Comandos de Repetição

MC102

Motivação

Imprimindo números... na mão?

Digamos que você quer imprimir números consecutivos.

Imprimindo números... na mão?

Digamos que você quer imprimir números consecutivos.

Queremos imprimir 3 números:

```
1 n = int(input())
2 print(n)
3 print(n + 1)
4 print(n + 2)
```

Imprimindo números... na mão?

Digamos que você quer imprimir números consecutivos.

Queremos imprimir 3 números:

```
1 n = int(input())
2 print(n)
3 print(n + 1)
4 print(n + 2)
```

E se forem 5?

```
1 n = int(input())
2 print(n)
3 print(n + 1)
4 print(n + 2)
5 print(n + 3)
6 print(n + 4)
```

Imprimindo números... na mão?

Digamos que você quer imprimir números consecutivos.

Queremos imprimir 3 números:

```
1 n = int(input())
2 print(n)
3 print(n + 1)
4 print(n + 2)
```

E se forem 5?

```
1 n = int(input())
2 print(n)
3 print(n + 1)
4 print(n + 2)
5 print(n + 3)
6 print(n + 4)
```

E se quisermos imprimir **100** números?

Imprimindo números... na mão?

Digamos que você quer imprimir números consecutivos.

Queremos imprimir 3 números:

```
1 n = int(input())
2 print(n)
3 print(n + 1)
4 print(n + 2)
```

E se forem 5?

```
1 n = int(input())
2 print(n)
3 print(n + 1)
4 print(n + 2)
5 print(n + 3)
6 print(n + 4)
```

E se quisermos imprimir **100** números?

Ou pior, e se a quantidade depender de uma entrada do usuário `k`?

Imprimindo números... na mão?

Digamos que você quer imprimir números consecutivos.

Queremos imprimir 3 números:

```
1 n = int(input())
2 print(n)
3 print(n + 1)
4 print(n + 2)
```

E se forem 5?

```
1 n = int(input())
2 print(n)
3 print(n + 1)
4 print(n + 2)
5 print(n + 3)
6 print(n + 4)
```

E se quisermos imprimir **100** números?

Ou pior, e se a quantidade depender de uma entrada do usuário `k`?

Isso fere o princípio **DRY** (*Don't Repeat Yourself*). Se você está dando `CTRL+C CTRL+V` no seu código, pare.

✋ Busque ajuda!

Menor número

Dado 2 números, queremos saber qual é o menor.

```
a = int(input())
b = int(input())
if a <= b:
    print(a)
else:
    print(b)
```

Menor número

Dado 2 números, queremos saber qual é o menor.

```
1 a = int(input())
2 b = int(input())
3 if a <= b:
4     print(a)
5 else:
6     print(b)
```

Dados 3 números, queremos saber qual é o menor.

```
1 a = int(input())
2 b = int(input())
3 c = int(input())
4 if a <= b and a <= c:
5     print(a)
6 elif b <= a and b <= c:
7     print(b)
8 else:
9     print(c)
```

Menor número

Dado 2 números, queremos saber qual é o menor.

```
a = int(input())
b = int(input())
if a <= b:
    print(a)
else:
    print(b)
```

Dados 3 números, queremos saber qual é o menor.

```
a = int(input())
b = int(input())
c = int(input())
if a <= b and a <= c:
    print(a)
elif b <= a and b <= c:
    print(b)
else:
    print(c)
```

Dados 100 números, queremos qual é o menor.

Menor número

Dado 2 números, queremos saber qual é o menor.

```
a = int(input())
b = int(input())
if a <= b:
    print(a)
else:
    print(b)
```

Dados 3 números, queremos saber qual é o menor.

```
a = int(input())
b = int(input())
c = int(input())
if a <= b and a <= c:
    print(a)
elif b <= a and b <= c:
    print(b)
else:
    print(c)
```

Dados 100 números, queremos qual é o menor.

- nem f*dendo que eu vou escrever o código....

While

O comando `while`

Para repetir ações, usamos laços (loops). O mais fundamental é o `while`.

O comando `while`

Para repetir ações, usamos laços (loops). O mais fundamental é o `while`.

</> Code

Dado n e k, queremos imprimir os k números consecutivos começando em n.

O comando `while`

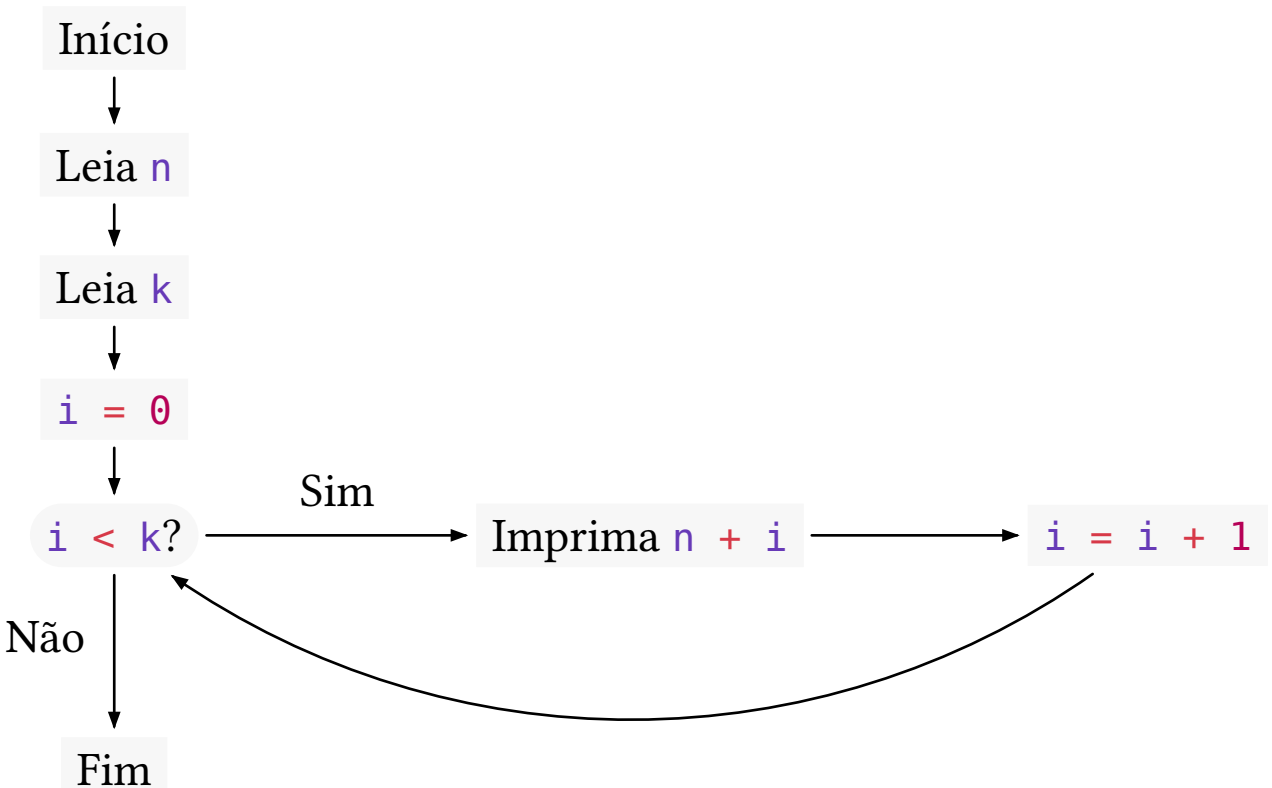
Para repetir ações, usamos laços (loops). O mais fundamental é o `while`.

</> Code

Dado n e k, queremos imprimir os k números consecutivos começando em n.

```
Leia n
Leia k
i = 0
Enquanto i < k faça:
    Imprima n + i
    Acrescente 1 à i
```

Fluxograma



O comando `while`

```
1 Leia n
2 Leia k
3 i = 0
4 Enquanto i < k faça:
5     Imprima n + i
6     Acrescente 1 à i
```

O comando `while`

```
Leia n
Leia k
i = 0
Enquanto i < k faça:
    Imprima n + i
    Acrescente 1 à i
```

```
n = int(input())
k = int(input())
i = 0
while i < k:
    print(n + i)
    i += 1
```

O comando `while`

```
Leia n
Leia k
i = 0
Enquanto i < k faça:
    Imprima n + i
    Acrescente 1 à i
```

```
n = int(input())
k = int(input())
i = 0
while i < k:
    print(n + i)
    i += 1
```

O bloco indentado será executado **enquanto** a condição for `True`.

O comando `while`

```
Leia n
Leia k
i = 0
Enquanto i < k faça:
    Imprima n + i
    Acrescente 1 à i
```

```
n = int(input())
k = int(input())
i = 0
while i < k:
    print(n + i)
    i += 1
```

O bloco indentado será executado **enquanto** a condição for `True`.

A sintaxe do `while` é muito similar a do `if`:

O comando `while`

```
Leia n
Leia k
i = 0
Enquanto i < k faça:
    Imprima n + i
    Acrescente 1 à i
```

```
n = int(input())
k = int(input())
i = 0
while i < k:
    print(n + i)
    i += 1
```

O bloco indentado será executado **enquanto** a condição for `True`.

A sintaxe do `while` é muito similar a do `if`:

- O `:` indica o começo do bloco

O comando `while`

```
1 Leia n
2 Leia k
3 i = 0
4 Enquanto i < k faça:
5     Imprima n + i
6     Acrescente 1 à i
```

```
1 n = int(input())
2 k = int(input())
3 i = 0
4 while i < k:
5     print(n + i)
6     i += 1
```

O bloco indentado será executado **enquanto** a condição for `True`.

A sintaxe do `while` é muito similar a do `if`:

- O `:` indica o começo do bloco
- Que tem que ser identado

O comando `while`

```
Leia n
Leia k
i = 0
Enquanto i < k faça:
    Imprima n + i
    Acrescente 1 à i
```

```
n = int(input())
k = int(input())
i = 0
while i < k:
    print(n + i)
    i += 1
```

O bloco indentado será executado **enquanto** a condição for `True`.

A sintaxe do `while` é muito similar a do `if`:

- O `:` indica o começo do bloco
- Que tem que ser identado

Atenção especial ao `x += 1`:

O comando `while`

```
Leia n
Leia k
i = 0
Enquanto i < k faça:
    Imprima n + i
    Acrescente 1 à i
```

```
n = int(input())
k = int(input())
i = 0
while i < k:
    print(n + i)
    i += 1
```

O bloco indentado será executado **enquanto** a condição for `True`.

A sintaxe do `while` é muito similar a do `if`:

- O `:` indica o começo do bloco
- Que tem que ser identado

Atenção especial ao `x += 1`:

- É uma forma mais compacta de escrever `x = x + 1`.

O comando `while`

```
Leia n
Leia k
i = 0
Enquanto i < k faça:
    Imprima n + i
    Acrescente 1 à i
```

```
n = int(input())
k = int(input())
i = 0
while i < k:
    print(n + i)
    i += 1
```

O bloco indentado será executado **enquanto** a condição for `True`.

A sintaxe do `while` é muito similar a do `if`:

- O `:` indica o começo do bloco
- Que tem que ser identado

Atenção especial ao `x += 1`:

- É uma forma mais compacta de escrever `x = x + 1`.
- Existem equivalentes para outras oprações (`-=`, `*=`, `/=`, etc), mas são menos usados.

O comando `while`

```
Leia n
Leia k
i = 0
Enquanto i < k faça:
    Imprima n + i
    Acrescente 1 à i
```

```
n = int(input())
k = int(input())
i = 0
while i < k:
    print(n + i)
    i += 1
```

O bloco indentado será executado **enquanto** a condição for `True`.

A sintaxe do `while` é muito similar a do `if`:

- O `:` indica o começo do bloco
- Que tem que ser identado

Atenção especial ao `x += 1`:

- É uma forma mais compacta de escrever `x = x + 1`.
- Existem equivalentes para outras oprações (`-=`, `*=`, `/=`, etc), mas são menos usados.

Bugs 

Bugs 

Como saber se o código faz o que queremos?

Bugs 

Como saber se o código faz o que queremos? E se cairmos num **loop infinito**?

Bugs 

Como saber se o código faz o que queremos? E se cairmos num **loop infinito**? Uma ferramenta poderosa (e analógica) é o **Teste de Mesa**.

Bugs 

Como saber se o código faz o que queremos? E se cairmos num **loop infinito**? Uma ferramenta poderosa (e analógica) é o **Teste de Mesa**.

- Simular o programa usando papel e lápis

Bugs 

Como saber se o código faz o que queremos? E se cairmos num **loop infinito**? Uma ferramenta poderosa (e analógica) é o **Teste de Mesa**.

- Simular o programa usando papel e lápis
- Para alguns valores de entrada

Bugs 

Como saber se o código faz o que queremos? E se cairmos num **loop infinito**? Uma ferramenta poderosa (e analógica) é o **Teste de Mesa**.

- Simular o programa usando papel e lápis
- Para alguns valores de entrada

É importante olhar também para os casos mais críticos, chamados “edge cases” ou casos de borda:

Bugs 

Como saber se o código faz o que queremos? E se cairmos num **loop infinito**? Uma ferramenta poderosa (e analógica) é o **Teste de Mesa**.

- Simular o programa usando papel e lápis
- Para alguns valores de entrada

É importante olhar também para os casos mais críticos, chamados “edge cases” ou casos de borda:

- E se k ou n for zero?

Bugs 

Como saber se o código faz o que queremos? E se cairmos num **loop infinito**? Uma ferramenta poderosa (e analógica) é o **Teste de Mesa**.

- Simular o programa usando papel e lápis
- Para alguns valores de entrada

É importante olhar também para os casos mais críticos, chamados “edge cases” ou casos de borda:

- E se k ou n for zero?
- E se k ou n forem negativos?

Bugs 

Como saber se o código faz o que queremos? E se cairmos num **loop infinito**? Uma ferramenta poderosa (e analógica) é o **Teste de Mesa**.

- Simular o programa usando papel e lápis
- Para alguns valores de entrada

É importante olhar também para os casos mais críticos, chamados “edge cases” ou casos de borda:

- E se k ou n for zero?
- E se k ou n forem negativos?

Bugs 

```
1 n = int(input())
2 k = int(input())
3 i = 0
4 while i < k:
5     print(n + i)
6     i += 1
```

Linha	n	k	i	
1	2			
2	2	3		
3	2	3	0	
4	2	3	0	`0 < 3`
5	2	3	0	imprime `2`
6	2	3	1	
4	2	3	1	`1 < 3`
5	2	3	1	imprime `3`
6	2	3	2	
4	2	3	2	`2 < 3`
5	2	3	2	imprime `4`
6	2	3	3	
4	2	3	3	`3 == 3`

Bugs 🪳 vs Duck 🦆

Pode parecer estranho, mas acompanha comigo:

Bugs 🪳 vs Duck 🦆

Pode parecer estranho, mas acompanha comigo:

- Você já deve ter ouvido falar coisas como “a melhor forma de entender algo é ensinar para alguém”.

Bugs 🪳 vs Duck 🦆

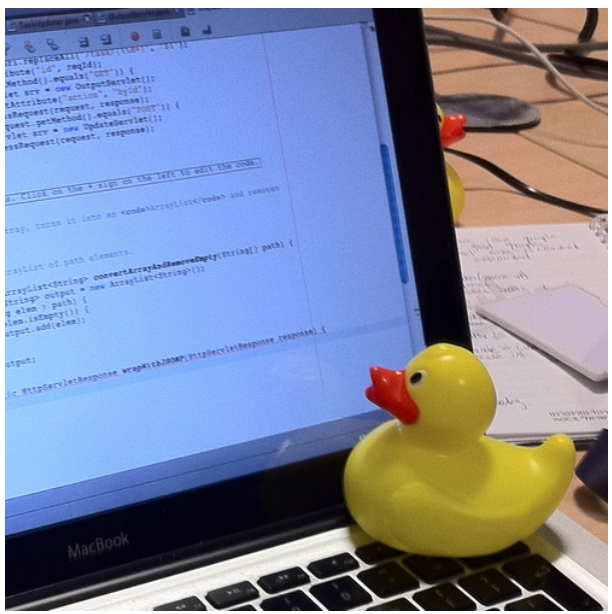
Pode parecer estranho, mas acompanha comigo:

- Você já deve ter ouvido falar coisas como “a melhor forma de entender algo é ensinar para alguém”.
- Isso é tão verdade na computação que possui um nome específico: **duck debugging**

Bugs 🪳 vs Duck 🦆

Pode parecer estranho, mas acompanha comigo:

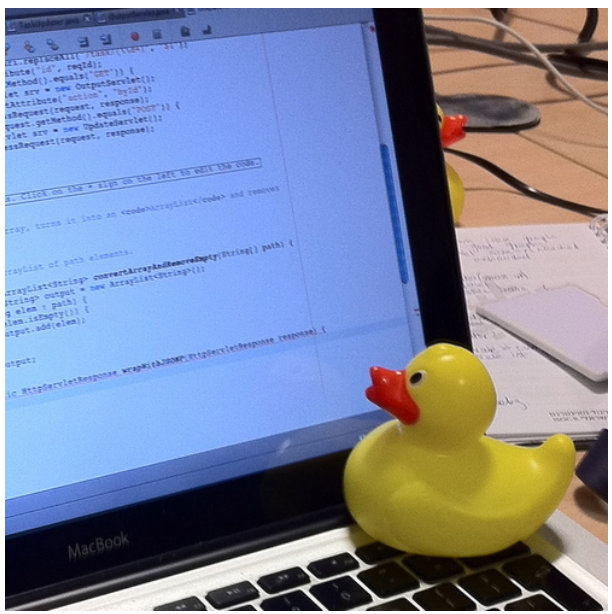
- Você já deve ter ouvido falar coisas como “a melhor forma de entender algo é ensinar para alguém”.
- Isso é tão verdade na computação que possui um nome específico: **duck debugging**



Bugs 🪳 vs Duck 🦆

Pode parecer estranho, mas acompanha comigo:

- Você já deve ter ouvido falar coisas como “a melhor forma de entender algo é ensinar para alguém”.
- Isso é tão verdade na computação que possui um nome específico: **duck debugging**

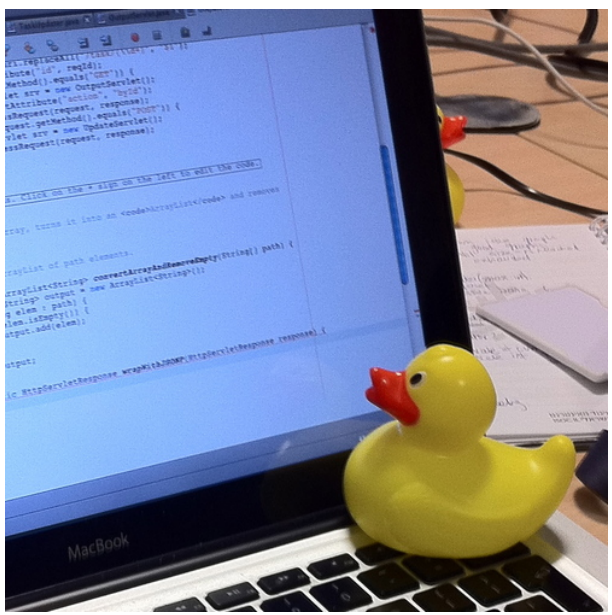


A ideia é ter um colega (ou um objeto, como um pato de borracha) para qual você possa explicar, em detalhes, o problema que você está tendo.

Bugs 🪳 vs Duck 🦆

Pode parecer estranho, mas acompanha comigo:

- Você já deve ter ouvido falar coisas como “a melhor forma de entender algo é ensinar para alguém”.
- Isso é tão verdade na computação que possui um nome específico: **duck debugging**



A ideia é ter um colega (ou um objeto, como um pato de borracha) para qual você possa explicar, em detalhes, o problema que você está tendo.

Chances são que, quando você terminar de explicar, alguma ideia salte à sua cabeça.

Voltando ao problema

Vimos esse código:

```
1 n = int(input())
2 k = int(input())
3 i = 0
4 while i < k:
5     print(n + i)
6     i += 1
```

Mas existem outras formas de fazer:

```
1 n = int(input())
2 k = int(input())
3 atual = n
4 while atual < n + k:
5     print(atual)
6     atual += 1
```

Progressão Aritmética

Progressão Aritmética

Relembrando uma Progressão Aritmética (PA):

Progressão Aritmética

Relembrando uma Progressão Aritmética (PA):

- é uma sequência de números $(a_1, a_2, ..., a_n)$

Progressão Aritmética

Relembrando uma Progressão Aritmética (PA):

- é uma sequência de números $(a_1, a_2, ..., a_n)$
- onde existe um número r tal que

Progressão Aritmética

Relembrando uma Progressão Aritmética (PA):

- é uma sequência de números $(a_1, a_2, ..., a_n)$
- onde existe um número r tal que
- $a_{i+1} = a_i + r$

Progressão Aritmética

Relembrando uma Progressão Aritmética (PA):

- é uma sequência de números $(a_1, a_2, ..., a_n)$
- onde existe um número r tal que
- $a_{i+1} = a_i + r$
- para todo $1 \leq i \leq n$

Progressão Aritmética

Relembrando uma Progressão Aritmética (PA):

- é uma sequência de números $(a_1, a_2, ..., a_n)$
- onde existe um número r tal que
- $a_{i+1} = a_i + r$
- para todo $1 \leq i \leq n$

A soma S da progressão aritmética é

$$S = \frac{n(a_1 + a_n)}{2} = a_1 n + \frac{n(n-1)r}{2}$$

Progressão Aritmética

Relembrando uma Progressão Aritmética (PA):

- é uma sequência de números $(a_1, a_2, ..., a_n)$
- onde existe um número r tal que
- $a_{i+1} = a_i + r$
- para todo $1 \leq i \leq n$

A soma S da progressão aritmética é

$$S = \frac{n(a_1 + a_n)}{2} = a_1 n + \frac{n(n-1)r}{2}$$

Ou é o que dizem!

Progressão Aritmética

Relembrando uma Progressão Aritmética (PA):

- é uma sequência de números $(a_1, a_2, ..., a_n)$
- onde existe um número r tal que
- $a_{i+1} = a_i + r$
- para todo $1 \leq i \leq n$

A soma S da progressão aritmética é

$$S = \frac{n(a_1 + a_n)}{2} = a_1 n + \frac{n(n-1)r}{2}$$

Ou é o que dizem!

</> Code

Dado a_1, n e r, imprima a soma elementos da PA correspondente.

Progressão Aritmética

```
a_1 = int(input())
n  = int(input())
r = int(input())
```

Progressão Aritmética

```
a_1 = int(input())
n  = int(input())
r = int(input())
```

```
i = 1
soma = 0
atual = a_1

while i <= n:
    soma += atual
    atual += r
    i = i + 1
```

Progressão Aritmética

```
a_1 = int(input())
n  = int(input())
r = int(input())

i = 1
soma = 0
atual = a_1

while i <= n:
    soma += atual
    atual += r
    i = i + 1

esperado = a_1 * n + n * (n-1) * r / 2

if soma != esperado:
    print("Fórmula incorreta!")
    print(esperado)
    print(soma)
else:
    print("Fórmula correta!")
```

Progressão Aritmética

```
1  a_1 = int(input())
2  n   = int(input())
3  r = int(input())

4  i = 1
5  soma = 0
6  atual = a_1
7
8  while i <= n:
9      soma += atual
10     atual += r
11     i = i + 1

12 esperado = a_1 * n + n * (n-1) * r / 2
13
14 if soma != esperado:
15     print("Fórmula incorreta!")
16     print(esperado)
17     print(soma)
18 else:
19     print("Fórmula correta!")
```

- A variável `i` está *contando* até `n`

Progressão Aritmética

```python
a_1 = int(input())
n  = int(input())
r = int(input())

i = 1
soma = 0
atual = a_1

while i <= n:
    soma += atual
    atual += r
    i = i + 1

esperado = a_1 * n + n * (n-1) * r / 2

if soma != esperado:
    print("Fórmula incorreta!")
    print(esperado)
    print(soma)
else:
    print("Fórmula correta!")
```

- A variável `i` está *contando* até `n`
- A variável `soma` está *acumulando* o resultado.

Primos

Um clássico atemporal: Verificar se é primo.

Primos

Um clássico atemporal: Verificar se é primo. **Lembrando:** Primo só é divisível por 1 e por ele mesmo.

Versão Ingênua:

Testar todos os divisores de 2 até p .

Versão Otimizada: Se p tem um divisor maior que $\sqrt{p}$, obrigatoriamente tem um menor que $\sqrt{p}$.

```
p = int(input())
primo = True
k = 2
while k < p:
    if p % k == 0:
        primo = False
    k += 1
print(primo)
```

Primos

Um clássico atemporal: Verificar se é primo. **Lembrando:** Primo só é divisível por 1 e por ele mesmo.

Versão Ingênua:

Testar todos os divisores de 2 até p .

```
p = int(input())
primo = True
k = 2
while k < p:
    if p % k == 0:
        primo = False
    k += 1
print(primo)
```

Versão Otimizada: Se p tem um divisor maior que $\sqrt{p}$, obrigatoriamente tem um menor que $\sqrt{p}$. Basta testar até $\sqrt{p}$.

Primos

Um clássico atemporal: Verificar se é primo. **Lembrando:** Primo só é divisível por 1 e por ele mesmo.

Versão Ingênua:

Testar todos os divisores de 2 até p .

```
p = int(input())
primo = True
k = 2
while k < p:
    if p % k == 0:
        primo = False
    k += 1
print(primo)
```

Versão Otimizada: Se p tem um divisor maior que $\sqrt{p}$, obrigatoriamente tem um menor que $\sqrt{p}$. Basta testar até $\sqrt{p}$. E se achou um divisor, pare.

Primos

Um clássico atemporal: Verificar se é primo. **Lembrando:** Primo só é divisível por 1 e por ele mesmo.

Versão Ingênua:

Testar todos os divisores de 2 até p .

```
p = int(input())
primo = True
k = 2
while k < p:
    if p % k == 0:
        primo = False
    k += 1
print(primo)
```

Versão Otimizada: Se p tem um divisor maior que $\sqrt{p}$, obrigatoriamente tem um menor que $\sqrt{p}$. Basta testar até $\sqrt{p}$. E se achou um divisor, pare.

```
p = int(input())
primo = True
k = 2
while k * k <= p and primo:
    if p % k == 0:
        primo = False
    k += 1
print(p >= 2 and primo)
```

Primos

Um clássico atemporal: Verificar se é primo. **Lembrando:** Primo só é divisível por 1 e por ele mesmo.

Versão Ingênua:

Testar todos os divisores de 2 até p .

```
p = int(input())
primo = True
k = 2
while k < p:
    if p % k == 0:
        primo = False
    k += 1
print(primo)
```

Versão Otimizada: Se p tem um divisor maior que $\sqrt{p}$, obrigatoriamente tem um menor que $\sqrt{p}$. Basta testar até $\sqrt{p}$. E se achou um divisor, pare.

```
p = int(input())
primo = True
k = 2
while k * k <= p and primo:
    if p % k == 0:
        primo = False
    k += 1
print(p >= 2 and primo)
```

Diferença? Para $p = 27.644.437$: Ingênuo: 5.32s 🐌 Otimizado: 0.06s 🚀

Decomposição

Decomposição

</> Code

Dado um inteiro n, imprima todos os seus divisores.

Decomposição

</> Code

Dado um inteiro n, imprima todos os seus divisores.

```python
n = int(input())
d = 1
while d * d <= n:
    if n % d == 0:
        print(d)
        print(n / d)
    d += 1
```

Decomposição

</> Code

Dado um inteiro n, imprima todos os seus divisores.

```python
n = int(input())
d = 1
while d * d <= n:
    if n % d == 0:
        print(d)
        print(n / d)
    d += 1
```

Mas e se quiséssemos armazenar tais divisores para usarmos em outra conta ou depois?

Decomposição

</> Code

Dado um inteiro n, imprima todos os seus divisores.

```python
n = int(input())
d = 1
while d * d <= n:
    if n % d == 0:
        print(d)
        print(n / d)
    d += 1
```

Mas e se quiséssemos armazenar tais divisores para usarmos em outra conta ou depois?

Como vamos saber de antemão quantas variáveis precisamos criar para guardar essas informações?

Decomposição

</> Code

Dado um inteiro n, imprima todos os seus divisores.

```python
n = int(input())
d = 1
while d * d <= n:
    if n % d == 0:
        print(d)
        print(n / d)
    d += 1
```

Mas e se quiséssemos armazenar tais divisores para usarmos em outra conta ou depois?

Como vamos saber de antemão quantas variáveis precisamos criar para guardar essas informações?

E qual o nome que daríamos a tanta variável?

```
div1
div2
div3
...
```

Listas

Listas

Às vezes precisamos guardar um monte de dados, e criar `var1`, `var2`, `var3`... é inviável. Listas (`list`) resolvem isso.

Listas

Às vezes precisamos guardar um monte de dados, e criar `var1`, `var2`, `var3`... é inviável. Listas (`list`) resolvem isso.

Criando:

```
vazia = []
numeros = [1, 7, 15]
nomes = ["Ana", "Bob"]
mista = [1, "Oi", 3.14]
```

Listas

Às vezes precisamos guardar um monte de dados, e criar `var1`, `var2`, `var3`... é inviável. Listas (`list`) resolvem isso.

Criando:

```
vazia = []
numeros = [1, 7, 15]
nomes = ["Ana", "Bob"]
mista = [1, "Oi", 3.14]
```

Acessando (Índice começa em 0!):

```
l = ["A", "B", "C"]
print(l[0]) # 'A'
print(l[2]) # 'C'
```

Listas

Às vezes precisamos guardar um monte de dados, e criar `var1`, `var2`, `var3`... é inviável. Listas (`list`) resolvem isso.

Criando:

```
vazia = []
numeros = [1, 7, 15]
nomes = ["Ana", "Bob"]
mista = [1, "Oi", 3.14]
```

Acessando (Índice começa em 0!):

```
l = ["A", "B", "C"]
print(l[0]) # 'A'
print(l[2]) # 'C'
```

Danger

Acessar um índice que não existe explode o programa.

```
l = [1, 2]
print(l[5])
```

```
IndexError: list index out of range
```

Operações com Listas

Operações com Listas

Adicionar elementos no final: `.append()`

```
lista = []
lista.append("Primeiro")
lista.append("Segundo")
# lista agora é ["Primeiro", "Segundo"]
```

Operações com Listas

Adicionar elementos no final: `.append()`

```
lista = []
lista.append("Primeiro")
lista.append("Segundo")
# lista agora é ["Primeiro", "Segundo"]
```

Verificar se elemento existe na lista: `in`

```
numeros = [1, 2, 3, 5, 8]
k = int(input())
print(k in numeros) # True ou False
```

Lendo uma lista de nomes

Se tivermos a quantidade n de nomes

Lendo uma lista de nomes

Se tivermos a quantidade n de nomes, podemos:

```
1  n = int(input())    # qtd de nomes
2  l = []              # lista vazia
3  i = 0
4  while i < n:
5      nome = input()  # lê o nome
6      l.append(nome)
7      i += 1
8
9  print("Nomes digitados:")
10 i = 0
11 while i < n:
12     print(i, l[i])
13     i += 1
```

Lendo uma lista de nomes

Se tivermos a quantidade n de nomes, podemos:

```
1  n = int(input())    # qtd de nomes
2  l = []              # lista vazia
3  i = 0
4  while i < n:
5      nome = input()  # lê o nome
6      l.append(nome)
7      i += 1
8
9  print("Nomes digitados:")
10 i = 0
11 while i < n:
12     print(i, l[i])
13     i += 1
```

E se quisermos conferir se uma certa aluna está matriculada?

Lendo uma lista de nomes

Se tivermos a quantidade n de nomes, podemos:

```
1  n = int(input())    # qtd de nomes
2  l = []              # lista vazia
3  i = 0
4  while i < n:
5      nome = input()  # lê o nome
6      l.append(nome)
7      i += 1
8
9  print("Nomes digitados:")
10 i = 0
11 while i < n:
12     print(i, l[i])
13     i += 1
```

E se quisermos conferir se uma certa aluna está matriculada?

```
14 nome = input()
15 if nome in l:
16     print("Está matriculada!")
17 else:
18     print("Hora de chorar pra DAC!")
```

Lendo uma lista de nomes

Se tivermos a quantidade n de nomes, podemos:

```
1  n = int(input())    # qtd de nomes
2  l = []              # lista vazia
3  i = 0
4  while i < n:
5      nome = input()  # lê o nome
6      l.append(nome)
7      i += 1
8
9  print("Nomes digitados:")
10 i = 0
11 while i < n:
12     print(i, l[i])
13     i += 1
```

E se quisermos conferir se uma certa aluna está matriculada?

```
14 nome = input()
15 if nome in l:
16     print("Está matriculada!")
17 else:
18     print("Hora de chorar pra DAC!")
```

Ou contar quantos `Enzo` existem?

Lendo uma lista de nomes

Se tivermos a quantidade n de nomes, podemos:

```
1  n = int(input())    # qtd de nomes
2  l = []              # lista vazia
3  i = 0
4  while i < n:
5      nome = input()  # lê o nome
6      l.append(nome)
7      i += 1
8
9  print("Nomes digitados:")
10 i = 0
11 while i < n:
12     print(i, l[i])
13     i += 1
```

E se quisermos conferir se uma certa aluna está matriculada?

```
14 nome = input()
15 if nome in l:
16     print("Está matriculada!")
17 else:
18     print("Hora de chorar pra DAC!")
```

Ou contar quantos `Enzo` existem?

```
19 cnt_enzo = 0
20 for nome in l: # para cada `nome` em l
21     if nome == "Enzo":
22         cnt_enzo += 1
23 print(cnt_enzo)
```

Lendo uma lista de nomes

Se tivermos a quantidade n de nomes, podemos:

```
1  n = int(input())    # qtd de nomes
2  l = []              # lista vazia
3  i = 0
4  while i < n:
5      nome = input()  # lê o nome
6      l.append(nome)
7      i += 1
8
9  print("Nomes digitados:")
10 i = 0
11 while i < n:
12     print(i, l[i])
13     i += 1
```

E se quisermos conferir se uma certa aluna está matriculada?

```
14 nome = input()
15 if nome in l:
16     print("Está matriculada!")
17 else:
18     print("Hora de chorar pra DAC!")
```

Ou contar quantos `Enzo` existem?

```
19 cnt_enzo = 0
20 for nome in l: # para cada `nome` em l
21     if nome == "Enzo":
22         cnt_enzo += 1
23 print(cnt_enzo)
```

Mais operações em listas

- índices negativos
- comprimento `len`
- intervalos (slices) `[]`
- `.insert()`
- `.index()`
- `.remove()` (não utilize dentro de laços baseados na própria lista)
- `.pop()`
- `.count()`
- `.sort()` e `sorted()`
- `min()`, `max()`, `sum()`

Tuplas

É tipo lista, só que imutáveis.

For e Range

Padrão de contagem

É muito comum contarmos de um número até outro em um laço:

```
1 i = 0
2 while i < n:
3     l.append(int(input()))
4     i += 1
```

Padrão de contagem

É muito comum contarmos de um número até outro em um laço:

```
1 i = 0
2 while i < n:
3     l.append(int(input()))
4     i += 1
```

Há uma estrutura que fica sempre aparecendo:

- Inicializa uma variável com zero (linha 1)
- Executa um laço até um valor (linha 2)
- Faz alguma coisa (linha 3)
- Soma um na variável ao final do laço (linha 4)

Padrão de contagem

É muito comum contarmos de um número até outro em um laço:

```
1 i = 0
2 while i < n:
3     l.append(int(input()))
4     i += 1
```

```
1 for i in range(n):
2     l.append(int(input()))
```

Há uma estrutura que fica sempre aparecendo:

- Inicializa uma variável com zero (linha 1)
- Executa um laço até um valor (linha 2)
- Faz alguma coisa (linha 3)
- Soma um na variável ao final do laço (linha 4)

Padrão de contagem

É muito comum contarmos de um número até outro em um laço:

```
1 i = 0
2 while i < n:
3     l.append(int(input()))
4     i += 1
```

```
1 for i in range(n):
2     l.append(int(input()))
```

O `range(n)` é como se fosse uma lista `[0, 1, ..., n - 1]`

Há uma estrutura que fica sempre aparecendo:

- Inicializa uma variável com zero (linha 1)
- Executa um laço até um valor (linha 2)
- Faz alguma coisa (linha 3)
- Soma um na variável ao final do laço (linha 4)

Padrão de contagem

É muito comum contarmos de um número até outro em um laço:

```
1 i = 0
2 while i < n:
3     l.append(int(input()))
4     i += 1
```

Há uma estrutura que fica sempre aparecendo:

- Inicializa uma variável com zero (linha 1)
- Executa um laço até um valor (linha 2)
- Faz alguma coisa (linha 3)
- Soma um na variável ao final do laço (linha 4)

```
1 for i in range(n):
2     l.append(int(input()))
```

O `range(n)` é como se fosse uma lista `[0, 1, ..., n - 1]`

- Mas não é uma lista! Seu tipo é `range`

Padrão de contagem

É muito comum contarmos de um número até outro em um laço:

```
1 i = 0
2 while i < n:
3     l.append(int(input()))
4     i += 1
```

Há uma estrutura que fica sempre aparecendo:

- Inicializa uma variável com zero (linha 1)
- Executa um laço até um valor (linha 2)
- Faz alguma coisa (linha 3)
- Soma um na variável ao final do laço (linha 4)

```
1 for i in range(n):
2     l.append(int(input()))
```

O `range(n)` é como se fosse uma lista `[0, 1, ..., n - 1]`

- Mas não é uma lista! Seu tipo é `range`
- Pode ser convertido em lista usando `list(range(n))`

O comando `for`

Duas formas (principais) de usar o `for`:

O comando `for`

Duas formas (principais) de usar o `for`:

Iterando em lista:

```
1 lista = [10, 20, 30]
2 for x in lista:
3     print(x)
```

O comando `for`

Duas formas (principais) de usar o `for`:

Iterando em lista:

```
1 lista = [10, 20, 30]
2 for x in lista:
3     print(x)
```

Aqui, `x` assume o valor de cada elemento da lista, um por vez.

```
10
20
30
```

O comando `for`

Duas formas (principais) de usar o `for`:

Iterando em lista:

```
1 lista = [10, 20, 30]
2 for x in lista:
3     print(x)
```

Gerando sequência numérica:

```
1 n = 5
2 for i in range(n):
3     print(i)
```

Aqui, `x` assume o valor de cada elemento da lista, um por vez.

```
10
20
30
```

O comando `for`

Duas formas (principais) de usar o `for`:

Iterando em lista:

```
1 lista = [10, 20, 30]
2 for x in lista:
3     print(x)
```

Aqui, `x` assume o valor de cada elemento da lista, um por vez.

```
10
20
30
```

Gerando sequência numérica:

```
1 n = 5
2 for i in range(n):
3     print(i)
```

```
0
1
2
3
4
```

Revisitando soluções

</> Code

Dado n e k, queremos imprimir os k números consecutivos começando em n.

Revisitando soluções

</> Code

Dado n e k, queremos imprimir os k números consecutivos começando em n.

```
1 n = int(input())
2 k = int(input())
3
4 for i in range(k):
5     print(n + i )
```

Revisitando soluções

</> Code

Dado a_1, n e r, imprima a soma elementos da PA correspondente.

Revisitando soluções

</> Code

Dado a_1, n e r, imprima a soma elementos da PA correspondente.

```python
n = int(input())
r = int(input())
a1 = int(input())

soma = 0
atual = a1
for i in range(n):
    soma += atual
    atual += r

print(soma)
```

Revisitando soluções

</> Code

Dado a_1, n e r, imprima a soma elementos da PA correspondente.

```
1  n = int(input())
2  r = int(input())
3  a1 = int(input())
4
5  soma = 0
6  atual = a1
7  for i in range(n):
8      soma += atual
9      atual += r
10
11 print(soma)
```

Perceba que a variável `i`, criada pelo `for` não é útil pelos valores que ela assume.

Revisitando soluções

</> Code

Dado a_1, n e r, imprima a soma elementos da PA correspondente.

```
1  n = int(input())
2  r = int(input())
3  a1 = int(input())
4
5  soma = 0
6  atual = a1
7  for i in range(n):
8      soma += atual
9      atual += r
10
11 print(soma)
```

Perceba que a variável `i`, criada pelo `for` não é útil pelos valores que ela assume.

Apenas conta a quantidade de vezes (`n`) que fazemos o laço

Revisitando soluções

</> Code

Dado a_1, n e r, imprima a soma elementos da PA correspondente.

```python
n = int(input())
r = int(input())
a1 = int(input())

soma = 0
atual = a1
for i in range(n):
    soma += atual
    atual += r

print(soma)
```

Perceba que a variável `i`, criada pelo `for` não é útil pelos valores que ela assume.

Apenas conta a quantidade de vezes (`n`) que fazemos o laço

Nos casos que a variável do for não é utilizado, é comum substituí-la por `_`.

Revisitando soluções

</> Code

Dado a_1, n e r, imprima a soma elementos da PA correspondente.

```
1  n = int(input())
2  r = int(input())
3  a1 = int(input())
4
5  soma = 0
6  atual = a1
7  for i in range(n):
8      soma += atual
9      atual += r
10
11 print(soma)
```

Perceba que a variável `i`, criada pelo `for` não é útil pelos valores que ela assume.

Apenas conta a quantidade de vezes (`n`) que fazemos o laço

Nos casos que a variável do for não é utilizado, é comum substituí-la por `_`.

```
7  for _ in range(n):
8      soma += atual
9      atual += r
```

Revisitando soluções

</> Code

Leia uma lista de n nomes, depois imprima-os.

Revisitando soluções

</> Code

Leia uma lista de n nomes, depois imprima-os.

```python
n = int(input())
nomes = []

for _ in range(n):
    nome = input()
    nomes.append(nome)

for nome in nomes:
    print(nome)
```

Range

O `range` gera uma sequência de números:

Range

O `range` gera uma sequência de números:

1. `range(fim)`: de 0 até fim-1.
   - `range(3)` → `[0, 1, 2]`

Range

O `range` gera uma sequência de números:

1. `range(fim)`: de 0 até fim-1.
   - `range(3)` → `[0, 1, 2]`
2. `range(inicio, fim)`: de inicio até fim-1.
   - `range(2, 5)` → `[2, 3, 4]`

Range

O `range` gera uma sequência de números:

1. `range(fim)`: de 0 até fim-1.
   - `range(3)` → `[0, 1, 2]`
2. `range(inicio, fim)`: de inicio até fim-1.
   - `range(2, 5)` → `[2, 3, 4]`
3. `range(inicio, fim, passo)`: pulando de passo em passo.

Range

O `range` gera uma sequência de números:

1. `range(fim)`: de 0 até fim-1.
   - `range(3)` → `[0, 1, 2]`
2. `range(inicio, fim)`: de inicio até fim-1.
   - `range(2, 5)` → `[2, 3, 4]`
3. `range(inicio, fim, passo)`: pulando de passo em passo.
   - `range(1, 10, 2)` → `[1 ,3, 5, 7, 9]`

Range

O `range` gera uma sequência de números:

1. `range(fim)`: de 0 até fim-1.
   - `range(3)` → `[0, 1, 2]`
2. `range(inicio, fim)`: de inicio até fim-1.
   - `range(2, 5)` → `[2, 3, 4]`
3. `range(inicio, fim, passo)`: pulando de passo em passo.
   - `range(1, 10, 2)` → `[1 ,3, 5, 7, 9]`
   - `range(10, 3, -2)` → `[10, 8, 6, 4]`

Revisando soluções

</> Code

Dado um inteiro p, determine se ele é primo.

Revisando soluções

</> Code

Dado um inteiro p, determine se ele é primo.

```python
p = int(input())
primo = True
raiz = int(p ** 0.5)
for k in range(2, raiz + 1):
    if p % k == 0:
        primo = False
        break  # para a execução do for
print(p >= 2 and primo)
```

Inception de Loops

</> Code

Imprimir todos os primos entre 2 e n.

```python
n = int(input())
for p in range(2, n + 1):
    primo =  # confere se é primo
    if primo:
        print(p)
```

```python
n = int(input())
for p in range(2, n + 1):
    primo = True
    for k in range(2, p):
        if k * k > p: break
        if p % k == 0:
            primo = False
            break # Sai do for interno
    if primo:
        print(p)
```