancie com nome s o objeto Screen do módulo turtle; (3) instancie com nome t o objeto Turtle do módulo turtle; (4) com o comando `t.forward(100)`, mova t em 100 px; (5) com o comando `t.left(30)`, vire t 30° para a esquerda; (6) mova t em 50 px; (7) vire t 45° para a direita com o comando `t.right(45)`; (8) mova t em 80 px para trás com o comando `t.backward(80)`. Esta sequência de comandos gerará a figura 14.

Uma solução:

```
import turtle          # 1
s = turtle.Screen()    # 2
t = turtle.Turtle()    # 3
t.forward(100)         # 4
t.left(30)             # 5
t.forward(50)          # 6
t.right(45)            # 7
t.backward(80)         # 8
```

Figura 14: Trajeto formado pela sequência de comandos.

Observe que a unidade angular dos comandos `left()` e `right()` é o grau. A `#` é usada para que o programador possa adicionar um comentário ao código. Tudo que estiver escrito à direita de `#`, será ignorado pelo interpretador Python.

Atividade 02

Já sabemos que o comando

```
for _ in range(n):
    <instruções Python>
```

repete `<instruções Python>` n vezes. Escreva um script que desenhe um quadrado de lado 100 px.

Uma solução:

```
for _ in range(4):
    t.forward(100)
    t.left(90)
```

Também já sabemos que o comando `range(4)` cria a sequência [0, 1, 2, 3]. De modo geral, range(n) = [0, 1, 2, ..., n – 1]. O conjunto de instruções que deve ser repetido pelo comando `for` é identificado pela identação (deslocamento de um TAB).

Atividade 03

Faça ajustes no script da atividade 02, para construir um pentágono regular de lado 100 px.

Uma solução:

```
for _ in range(5):
    t.forward(100)
    t.left(360/5)
```

Se um polígono regular tem n lados, seus ângulos internos terão $180^o - \frac{360^o}{n}$, logo seus ângulos externos terão $\frac{360^o}{n}$.

Esta atividade e as seguintes serão escritas no bloco de notas da IDLE do Python. Deste modo, poderemos gravar os nossos scripts.

Atividade 04

Nesta atividade, você vai construir um script pra desenhar um polígono com o número de lados que o usuário quiser. O usuário também deve informar o tamanho do lado.

Uma solução:

Clique Ctrl+N, para abrir o editor de texto da IDLE, e digite as instruções abaixo:

```
n = int(input("Informe o número de lados "))
lado = float(input("Informe o tamanho do lado "))
import turtle
s = turtle.Screen()
t = turtle.Turtle()
for _ in range(n):
    t.forward(lado)
    t.left(360/n)
```

Em seguida, salve o arquivo com o nome ativ04.py, e aperte F5 para executar o código.

Atividade 05

Vamos modificar o script da atividade anterior, preenchendo o polígono (Figura 15) e marcando os vértices do polígono com um ponto.

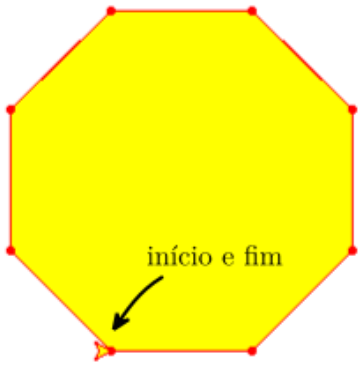
Uma solução:

Acrescente as novas linhas ao código do arquivo ativ04, e salve como ativ05.py.

```
n = int(input("Informe o número de lados "))
lado = float(input("Informe o tamanho do lado "))
```

Figura 15: Foi atribuído 8 à n.

```
import turtle
s = turtle.Screen()
t = turtle.Turtle()
t.color('red', 'yellow')
t.begin_full()
for _ in range(n):
    t.dot()
    t.forward(lado)
    t.left(360/n)
t.end_full()
```

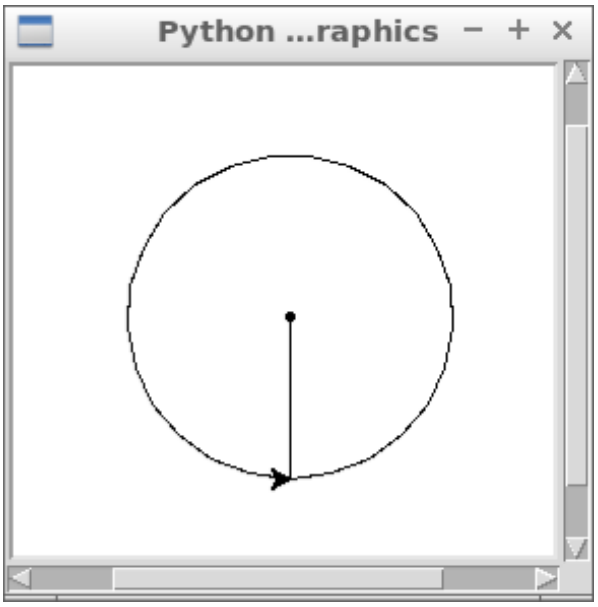


O comando `n = input("Informe o número de lados ")` imprime a mensagem "Informe o número de lados ", e fica esperando que o usuário escreva alguma coisa, que será guardado na variável n na forma de string. Como precisamos que n seja um número inteiro, usamos o comando `int`(n), que converte a string n em inteiro. Para converter *lado* de string em número real, usamos o comando `float`(*lado*). O comando `t.color`('*red*', '*yellow*') atribui a cor vermelha aos pontos e lados, e amarela ao polígono. O comando `t.dot`() cria uma ponto na posição em que estiver o objeto t. Para que o turtle compreenda que uma sequência de segmentos forma um polígono, os comandos que definem esta sequência devem ser escopo de `begin_full`, `end_full`.

Atividade 06

O comando `t.circle(100)` desenha um círculo de raio medindo 100 px. Por padrão, a "cabeça" da tartaruga t aponta inicialmente pra esquerda. Quando escrevemos `t.left(30)`, por exemplo, t muda de direção em 30^o para a esquerda. Para desenhar um círculo no centro da tela, estando t no centro (0, 0) e apontando para a esquerda, podemos mudar a posição inicial de t para o ponto (-r, 0), onde r é a medida do raio do círculo. Assim, t vai desenhando o círculo no sentido anti-horário em torno do ponto (0, 0). Desenhe um círculo com centro no centro do plano (Figura 16). O tamanho do raio deve ser solicitado para o usuário.

Figura 16: Ao iniciar o círculo, o cursor estava na posição $(0, -r)$, onde r é o raio.



Fonte: Autor

Uma solução:

```
r = float(input("Informe a medida do raio: "))
import turtle
s = turtle.Screen()
t = turtle.Turtle()
t.dot()
t.penup()
t.goto(0, -r)
t.pendown()
t.circle(r)
```

O comando `t.penup()` "levanta a caneta" t, para que ela possa ser movida sem riscar. O comando `t.pendown()` abaixa a caneta t. O comando `t.goto(x, y)` muda a posição de t de onde estiver para o ponto de coordenadas (x, y). Como t estava em (0, 0), e só precisávamos alterar o y, poderíamos ter usado ao invés de `t.goto(0, -r)`, `t.sety(-r)`.

Atividade 07

Nesta atividade, vamos escrever um script que gera um polígono regular no centro da tela. A posição de seus vértices será dada por coordenadas polares.

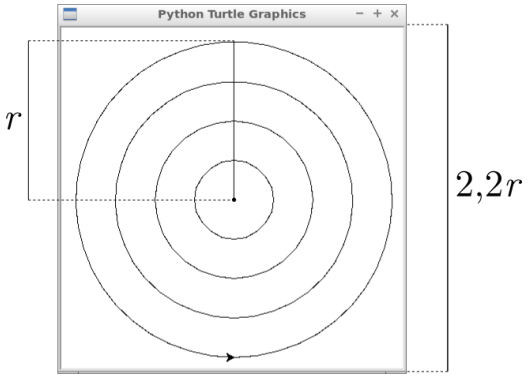
```
n = int(input("Informe o número de lados: "))
r = float(input("Informe a medida do raio: "))

from math import sin, cos, pi
import turtle
s = turtle.Screen()
t = turtle.Turtle()

t.dot()
t.penup()
t.setx(r)
t.pendown()
for i in range(n):
    ang = 2*pi*(i + 1)/n
    x = r*cos(ang)
    y = r*sin(ang)
    t.goto(x, y)
```

Para usar as funções seno e cosseno e a constante pi, precisamos importar sin e cos do módulo math. Observe que o ângulo ang precisa ser dado em radianos $\left(ang = \frac{2\pi}{n}i\right)$, para que seja interpretados nas funções `sin()` e `cos()`. Usamos o comando `for` de um jeito diferente dessa vez. `for i in range`(n) faz i assumir os valores de [0, 1, 2, ..., n – 1], um por um, fazendo com que a variável *arg* assuma os valores 0, $\frac{2\pi}{n}$, $\frac{4\pi}{n}$, ..., $\frac{2(n-1)\pi}{n}$.

Atividade 08

Nesta atividade, vamos fazer algumas modificações na tela de imagem. Desenhe em uma tela 4 círculos concêntricos, com um ponto no centro (Figura 17). O desenho deve ser impresso em uma tela quadrada de lado medindo $2{,}2r$, onde r é o raio do círculo maior, que deve ser informado pelo usuário.

Figura 17: Agora, temos controle sobre o tamanho da tela.



Fonte: Autor

Uma solução:

```
r = float(input("Informe o tamanho do raio: "))
import turtle
s = turtle.Screen()
s.setup(width = 2.2*r, height = 2.2*r, startx = 0, starty = 0)
t = turtle.Turtle()
t.speed(0)
t.dot()
for k in range(1, 5):
    t.penup()
    t.sety(-.25*k*r)
    t.pendown()
    t.circle(.25*k*r)
```

Aqui, aprendemos um novo comando. A função s.setup() recebe quatro parâmetros que interferem na tela de desenho. Os parâmetros `width` e `height` informam o comprimento e a altura da tela, respectivamente. Os parâmetros `startx`, e `starty` indicam as coordenadas do canto superior esquerdo da tela em relação ao monitor. O comando `t.speed()` controla a velocidade com que o desenho é feito.

Atividade 09

Crie um script que desenhe um retângulo cujas medidas de comprimento e largura sejam, respectivamente, 300 px e 250 px, no canto superior esquerdo do monitor. O retângulo deve ser quadriculado, com quadrados de 40 px de lado.

Uma solução:

```
comp, larg = 300, 250
import turtle
s = turtle.Screen()
s.setup(width = comp, height = larg, startx = 0, starty = 0)
t = turtle.Turtle()
t.speed(0)
unid = 40
xmin = -unid*(comp//unid)/2
xmax = -xmin
ymin = -unid*(larg//unid)/2
ymax = -ymin

"Linhas horizontais"
for i in range(larg//unid + 1):
    t.penup()
    t.goto(xmin, ymin + unid*i)
    t.pendown()
    t.setx(xmax)

"Linhas verticais"
for i in range(comp//unid + 1):
    t.penup()
    t.goto(xmin + unid*i, ymin)
    t.pendown()
    t.sety(ymax)
```

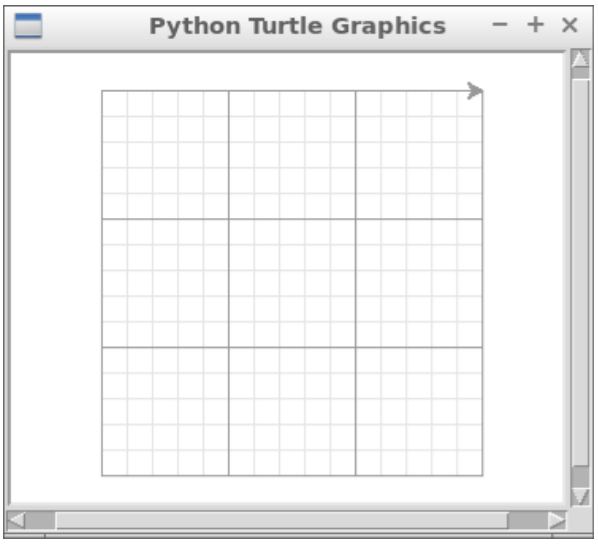
Apesar do resultado simples, a resolução desta atividade exige mais habilidades de aritmética. Cada quadrado do quadriculado deve ter 40 px de lado. Guardamos isso na varável unid (unidade). A esta altura, os estudantes já sabem que a tela criada com o comando `s = turtle.Screen()` tem a origem (0, 0) no seu centro, portanto

a abcissa da reta vertical mais a esquerda, que estamos indicando por *xmin* será a metade da diferença entre a largura e o resto da divisão da largura pela *unid*. Raciocínio semelhante para a ordenada *ymin*. A compreensão desta atividade reforça os conhecimentos de divisão euclidiana e congruência modular.

Atividade 10

Altere a resolução da atividade anterior de modo que cada quadriculado, agora com 80 px, seja dividido em 5 quadriculados menores com linhas da cor cinza (Figura 18).

Figura 18: Este quadriculado, apesar de simples, é um exercício que agrega muita informação.



Fonte: Autor

Uma solução:

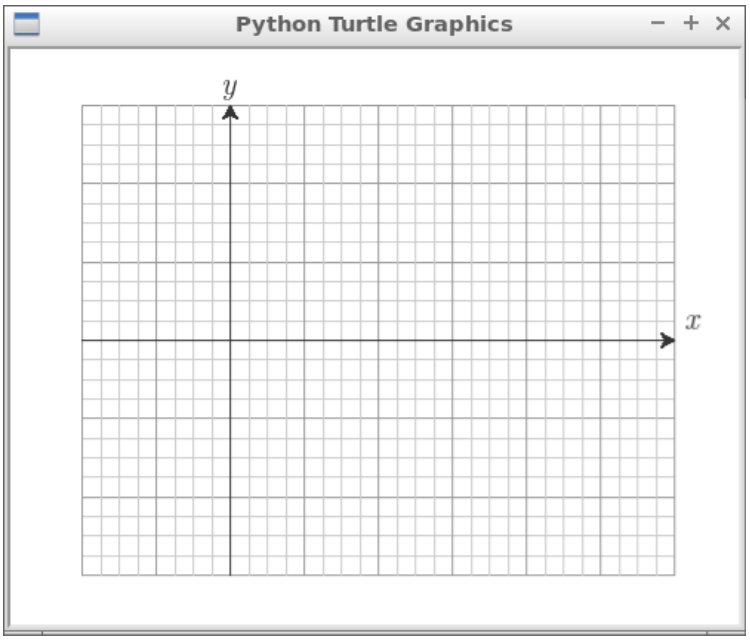
```
comp, larg = 300, 250
import turtle
s = turtle.Screen()
s.setup(width = comp, height = larg, startx = 0, starty = 0)
t = turtle.Turtle()
t.speed(0)
unid = 80
subunid = unid/5
```

```
xmin = -unid*(comp//unid)/2
xmax = -xmin
ymin = -unid*(larg//unid)/2
ymax = -ymin
for i in range(5*(larg//unid)):
    t.penup()
    t.goto(xmin, ymin + subunid*i)
    t.pendown()
    t.setx(xmax)

for i in range(5*(comp//unid)):
    t.penup()
    t.goto(xmin + subunid*i, ymin)
    t.pendown()
    t.sety(ymax)

t.color("#999999")
for i in range(larg//unid + 1):
    t.penup()
    t.goto(xmin, ymin + unid*i)
    t.pendown()
    t.setx(xmax)

for i in range(comp//unid + 1):
    t.penup()
    t.goto(xmin + unid*i, ymin)
    t.pendown()
    t.sety(ymax)
```

Atividade 11

Nesta atividade, vamos desenhar a representação da região retangular do plano limitada pelos pontos (–2, -3) e (6, 3) (Figura 19). A tela deve ter 500 px de largura por 400 px de altura. O quadriculado deve ser formado por linhas de unidade em unidade, com linhas intermediárias com intervalos de 0,25. Devem ser preservadas margens de 10% das dimensões. Os eixos cartesianos devem ser destacados.

Figura 19: Desenho a ser gerado na atividade 11.



Fonte: Autor

Uma solução:

```
comp, larg = 500, 400
import turtle
s = turtle.Screen()
s.setup(width = comp, height = larg, startx = 0, starty = 0)
t = turtle.Turtle()
t.speed(0)

A, B = (-2, -3), (6, 3)

def Tx(x):
    return .8*((x - A[0])/(B[0] - A[0]) - .5)*comp

def Ty(y):
    return .8*((y - A[1])/(B[1] - A[1]) - .5)*larg

# Linhas horizontais
t.color("#cccccc")
for i in range(4*(B[1] - A[1]) + 1):
    x = Tx(A[0])
    y = Ty(A[1]) + .25*.8*i*larg/(B[1] - A[1])
```

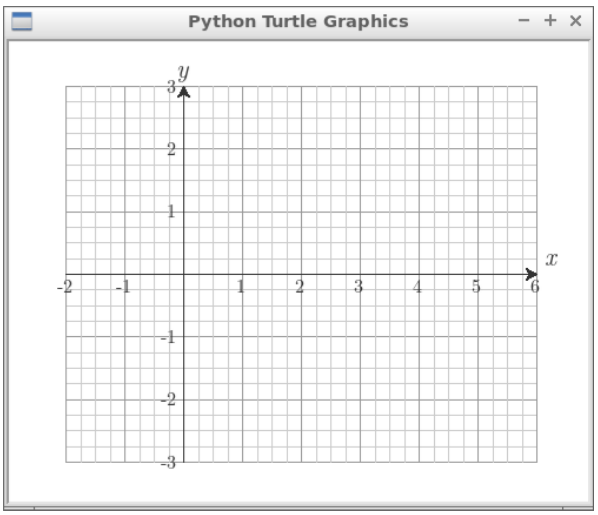
```
    t.penup()
    t.goto(x, y)
    x = Tx(B[0])
    t.pendown()
    t.setx(x)
t.color("#999999")
for i in range(B[1] - A[1] + 1):
    x = Tx(A[0])
    y = Ty(A[1]) + .8*i*larg/(B[1] - A[1])
    t.penup()
    t.goto(x, y)
    x = Tx(B[0])
    t.pendown()
    t.setx(x)

# Linhas verticais
t.color("#cccccc")
for i in range(4*(B[0] - A[0]) + 1):
    x = Tx(A[0]) + .25*.8*i*comp/(B[0] - A[0])
    y = Ty(A[1])
    t.penup()
    t.goto(x, y)
    y = Ty(B[1])
    t.pendown()
    t.sety(y)

t.color("#999999")
for i in range(B[0] - A[0] + 1):
    x = Tx(A[0]) + .8*i*comp/(B[0] - A[0])
    y = Ty(A[1])
    t.penup()
    t.goto(x, y)
    y = Ty(B[1])
    t.pendown()
    t.sety(y)

t.color("#333333")
# Eixo X
t.penup()
t.goto(Tx(A[0]), Ty(0))
t.pendown()
t.setx(Tx(B[0]))
```

```
t.stamp()
t.write(" x", False, align = "left",
        font = ("Latin Modern Roman", 16, "italic"))

# Eixo Y
t.penup()
t.goto(Tx(0), Ty(A[1]))
t.pendown()
t.sety(Ty(B[1]))
t.write("y", False, align = "center",
        font = ("Latin Modern Roman", 16, "italic"))
t.left(90)
t.stamp()
```

Atividade 12

Nesta atividade, você deve graduar os eixos cartesianos do plano da atividade anterior (Figura 20). Acrescente o código abaixo ao script da atividade 11.

Figura 20: Eixos graduados.



Fonte: Autor

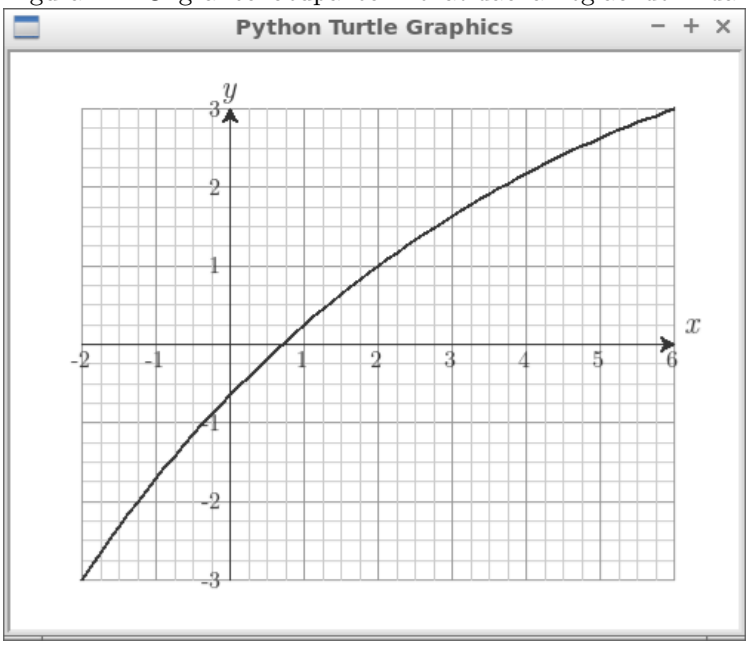
```
# Graduacao no eixo X
t.penup()
y = Ty(-.4)
for i in range(B[0] - A[0] + 1):
    x = Tx(A[0]) + .8*i*comp/(B[0] - A[0])
    t.goto(x, y)
```

```
    if A[0] + i != 0:
        t.write(A[0] + i, False, align = "center",
                font = ("Latin Modern Roman", 12, "normal"))
# Graduacao no eixo Y
t.penup()
x = Tx(-.1)
for i in range(B[1] - A[1] + 1):
    y = Ty(A[1] - .2) + .8*i*larg/(B[1] - A[1])
    t.goto(x, y)
    if A[1] + i != 0:
        t.write(A[1] + i, False, align = "right",
                font = ("Latin Modern Roman", 12, "normal"))
```

Atividade 13

Desenhe o gráfico de $f(x) = -12 + \sqrt{250 - (x-11)^2}$, com $x \in [-2, 6]$ no plano da atividade anterior (Figura 21). Acrescente as linhas abaixo ao script da página anterior.

Figura 21: O gráfico ocupa com exatidão a região definida.



Fonte: Autor

```
from math import sqrt
def f(p):
    return -12 + sqrt(250 - (p - 11)**2)

t.goto(Tx(A[0]), Ty(f(A[0])))
t.pensize(2)
t.pendown()
for i in range(8*(B[0] - A[0]) + 1):
    x = Tx(A[0]) + .125*.8*i*comp/(B[0] - A[0])
    y = Ty(f(A[0] + .125*i))
    t.goto(x, y)
```

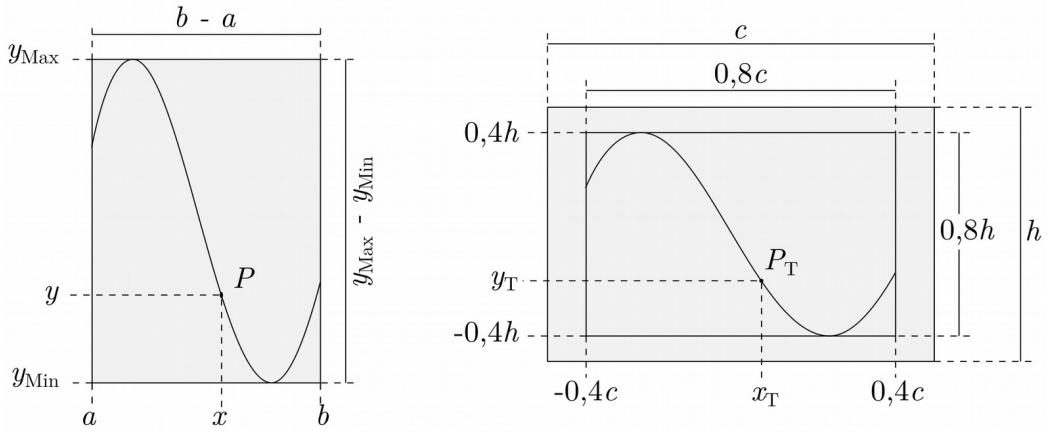
5. CONSTRUÇÃO DO APLICATIVO

Neste capítulo, descreveremos todo o processo de desenvolvimento do aplicativo que gera gráfico de funções, desde o seu projeto até a sua implementação em Python.

Queremos que o nosso aplicativo seja capaz de reproduzir o gráfico de uma função na região da tela de dimensões definidas pelo usuário (Figura 22). O intervalo de domínio da função também deve ser definido pelo usuário. O aplicativo deve ser capaz de ajustar o conjunto imagem da função à altura da tela.

Seja f a função que queremos desenhar. Chamaremos de a e b os extremos, inferior e superior, do intervalo do domínio de f, y_{Max} e y_{Min} são o maior e o menor valor de $f(x)$, com $x \in [a, b]$. c e h são as medidas, em pixels, das dimensões da tela, definidas pelo usuário.

Figura 22: (Imagem feita no Inkscape pelo autor) À esquerda, temos destacado um ponto P no sistema de coordenadas cartesianas padrão. À direita, temos o ponto correspondente a P, P_{T}, no sistema ortogonal do ambiente turtle.



Fonte: Autor

5.1. MUDANÇA DE SISTEMA DE COORDENADAS

Já vimos que a tela de desenho do turtle é apenas um retângulo branco, a origem desse plano fica no seu centro, e as dimensões da área visível dependem das

dimensões do monitor. O que queremos aqui é definir as funções que levem um ponto P do sistema de coordenadas padrão ao ponto P_{T} do sistema definido na tela de desenho. Para simplificar o processo, criaremos duas funções, $T_{\mathrm{x}}(x) = x_{\mathrm{T}}$, e $T_{\mathrm{y}}(y) = y_{\mathrm{T}}$. Estas funções podem ser calculadas facilmente com as proporções abstraídas da Figura .

Para calcular $T_{\mathrm{x}}(x)$, fazemos:

$$\frac{x-a}{b-a} = \frac{x_T + 0,4c}{0,8c} \quad \Longrightarrow \quad x_T + 0,4c = \frac{0,8c}{b-a}(x-a) \quad \Longrightarrow \quad x_T = 0,8\left(\frac{x-a}{b-a} - \frac{1}{2}\right)c$$

De modo análogo, $y_T = 0,8\left(\dfrac{y - y_{Min}}{y_{Max} - y_{Min}} - \dfrac{1}{2}\right)h.$

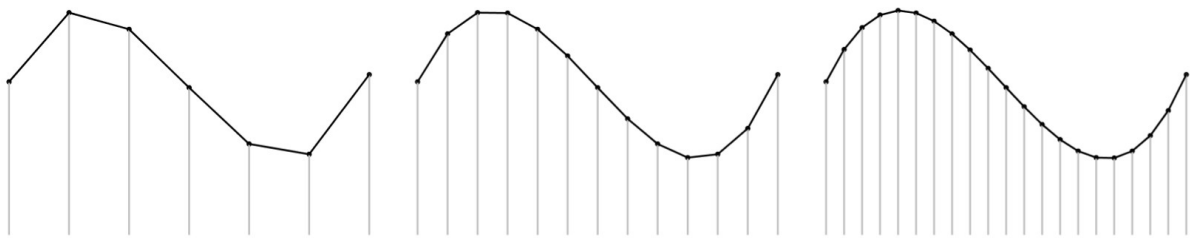
Em Python, estas funções são escritas assim:

```
def Tx(x):
    return .8*((x - a)/(b - a) - .5)*c

def Ty(y):
    return .8*((y - yMin)/(yMax - yMin) - .5)*h
```

5.2. ESCOLHA DA RESOLUÇÃO DO GRÁFICO

Outra observação importante é que, embora queiramos representar funções reais contínuas em $[a,\ b]$, de qualquer tipo (polinomiais, trigonométricas), os gráficos que vamos gerar são, na verdade, apenas poligonais. Subdivisões maiores em $[a,\ b]$ darão a ilusão de que os gráficos são curvas (Figura 22).

Figura 23: Neste exemplo, temos o gráfico de uma mesma função $f(x)$, com x no mesmo intervalo $[a,\ b]$. À esquerda, o intervalo foi dividido em 6 partes iguais; ao centro, em 12 partes; e à direita, em 20 partes, o que dá a impressão de que se trata de uma curva. (Desenho feito no GeoGebra e editado no Inkscape).



Fonte: Autor

Precisaremos repartir o intervalo $[a,\ b]$ em n segmentos de medidas iguais. Portanto, interpolaremos entre a e b números $x_i = a + \dfrac{(b-a)}{n}i$, com $i \in \mathbb{N},\ 0 \leq i \leq n$. Abaixo, o fragmento do script que implementa a função:

```
def f(x):
    return < Uma sentença da função >

a = float(input("Informe o ínfimo do domínio: "))
b = float(input("Informe o supremo do domínio: "))
n = 50
X, Y = [a], [f(a)]
for i in range(n):
    x = a + (i + 1)*(b - a)/n
    y = f(x)
    X.append(x)
    Y.append(y)
```

Aqui, definimos que o intervalo $[a,\ b]$ vai ser repartido em 50 segmentos de mesma medida (`n = 50`). Em seguida, criamos duas listas X e Y, com um elemento inicial em cada, a e $f(x)$, respectivamente. Depois, no laço `for`, x e y são atualizados e adicionados as suas respectivas listas X e Y.

Já definimos como armazenar as coordenadas do gráfico. Observe que ainda são os pontos no plano cartesiano canônico, não os transformados por $T_{\mathrm{x}}(x) = x_{\mathrm{T}}$, e $T_{\mathrm{y}}(y) = y_{\mathrm{T}}$. A transformação ocorrerá no momento em que o gráfico estiver sendo desenhado pelos comandos abaixo:

```
t.penup()
t.goto(Tx(X[0]), Ty(Y[0]))
t.pendown()
for i in range(1, len(X)):
    t.goto(Tx(X[i]), Ty(Y[i]))
```

Todos estes comandos já foram explicados em atividades anteriores.

O usuário poderá escolher em quanto será divido os intervalos de uma unidade no quadriculado do plano cartesiano, horizontal e verticalmente, semelhante ao que foi pedido na **atividade 10**. Se ele se omitir será atribuído uma divisão em 4. O trecho de código abaixo é responsável por essa funcionalidade.

```
yMin, yMax = min(Y), max(Y)

subX = input("Número de subdivisões do eixo X (1, 2, 4 ou 5): ")
if subX == '':
    subX = 4
else:
    subX = int(subX)

subY = input("Número de subdivisões do eixo Y (1, 2, 4 ou 5): ")
if subY == '':
    subY = 4
else:
    subY = int(subY)
```

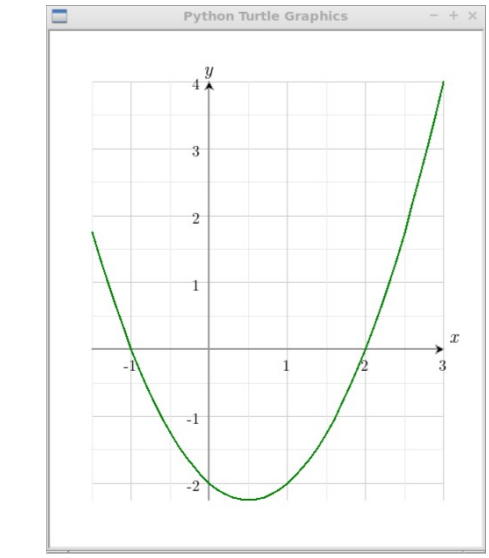
De modo semelhante, ao ser solicitado que o usuário escolha as dimensões da tela, caso ele se omita, estas dimensões serão atribuídas pelo programa. No código abaixo, fica definido que as dimensões padrão serão 500×400 pixel^2.

```
c = input("Comprimento da tela: ")
if c == '':
    c = 500
else:
    c = int(c)
h = input("Altura da tela: ")
if h == '':
    h = 400
else:
    h = int(h)

import turtle
s = turtle.Screen()
s.setup(width = c, height = h, startx = 0, starty = 0)
```

Vários trechos do script serão omitidos, porque a sua descoberta é um desafio interessante para os estudantes. Até a conclusão deste texto, muitos requisitos desejáveis deixaram de ser atendidos. Não chegou-se a implementar uma interface gráfica. As funções ainda precisam ser definidas no código. Seria interessante se o usuário pudesse escrevê-las ao ser solicitado pela função `input()`.

Encerra-se este capítulo com alguns gráficos gerados por nosso programa.

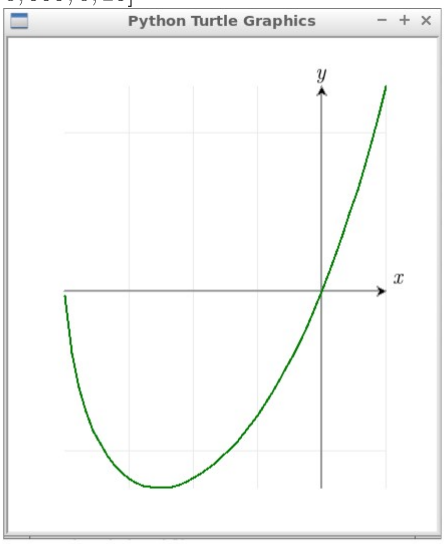
Figura 24: Gráfico de $f(x) = x^2 - x - 2$.




```
Informe o ínfimo do domínio: -1.5
Informe o supremo do domínio: 3
Número de subdivisões do eixo X (1, 2, 4 ou 5): 2
Número de subdivisões do eixo Y (1, 2, 4 ou 5): 2
Comprimento da tela: 500
Altura da tela: 600
>>>
Ln: 12  Col: 4
```

Fonte: Autor

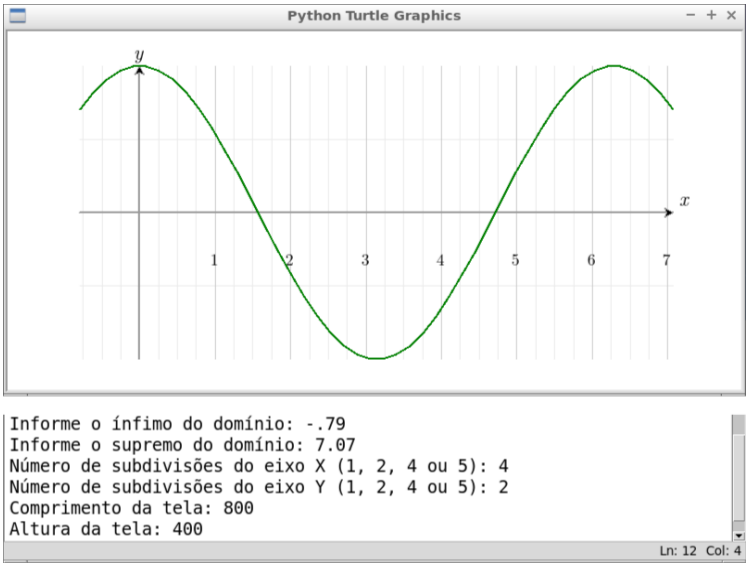
Figura 25: Gráfico de $(x + 1)^{(x + 1)} - 1$, com $x \in [-0,999; 0,25]$




```
Informe o ínfimo do domínio: -.999
Informe o supremo do domínio: .25
Número de subdivisões do eixo X (1, 2, 4 ou 5): 4
Número de subdivisões do eixo Y (1, 2, 4 ou 5): 4
Comprimento da tela: 400
Altura da tela: 500
>>>
Ln: 64  Col: 49
```

Fonte: Autor

Figura 26: Gráfico de $f(x) = \text{sen}(x)$, com $x \in \left[-\frac{\pi}{4}, \frac{9\pi}{4}\right]$.



Fonte: Autor

6. CONSIDERAÇÕES FINAIS

Os recursos de informática, sejam eles os tradicionais softwares educativos ou os ambientes de programação, são apenas ferramentas para o processo de ensino e aprendizagem. Porém, quando bem aplicados, oferecem muitas vantagens. Esta experiência de introdução a programação com Python mostrou-se positiva em facilitar a aprendizagem de Matemática, ou pelo menos, de validar as soluções de problemas encontradas da maneira tradicional. Na maioria das vezes, os estudantes concluem a resolução de um problema com a dúvida de se realmente chegaram a resposta correta. A programação oferece uma maneira diferente de validarem suas soluções, pois computadores são intolerantes a instruções inconsistentes, o que obriga os estudantes a serem mais atenciosos. No processo de programar uma verificação matemática, o aluno adquire uma nova camada de compreensão do problema em questão.

Não foi possível estender plenamente estes estudos a todos os estudantes das turmas, devido a pequena quantidade de computadores disponível na sala de informática da escola. Há várias opções de ambientes de programação Python para celulares com Android (o mais comum entre os estudantes), mas estes aparelhos ainda têm várias limitações, se comparados a computadores desktop. O período de um semestre é curto pra contemplar todas as atividades descritas neste trabalho. Neste ano de 2018, a Secretaria Estadual de Educação determinou a antecipação do ano letivo, o que encurtou o intervalo entre a 3ª e a 4ª avaliação e a recuperação. Isso comprometeu um pouco as oficinas, e também impossibilitou uma pesquisa mais rigorosa dos resultados do projeto. No entanto, a programação foi apenas um instrumento a mais no ensino de Matemática, e não o objetivo principal.

O projeto se mostrou mais dinâmico com os estudantes que se dispuseram a participar no contraturno. Aplicá-lo em turmas inteiras foi desafio bem maior, mas verificou-se também, alguns ganhos na melhoria da compreensão matemática e do raciocínio lógico. A conclusão a que se chega é que vale a pena introduzir o estudo da linguagem de programação Python, com o objetivo de melhorar a aprendizagem de Matemática. Na escola estadual Lameira Bittencourt, em Castanhal-PA, a proposta já está inserida no seu Plano de Ação de 2019, desta vez contemplando não apenas alunos, como também os professores de Matemática da escola.

REFERÊNCIAS

[1] ALMEIDA, Marcus Garcia, [et. al]. Desafios permanentes: projeto político pedagógico, gestão escolar, métricas no contexto das TICs. Vol. 4. Rio de Janeiro: Brasport, 2015.

[2] Base Nacional Comum Curricular, 2018.

[3] BRIGGS, Jason R. Python for Kids: a playful introduction to programming. 1ª ed. San Francisco: No Starch Press, 2013.

[4] COURANT, Richard. Calculo Diferencial e Integral. Vol. 1. 1ª ed. Porto Alegre: Editora Globo, 1963.

[5] FEIO, Evandro dos Santos Paiva. Matemática e linguagem: um enfoque na conversão da língua natural para a linguagem matemática. 2009. 101 f. Dissertação (Mestrado) – Universidade Federal do Pará, Instituto de Educação Matemática e Científica, Belém, 2009. Programa de Pós-Graduação em Educação em Ciências e Matemáticas.

[6] GOMES, Jonas; VELHO, Luiz. Fundamentos da Computação Gráfica. Rio de Janeiro: Impa, 2003.

[7] IEZZI, Gerson, [et. al]. Matemática: ciência e aplicações, ensino médio. Vol. 2. São Paulo: Saraiva, 2016.

[8] PERKOVIC, Ljubomir. Introdução à computação usando Python: um foco no desenvolvimento de aplicações. 1ª ed. Rio de Janeiro: LTC, 2016.

[9] SCHEINERMAN, Edward R. Matemática discreta: uma introdução. São Paulo: Cengage Learning, 2016.

[10] https://www.edsurge.com/news/2016-05-31-how-programming-supports-math-class-not-the-other-way-around (acessado em 14/11/2018).

[11] https://novaescola.org.br/conteudo/5344/a-base-quer-a-tecnologia-na-sua-disciplina-e-agora (acessado em 11/03/2019).