

Curso: Engenharia de Software

Disciplina: 12413 – Algoritmos de Programação, Projetos e Computação

Carga Horária: 2T e 2P

Profa. Angela Engelbrecht

Contato: angel@puc-campinas.edu.br

profa.angela@gmail.com e nas plataformas: **Teams** e **Canvas**

Material aula: CANVAS

Sejam bem-vindos!!!

Objetivos (competência derivadas, habilidades e atitudes)

1. Discernir e avaliar a história da computação
 2. Reconhecer os limites da computação, suas unidades e o nível funcional de arquitetura dos computadores
 3. Resolver problemas que tenham solução **algorítmica**
 4. **Selecionar e aplicar tecnologias** a serem utilizadas no produto de software
 5. **Solucionar problemas** simples, complexos ou críticos, aplicando em equipes os métodos de trabalho que proporcionem o desenvolvimento das relações interpessoais, a colaboração e a franca comunicação.
-
1. Avaliar opções e tomar decisões em situações de orientação, direcionamento, apreciação, resolução de conflitos e otimização.
 2. Exercer a liderança requerida pelos diversos papéis e atividades do Engenheiro de Software.
 3. Aplicar **raciocínio lógico e abstrato**
 4. **Codificar, avaliar e corrigir** programas.

Objeto de Conhecimento (Conteúdos)

1. Evolução dos computadores em perspectiva histórica. Origens, fundamentos e limites da computação.
2. Os principais componentes de um computador. **Unidades** utilizadas para expressar **quantidades** de **bytes** e realizar operações aritméticas com quantidades expressas em diferentes bases numéricas
1. Algoritmos. Codificação e depuração de programas. Lógica matemática e computacional aplicada. Tamanhos e codificação dos tipos de dados na memória (ASCII e UNICODE).
2. Estrutura de dados
3. Comandos de Entrada e Saída
4. Seleção e Laços
5. Modulação
6. Tecnologia e ferramental do profissional de TI

Conteúdo Programático



- Introdução a Computação e a Programação
- Ambiente. Detalhes importantes da sintaxe;
- Tipos de Dados primitivos, constantes e variáveis;
- Comandos de entrada e saída de dados;
- Comandos condicionais;
- Comandos Repetitivos;
- Strings;
- Listas, tuplas, set e dicionários;
- Funções e procedimentos
- Bibliotecas

Introdução - conceitos básicos computação - programação

Plano da Disciplina - Bibliografia -
Critério de Avaliação

introdução

exemplos de
ambientes/IDLE

características da
linguagem python

constantes - variáveis

Tipos de dados primitivos

tipos de dados: inteiros, reais e literais

tipo
complexo

tipo lógico - álgebra
booleana

Comandos básicos de programação

range()

input()

while

for

print()

if - else elif

formatos - f-string

string

Estrutura de Dados

listas

tuplas

dicionários

listas como matriz

funções - módulos

funções recursivas

comprehension

validações

exceções

Estrutura de
programa

Bibliografia

1. Downey, Allen B. ; “Pense em Python: Pense Como um Cientista da Computação”; Novatec Editora.
2. Halterman, Richard L.; Fundamentals of Python Programming. 2018.
Ed by Southern Adventist University in Collegedale –
<https://www.dbooks.org/fundamentals-of-python-programming-1200/pdf/>. Acessado em: 21/02/2021

Bibliografia Complementar

1. Forbellone, A.; Eberspacher, H. - "Lógica de Programação", Editora Makron Books, 2000.
2. ASCENDIO, A.; Campos, E. - "Fundamentos da Programação de Computadores", Prentice Hall, 2002.

TEÓRICA: Duas Avaliações – A1 e A2 (valor máximo: 10,0 cada uma)

PLANO DE ENSINO

$$\text{Média Teórica (MT)} = 0,4 * A1 + 0,6 * A2$$

PRÁTICA: Diversas Atividades de Programação -> **Média Prática (MP):** valor máximo 10 pontos

MÉDIA FINAL:

Se **MT** e **MP** forem maiores ou iguais à 5,0 então a **Média Final** = $0,6 * MT + 0,3 * MP + 0,1 * PI(*)$

Se **MT** e/ou **MP** for(em) menor(es) do que 5,0 então a **Média Final** = $\min (MT, MP)$

(*) Projeto Integrador (PI): para os alunos que não fazem Projeto Integrador, haverá uma Atividade Prática, para a composição do ponto - 10%

SUBSTITUTIVA: Os alunos que não obtiverem média superior ou igual à 5,0 na **MT** poderão fazer uma terceira atividade teórica **A3** que poderá substituir **uma das notas (A1 ou A2)**. A atividade **A3** incluirá **toda a matéria**.

Ausências e **provas** e/ou **atividades práticas** perdidas -> há regras a serem seguidas - **informe-se** na área logada de como proceder

IMPORTANTE!

A constatação de **plágio**, seja entre colegas ou de fontes externas, tais como internet, em qualquer atividade avaliada, implicará na atribuição da nota mínima (zero) para a atividade, para todos os envolvidos. Entende-se por envolvidos, tanto os alunos que fizeram o plágio, como os que permitiram que ele fosse feito. Dependendo da gravidade do incidente, a Direção e o Conselho de Faculdade à qual se vincula o aluno serão acionados para a adoção das sanções disciplinares cabíveis.

Na dúvida do que se considera plágio, o professor deve ser consultado antes de qualquer entrega.

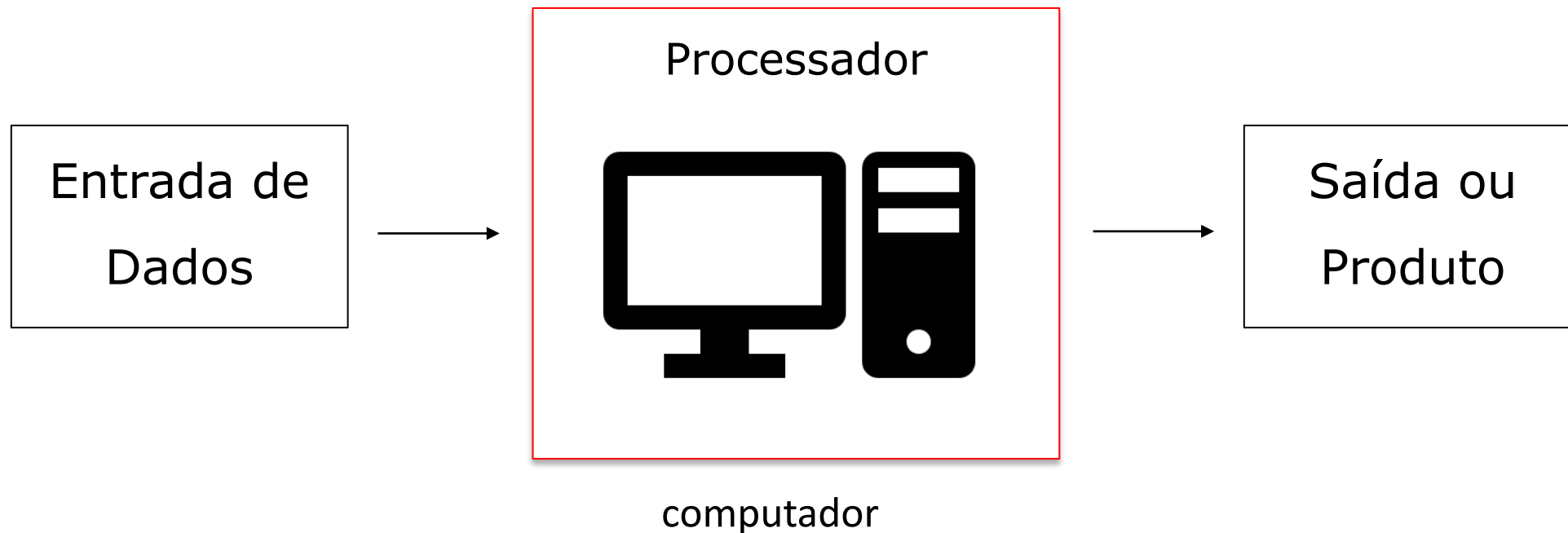
UM POUCO DE HISTÓRIA

A **busca pela mecanização das tarefas humanas** em qualquer área de atividade, seja ela profissional ou não, vem de **longa data**. Podemos observar isso, de modo geral, no uso de ferramentas e máquinas nas indústrias, no campo, nas residências.

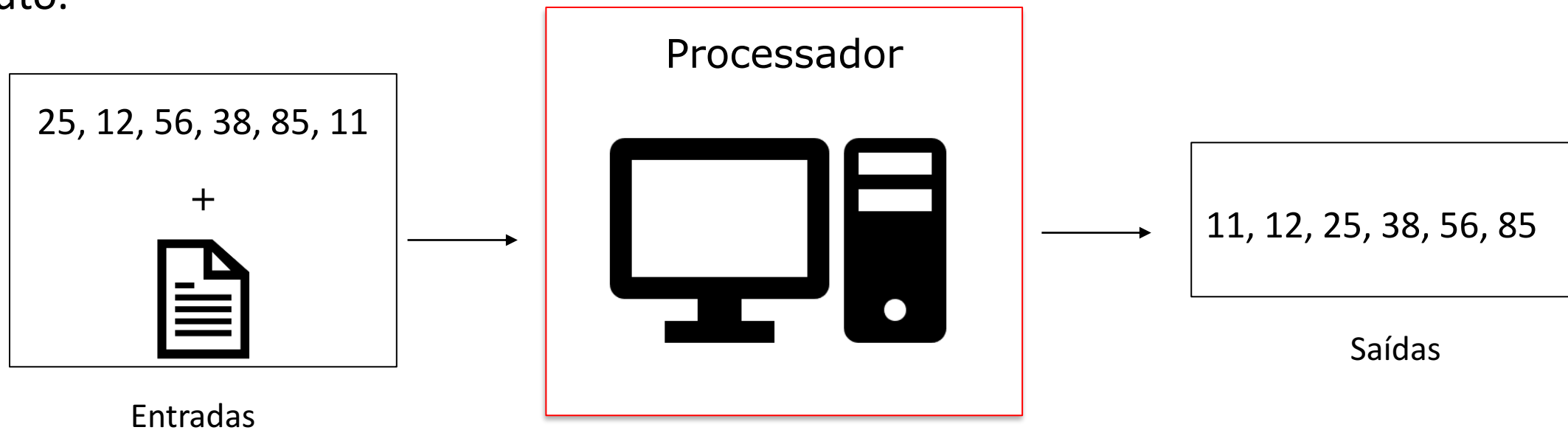
No caso do **tratamento da informação** buscou-se formas de **como compreender, avaliar e comunicar as ideias**, que ao longo do tempo foram denominadas de hardware (ferramentas) e software (ideias).

No início de 1960 os **circuitos integrados** marcaram a evolução da tecnologia da informação, permitindo os **avanços** na direção de produzir processadores cada vez **menores** e ao mesmo tempo mais **rápidos** e com maior poder de **armazenamento**.

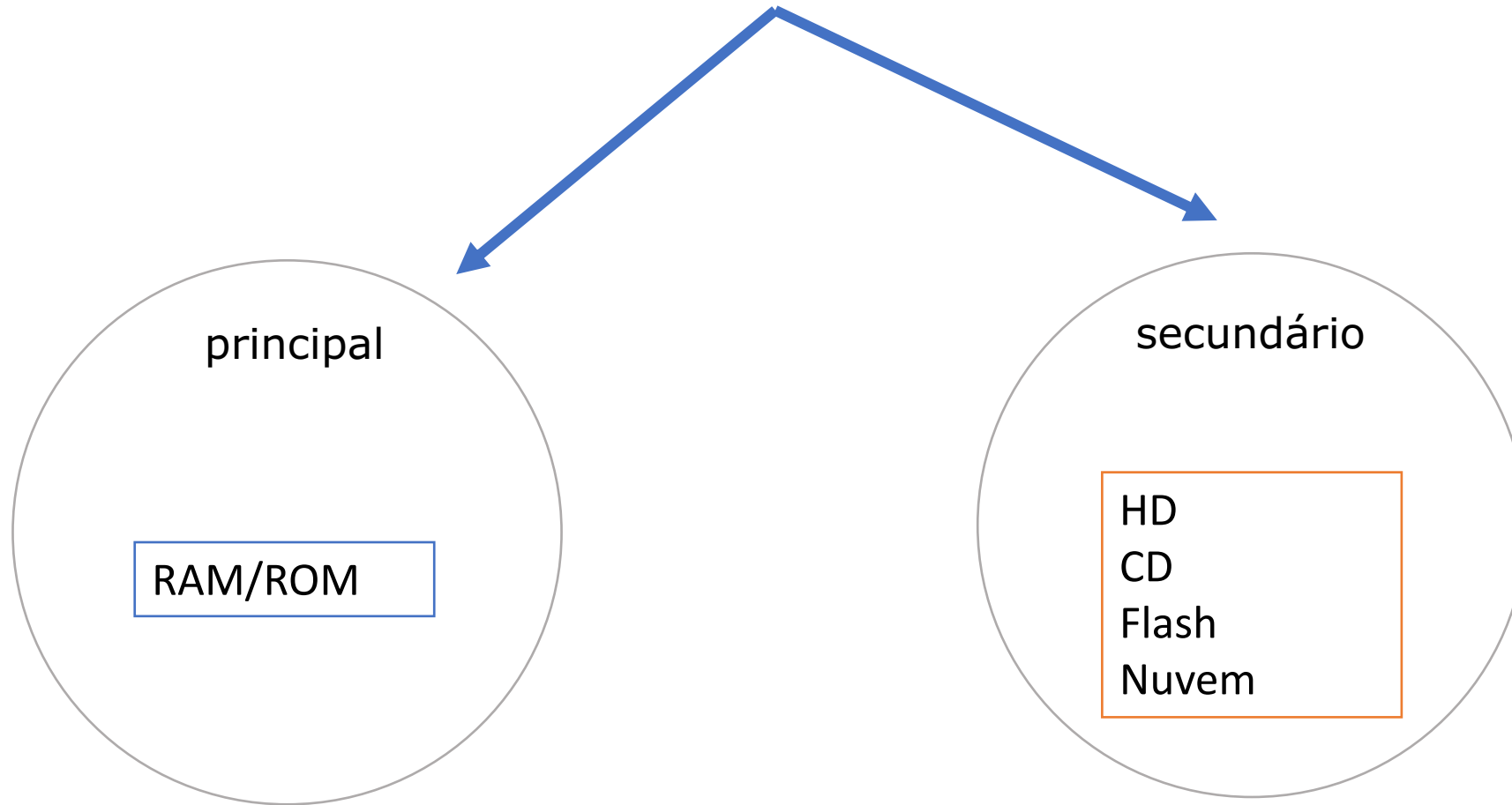
O processamento da informação pode ser representado basicamente por:



Exemplo: colocar em ordem crescente o seguinte conjunto de números inteiros: 25, 12, 56, 38, 85 e 11. O conjunto **de números** e as instruções de como organizá-los em ordem crescente são os dados de entrada. As **instruções devem ser dadas em alguma linguagem de programação**. O conjunto ordenado representa a saída gerada como produto.

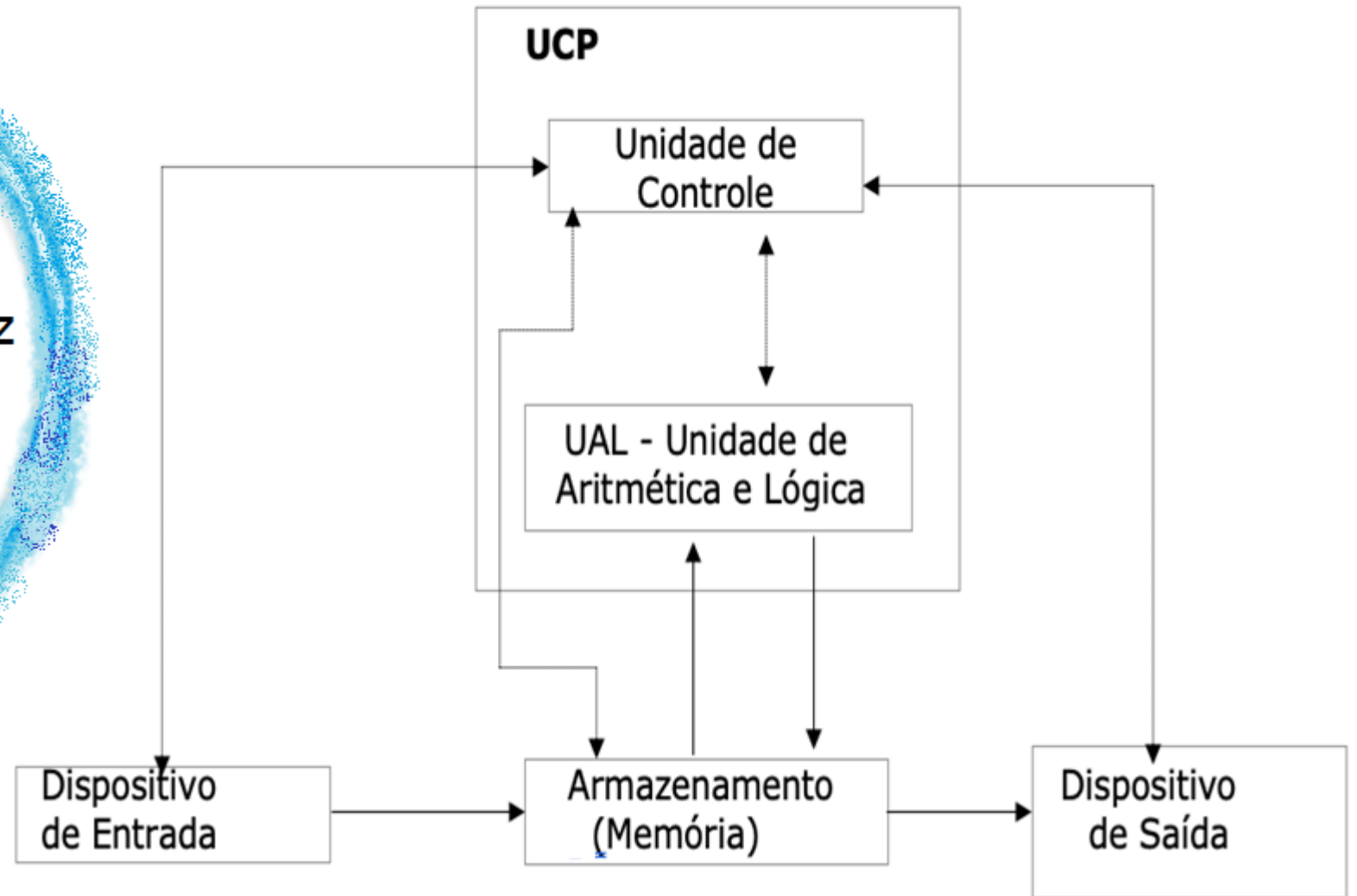


Após a captura dos dados eles devem ser retidos – **armazenados** - para serem utilizados no processamento.



UCP – Unidade Central de Processamento
(CPU – *Central Processing Unit*)

Uma unidade central de processamento (CPU) faz o controle de todas as operações.



- **Programas** e **dados** devem estar na memória principal durante a execução do programa.
- A memória é dividida em unidades pequenas e de mesmo tamanho, chamadas **palavras**.
- Cada palavra possui um único endereço e é capaz de armazenar **uma** informação.

Exemplo: um computador com palavra de 16 bits

Endereço de Byte	Célula – 8bits (1 byte)								
end 1	0	1	1	0	0	0	0	1	Palavra 0
end 2	1	0	0	0	0	0	1	0	
end 3	0	0	0	0	0	1	1	1	Palavra 1
end 4	1	0	1	1	0	0	0	1	
end 5									
end 6									

Toda informação armazenada em um processador eletrônico é transformada em 0's e 1's, a unidade básica da informação – bit.

Representa dois estados possíveis: 0 ou 1, falso ou verdadeiro, ligado ou desligado.

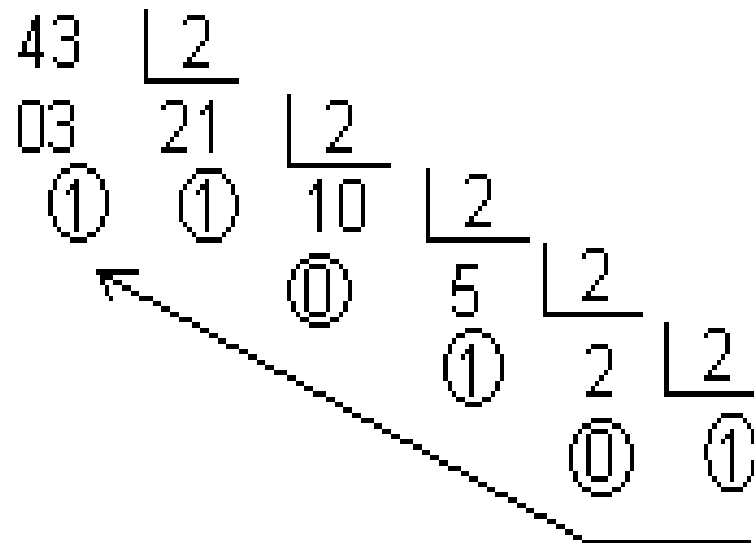
A compreensão dessa codificação da informação em 0's e 1's envolve entender, a princípio, duas coisas:

- ✓ o sistema de numeração binário – base 2
- ✓ a formação das palavras **utilizando-se o código ASCII**.

O sistema é chamado de **binário** pois trabalha com **dois símbolos** distintos: 0 e 1, assim como nosso sistema é chamado de **decimal** pois possui **10 símbolos**: 1, 2, 3, 4, 5, 6, 7, 8, 9 e 0.

A transformação de um **número decimal** para a **base binária** é feita por meio de **divisões sucessivas** por **dois**, tomando-se o **resto** das divisões ocorridas, na ordem inversa. Isto é, da última divisão ocorrida para a primeira.

Exemplo: a transformação do número 43 decimal para binário



43 em decimal = 101011 em binário

Para transformar um número na base 2 (binária) utiliza-se potências de 2.

Exemplo 1.3: 101011, para a base decimal,

1	0	1	0	1	1
					$1 \times 2^0 = 1$
					$1 \times 2^1 = 2$
					$0 \times 2^2 = 0$
					$1 \times 2^3 = 8$
					$0 \times 2^4 = 0$
					$1 \times 2^5 = 32$
					<hr/>
					43

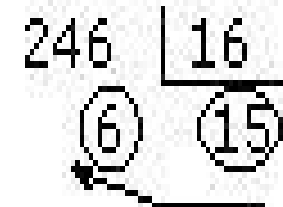
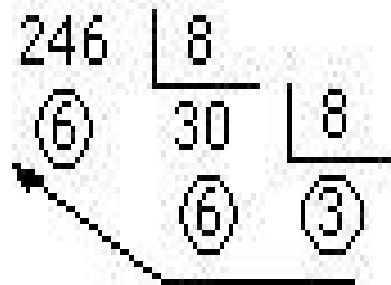
101011 em binário = 43 em decimal

Outros sistemas de numeração:

- octal (base 8)
- hexadecimal (base 16)

Da mesma forma passa-se da base decimal para a octal e hexadecimal, efetuando-se divisões sucessivas por oito e por dezesseis, respectivamente.

Exemplo: passar o número 246 para as bases 8 (octal) e 16 (hexadecimal):



decimal	octal	hexadecimal
246	366	F6

Para um conjunto de 4 números:

Decimal	Binária	Octal	Hexadecimal
0	0000	0	0
1	0001	1	1
2	0010	2	2
3	0011	3	3
4	0100	4	4
5	0101	5	5
6	0110	6	6
7	0111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

As palavras – os caracteres – são codificados de acordo com a tabela ASCII

dec.	hex.	octal	ASCII	mnem.	dec.	hex.	octal	ASCII	dec.	hex.	octal	ASCII	dec.	hex.	octal	ASCII
0	00	000	^@	NUL	32	20	040		64	40	100	@	96	60	140	.
1	01	001	^A	SOH	33	21	041	!	65	41	101	A	97	61	141	a
2	02	002	^B	STX	34	22	042	"	66	42	102	B	98	62	142	b
3	03	003	^C	ETX	35	23	043	#	67	43	103	C	99	63	143	c
4	04	004	^D	EOT	36	24	044	\$	68	44	104	D	100	64	144	d
5	05	005	^E	ENQ	37	25	045	%	69	45	105	E	101	65	145	e
6	06	006	^F	ACK	38	26	046	&	70	46	106	F	102	66	146	f
7	07	007	^G	BELL	39	27	047	'	71	47	107	G	103	67	147	g
8	08	010	^H	BS	40	28	050	(72	48	110	H	104	68	150	h
9	09	011	^I	HTAB	41	29	051)	73	49	111	I	105	69	151	i
10	0A	012	^J	LF	42	2A	052	*	74	4A	112	J	106	6A	152	j
11	0B	013	^K	VTAB	43	2B	053	+	75	4B	113	K	107	6B	153	k
12	0C	014	^L	FF	44	2C	054	,	76	4C	114	L	108	6C	154	l
13	0D	015	^M	CR	45	2D	055	-	77	4D	115	M	109	6D	155	m
14	0E	016	^N	SO	46	2E	056	.	78	4E	116	N	110	6E	156	n
15	0F	017	^O	SI	47	2F	057	/	79	4F	117	O	111	6F	157	o
16	10	020	^P	DLE	48	30	060	0	80	50	120	P	112	70	160	p
17	11	021	^Q	DC1	49	31	061	1	81	51	121	Q	113	71	161	q
18	12	022	^R	DC2	50	32	062	2	82	52	122	R	114	72	162	r
19	13	023	^S	DC3	51	33	063	3	83	53	123	S	115	73	163	s
20	14	024	^T	DC4	52	34	064	4	84	54	124	T	116	74	164	t
21	15	025	^U	NACK	53	35	065	5	85	55	125	U	117	75	165	u
22	16	026	^V	SYN	54	36	066	6	86	56	126	V	118	76	166	v
23	17	027	^W	ETB	55	37	067	7	87	57	127	W	119	77	167	w
24	18	030	^X	CAN	56	38	070	8	88	58	130	X	120	78	170	x
25	19	031	^Y	EN	57	39	071	9	89	59	131	Y	121	79	171	y
26	1A	032	^Z	SUB	58	3A	072	:	90	5A	132	Z	122	7A	172	z
27	1B	033	^[ESC	59	3B	073	;	91	5B	133	[123	7B	173	{
28	1C	034	^\ ^	FS	60	3C	074	<	92	5C	134	\	124	7C	174	
29	1D	035	^] ^	GS	61	3D	075	=	93	5D	135]	125	7D	175	}
30	1E	036	^^ ^	RS	62	3E	076	>	94	5E	136	^	126	7E	176	~
31	1F	037	^_ ^	US	63	3F	077	?	95	5F	137	_	127	7F	177	DEL

Na memória: frase “Bom Dia”

Código ASCII em hexadecimal

B = 42

o = 6F

m = 6D

sp = 20


D = 44

i = 69

a = 61

sp: espaço

Cada
caractere
ocupa
apenas um
byte: 8 bits

	4	2
		
B	0100 0010	
o	0110 1111	
m	0110 1101	
sp	0010 0000	
D	0100 0100	
i	0110 1001	
a	0110 0001	

Python, os programas são **interpretados** utilizando-se o conjunto de caracteres chamados – **UTF – 8**.

UTF: *Unicode Transformation Format* – 8-bit

- codificação criada por **Ken Thompson** e **Rob Pike**;
- possuem comprimento variável – de 1 a 4 bytes - para conseguir representar todos os códigos do Unicode.

Unicode: padrão de codificação que abrange todos os caracteres de todas as línguas escritas no planeta, permite que possam representá-los em programas de computadores.

Os **128** primeiros caracteres do **ASCII** estão representados no **Unicode**, preservando a codificação original.

Unicode - padrão segue o formato: **U**, símbolo **'+'** e quatro dígitos em **hexadecimal**.

Exemplo, vamos usar o exemplo visto anteriormente:

sp: espaço

caracteres	hexa - ASCII	Unicode
B	42	U+0042
o	6F	U+006F
m	6D	U+006D
sp	20	U+0020
D	44	U+0044
i	69	U+0069
a	61	U+0061

- uma das principais fontes de consulta sobre python é: python.org/
- caso queira, para a instalação do python: <https://python.org.br/instalacao-windows/> com IDLE
- outras IDE – *Integrated Development Environment*:
 - Spyder
 - Pycharm
- online:
 - <https://colab.research.google.com/>
 - <https://www.onlinegdb.com/>
 - <https://replit.com/>
- visualizar as variáveis e impressões na execução do programa:
<http://pythontutor.com/>

Python é uma linguagem de programação de **alto nível** e é **interpretada**.

Uma linguagem **interpretada** difere de uma **compilada**. A interpretada é executada sem passar por uma nova codificação, o que acontece com as compiladas. Isto é, o código fonte, das interpretadas, passa apenas pelo interpretador que já executa o comando.

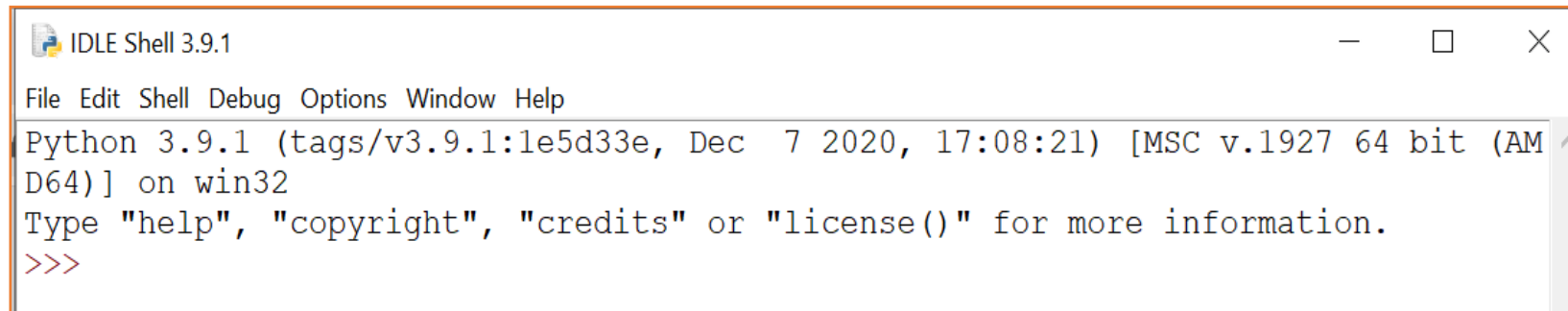
Nas **compiladas**, o código fonte passa pelo compilador que gera o **código objeto** e somente depois, esse código objeto é **executado**.

Podemos usar o interpretador de **Python** de duas maneiras:

- no modo linha de comando ou **interativa** - *shell mode* ;
- no modo de **programa** – *script – program mode*.

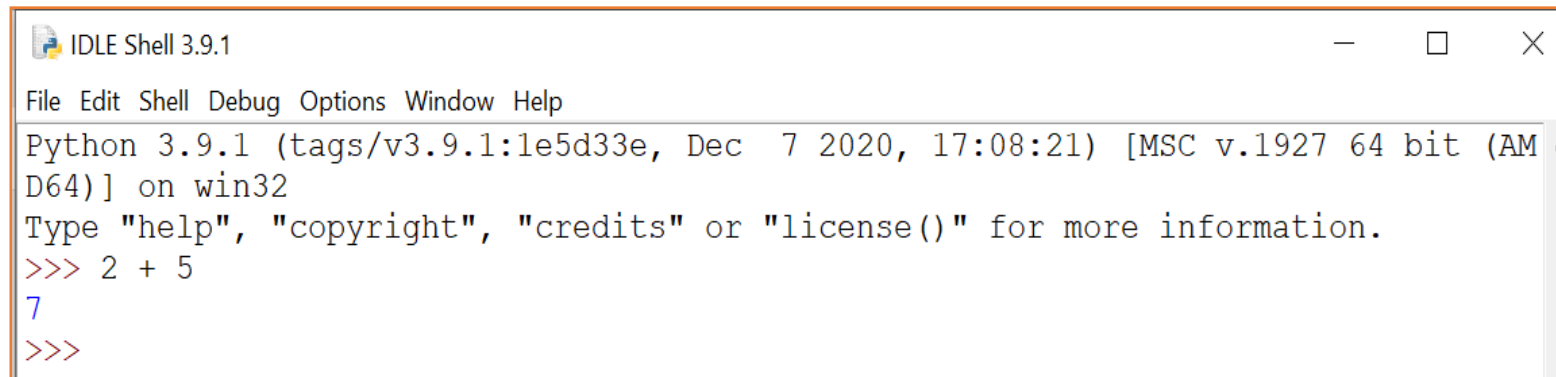
no modo linha de comando ou **interativa** - *shell mode* ;

Os comandos são lidos de um **console**, interpretados imediatamente e produzindo uma resposta ao comando digitado. O **prompt** primário usado é representado por três sinais de maior (>>>):



```
IDLE Shell 3.9.1
File Edit Shell Debug Options Window Help
Python 3.9.1 (tags/v3.9.1:1e5d33e, Dec 7 2020, 17:08:21) [MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
```

Após a digitação do comando seguido por <enter>, o comando é interpretado e, se não há erros, o resultado é mostrado:



```
IDLE Shell 3.9.1
File Edit Shell Debug Options Window Help
Python 3.9.1 (tags/v3.9.1:1e5d33e, Dec 7 2020, 17:08:21) [MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> 2 + 5
7
>>>
```

O uso do interpretador de forma **interativa** permite que se faça rapidamente testes dos comandos da linguagem. *Sendo uma ferramenta útil para aprender a linguagem.*

Na dúvida, sobre a **sintaxe** de um comando, use a função ***help()***, colocando dentro dos parênteses, o nome do comando/função, que deseja ter mais informações. Por exemplo, saber mais sobre ***print()***:

```
>>> help(print)
Help on built-in function print in module builtins:

print(...)
    print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

    Prints the values to a stream, or to sys.stdout by default.
    Optional keyword arguments:
    file: a file-like object (stream); defaults to the current sys.stdout.
    sep:   string inserted between values, default a space.
    end:   string appended after the last value, default a newline.
    flush: whether to forcibly flush the stream.

>>>
```

É mostrada a sintaxe da função ***print()*** e uma explicação sobre seus argumentos.

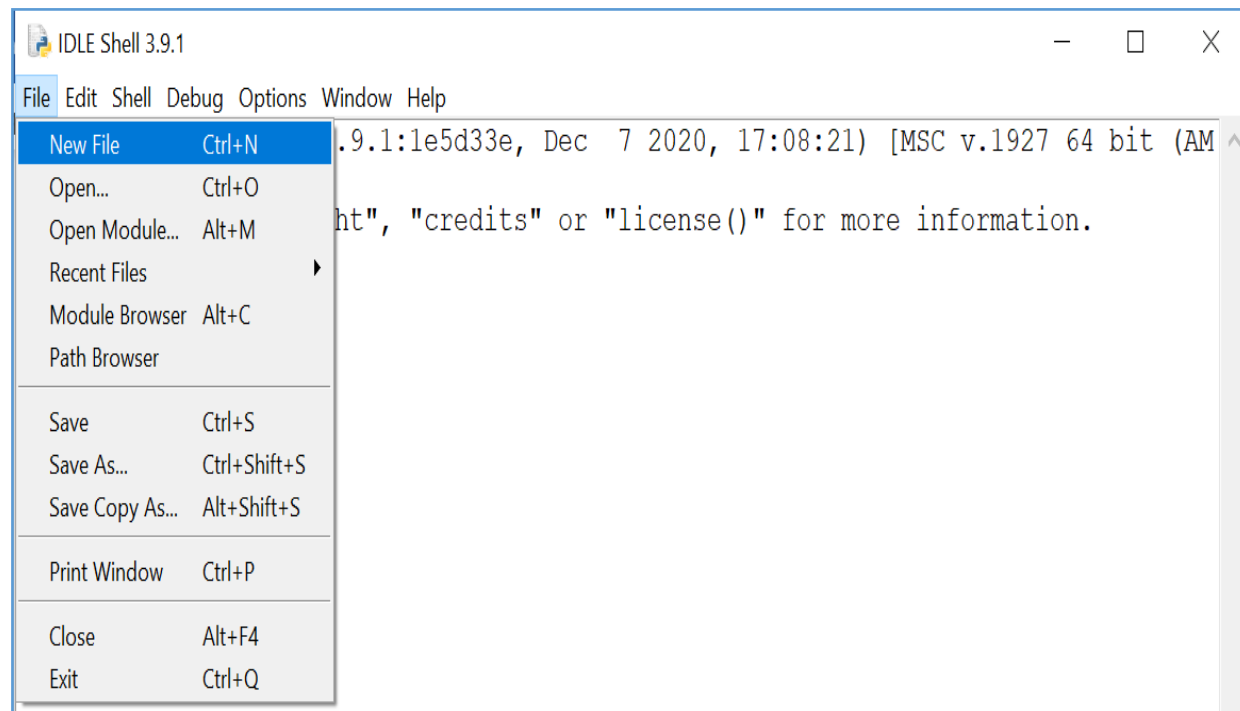
Posteriormente veremos o uso dessa função e os detalhes dos seus argumentos.

no modo de **programa** – *script* – *program mode*.

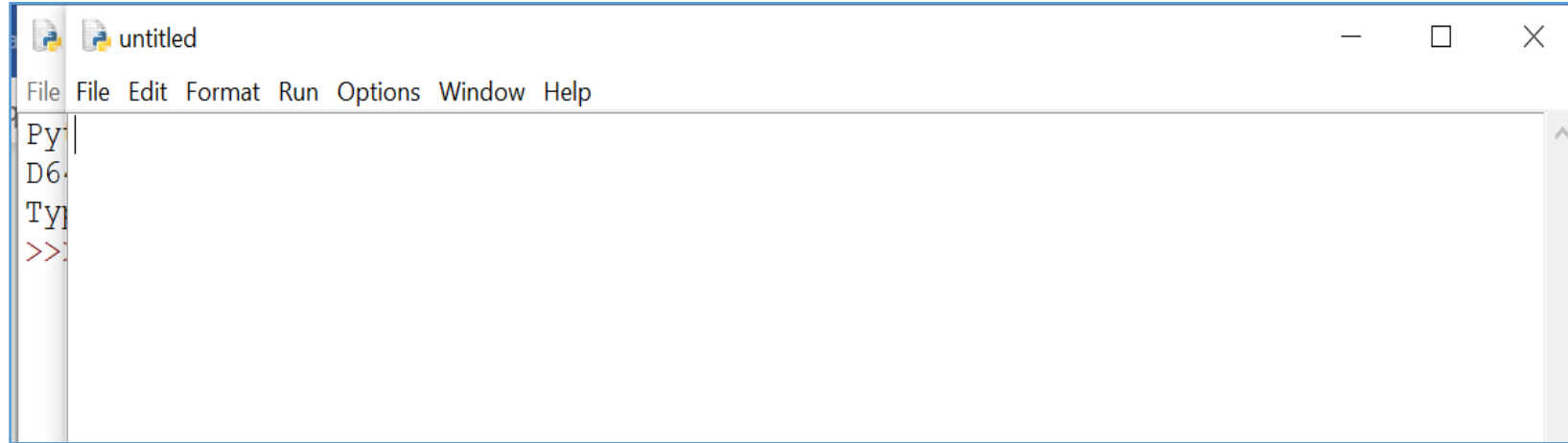
No **modo de programa**, é criado um arquivo com o programa desenvolvido e que será executado como um todo. Cria um arquivo .py

Aqui, **somente a título de exemplo**, se usarmos a IDE instalada junto com python:

Em **File**, escolha **New File**, como mostra a figura abaixo:

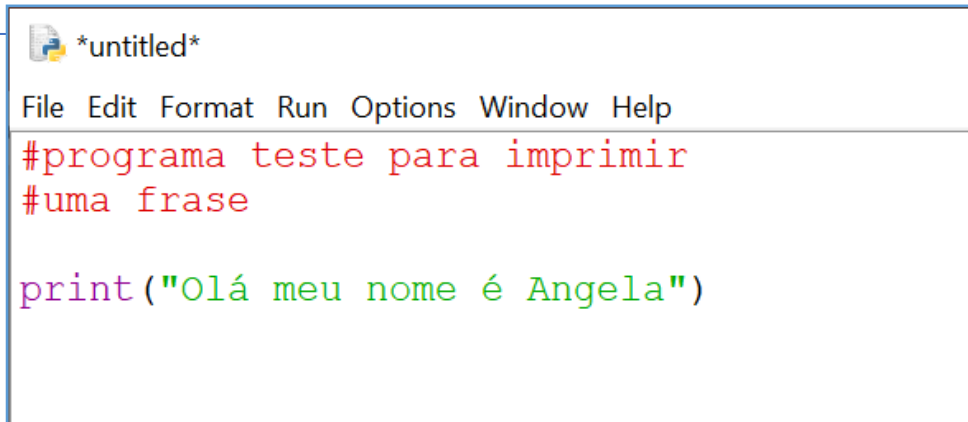


É criada uma janela, como mostra a figura abaixo:



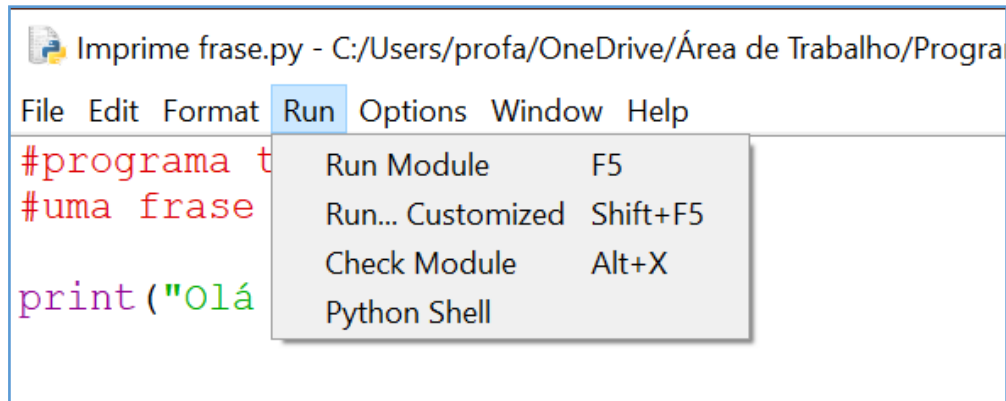
Observe que não há o prompt (**>>>**) do modo *shell*, pois as linhas de comandos, que vamos escrever, não serão interpretadas à medida que a tecla *<enter>* é usada. O conjunto todo será executado somente quando for solicitada sua **execução**.

Exemplo para imprimir uma frase: →



```
*untitled*  
File Edit Format Run Options Window Help  
#programa teste para imprimir  
#uma frase  
  
print("Olá meu nome é Angela")
```

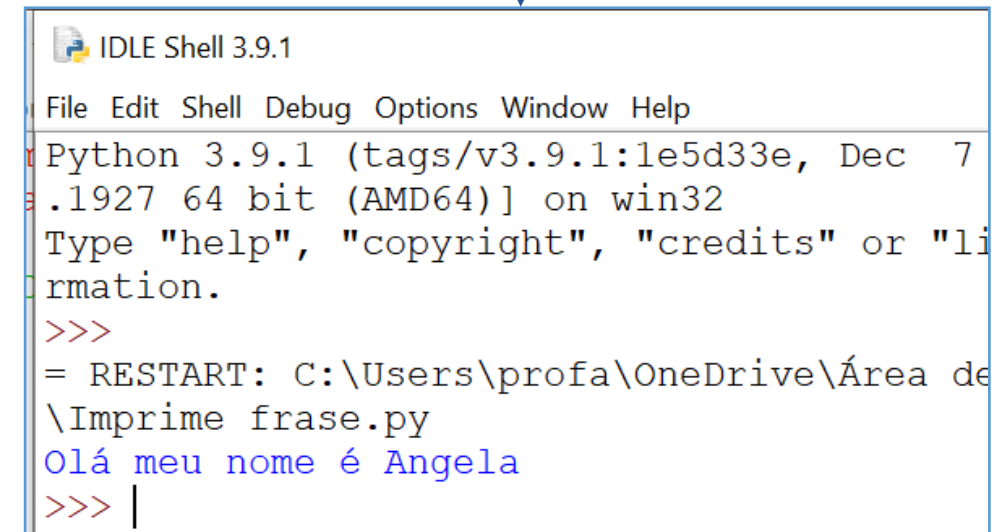
Para executar é necessário salvá-lo e depois executar (Run ou F5)



```
Imprime frase.py - C:/Users/profa/OneDrive/Área de Trabalho/Progra  
File Edit Format Run Options Window Help  
#programa t  
#uma frase  
  
print("Olá
```

Run Module F5
Run... Customized Shift+F5
Check Module Alt+X
Python Shell

O resultado é mostrado no console



```
IDLE Shell 3.9.1  
File Edit Shell Debug Options Window Help  
Python 3.9.1 (tags/v3.9.1:1e5d33e, Dec 7  
.1927 64 bit (AMD64)] on win32  
Type "help", "copyright", "credits" or "li  
rmation.  
>>>  
= RESTART: C:\Users\profa\OneDrive\Área de  
\Imprime frase.py  
Olá meu nome é Angela  
>>> |
```

Inicialmente, a fim de conhecer um pouco o modo **interativo** – vamos observar seu uso com alguns exemplos - **Conhecendo a linguagem usando exemplos**

O **modo interativo**, pode ser usado como uma **calculadora**. Digite uma operação aritmética simples, como no exemplo:

Ex01:

```
>>> 2 + 5 #<enter>  
7
```

observe que o símbolo **#** é usado, no início, para **comentários**. Para mais de uma linha de comentário, o símbolo deve ser repetido.

Ex02: uso de parênteses para mudar a hierarquia das operações

```
>>> 2 + 5 * 3 #<enter>
17
>>> (2 + 5) * 3 #<enter>
21
>>> |
```

Ex03: operações aritméticas: adição, subtração, multiplicação e divisão

```
>>> #operações aritméticas básicas:
>>> #adição: +
>>> #subtração: -
>>> #multiplicação: *
>>> #divisão: /
>>> 2 + 5
7
>>> 3 - 6
-3
>>> 2 * 3
6
>>> 6 / 2
3.0
>>> |
```

observe:

1. ***** para a multiplicação e **/** para a divisão;
2. que o **resultado** da divisão, embora não tenha **casas decimais**, foi apresentado como se fosse um número real: **3.0**

Por que?

Isso acontece por que o símbolo: **/**, que representa a operação de divisão, embora o dividendo e divisor sejam inteiros e múltiplos, produz um **resultado real**.

Para produzir um **resultado inteiro** – sem casas decimais, mesmo que eles não sejam múltiplos, é preciso usar duas barras: **//**

Ex04:

```
>>> #divisão com resultado real:/
>>> 6/2
3.0
>>> #divisão com resultado inteiro: //
>>> 6 // 2
3
>>> #exemplo com não múltiplos
>>> 11/2
5.5
>>> 11//2
5
>>> |
```

observe que o resultado de **11 dividido por 2**, usando duas barras **//**, resulta no **inteiro: 5** → é representado como **inteiro**, **sem casas decimais**. A parte decimal é descartada.

Resto da divisão: para obter o resto da divisão é usado o símbolo: **'%'**

```
>>> #resto da divisão: %  
>>> 6 % 2  
0  
>>> 11 % 2  
1  
>>>
```

Pode ser usado com **reais** – o resto da divisão, cujo **quociente é inteiro**. A operação não tem continuidade para quociente com casas decimais:

```
>>> 3.5 % 2  
1.5  
>>> 3.5 // 2  
1.0  
>>>
```

Potência: para o cálculo da potência são utilizados dois asteriscos: **

Exemplo: 2^3

```
>>> #operador para potência: **
>>> 2 ** 3
8
>>> |
```

O **expoente** pode ser um número **real** e **negativo**.

Exemplos:

```
>>> #outros exemplos
>>> #Raiz quadrada de 2 -> 2 elevado à 0.5
>>> 2 ** 0.5
1.4142135623730951
>>> #Fração: 1 / raiz(2) -> 2 elevado à -0.5
>>> 2 ** -0.5
0.7071067811865476
>>> |
```

Obs.: a **formatação** do resultado apresentado, por exemplo, duas casas decimais, será abordado posteriormente.

Elemento de destaque sobre a programação em python:

- os blocos de comandos são caracterizados por indentação – o conjunto de comando referentes àquele bloco, segue a mesma indentação – para cada nível são usados **4 espaços** – não recomendado uso da tecla *tab*

Outras características, sobre a estrutura de um programa em python – por exemplo: espaços entre variáveis, podem ser encontradas em:

- PEP-8 <https://peps.python.org/pep-0008/>

Zen of Python – 19 regras para um bom projeto em python – PEP-20
<https://peps.python.org/pep-0020/>

easter egg → `>>> import this`

NÚMEROS e OPERADORES em Python

Os **números** – são tipos de dados – seguem as regras da matemática, em relação às definições, operações e hierarquias e são definidos como:

- *int*- para os inteiros;
- *float*- para os reais;
- *complex*- para os complexos;
- *lógico* – valores: verdadeiro e falso.

INTEIROS

tipo: *int* de inteiros (*integer*)

Representação: zero, negativos e positivos;

Operadores básicos: adição (+), subtração (-), multiplicação (*), divisão resultado inteiro (//),
resto da divisão (%), potência (**);

Regras: as mesmas da aritmética;

Tamanho: em *python*: **não há limite de tamanho.**

Isso significa que se pode trabalhar com números inteiros, **com valores grandes**, e não teremos problemas de **representação** na memória.

Em outras linguagens, compiladas, embora também trabalhe com números grandes, há limites, em função do espaço de memória reservado para cada número.

INTEIROS (continuação)

Ex.:

```
>>>
>>> 12345678901234567890123456 * 100
1234567890123456789012345600
>>> 12345678901234567890123456 * 1000
12345678901234567890123456000
>>> 12345678901234567890123456 * 10000
123456789012345678901234560000
>>> 12345678901234567890123456 * 100000
1234567890123456789012345600000
>>> 12345678901234567890123456 * 200000
2469135780246913578024691200000
>>> 12345678901234567890123456 * 300000
3703703670370370367037036800000
>>> |
```


REAIS

Tipo: *float* de ponto flutuante (*floating point*)

Representação: zero, negativos e positivos com casas decimais;

Operações: adição (+), subtração (-), multiplicação (*), divisão resultado real (/), resto da divisão (%),
potência (**);

Regras: as mesmas da aritmética;

Notação científica: usada a letra **e** ou **E**, para representar a potência de **10**.

```
>>> 3.5e5  
350000.0  
>>> 7.2E3  
7200.0  
>>>
```

```
>>> 3.5e2 + 1.75E2  
525.0  
_
```

COMPLEXOS

Representação: compostos por duas partes, uma formada por um número real e outra a parte imaginária;

Formatos:

- $x + yj$, onde x e y são reais e $j = \sqrt{-1}$
- `complex` (x, y)

Operadores básicos: adição (+), subtração (-), multiplicação (*), divisão (/);

Regras: as mesmas da álgebra;


```
>>> #números complexos
>>> 2 + 3j
(2+3j)
>>> complex(3,5)
(3+5j)
```

Exemplos com as operações aritméticas:

```
>>> #operações com complexos
>>> #adição
>>> (2 + 3j) + (2.5 + 5j)
(4.5+8j)
>>> #subtração
>>> (2 + 3j) - (5 + 5j)
(-3-2j)
>>> #multiplicação
>>> (2 - 3j) * (3 + 5j)
(21+1j)
>>> #divisão
>>> (2 - 3j) / (4 + 2j)
(0.1-0.8j)
>>> |
```

Tipo lógico – boolean

Valores: *True* (verdadeiro) e *False* (falso)
False: representado pelo 0 (zero);
True: **qualquer** valor diferente de 0 (zero);

Operadores lógicos: *and* (e), *or* (ou) e *not* (não) 

Hierarquia: *not*(não), *and* (e), *or*(ou)

A	B	A and B
V	V	V
V	F	F
F	V	F
F	F	F

A	B	A or B
V	V	V
V	F	V
F	V	V
F	F	F

A	not A
V	F
F	V

V: verdade e F: falso

Regra dos operações como em C: *curto-circuito*

Operadores relacionais: ==, !=, >, >=, <, <= - usados para construir expressões relacionais cujo resultado será: *True* ou *False*

Operador	Significado	Uso	Resultado
==	igual	A == B	<i>True</i> se A for igual à B e <i>False</i> caso contrário
!=	não igual (diferente)	A != B	<i>True</i> se A for diferente de B e <i>False</i> caso contrário
>	maior	A > B	<i>True</i> se A for maior do que B e <i>False</i> caso contrário
>=	maior ou igual	A >= B	<i>True</i> se A for maior ou igual à B e <i>False</i> caso contrário
<	menor	A < B	<i>True</i> se A for menor do que B e <i>False</i> caso contrário
<=	menor ou igual	A <= B	<i>True</i> se A for menor ou igual à B e <i>False</i> caso contrário

STRING – um caractere ou uma cadeia de caracteres

tipo: *str* de *string*

Representação: letra – maiúsculas e minúsculas, dígitos, símbolos;

Operações: são várias operações para trabalhar com strings e serão abordadas posteriormente.

Algumas delas:

concatenação (+): unir strings formando outras strings -> **s1** + **s2**

duplicação (*): **s** * **n** -> duplicar **s**, **n** vezes , na própria **s**

len (s) - retorna o comprimento da string **s**

s1 *in* **s2** : retorna **True** se **s1** está contida em **s2** e **False** caso contrário

s1 *not* **in** **s2**: retorna **True** se **s1** não estiver contida em **s2** e **False** caso contrário

```
File Edit Format Run Options Window Help
```

```
# representação de string
s1 = "a"  #aspas duplas

s2 = 'a'  #aspas simples

s3 = "Ola! string 'com' aspas duplas. "

s4 = 'Ola! string "com" aspas simples.'

s5 = """Olá! 'Pode' conter aspas na string! "Bom dia."
      Enquanto eu não fechar com TRES aspas duplas,
      posso continuar meu texto.
      """

s6 = '''OI! 'Vale' também para aspas simples ou
      apóstrofes.
      '''

#posso usar também como comentário
''' Comentário com mais de uma linha
termina quando forem fechadas as aspas.'''

print("s1:", s1)
print("s2:", s2)
print("s3:", s3)
print("s4:", s4)
print("s5:", s5)
print("s6:", s6)
```

Representação de
valores de uma
string

```
== RESTART: C:/Users/profa/OneDrive/Área de Trabalho/
s1: a
s2: a
s3: Ola! string 'com' aspas duplas.
s4: Ola! string "com" aspas simples.
s5: Olá! 'Pode' conter aspas na string! "Bom dia."
      Enquanto eu não fechar com TRES aspas duplas,
      posso continuar meu texto.

s6: OI! 'Vale' também para aspas simples ou
      apóstrofes.
```

File Edit Format Run Options Window Help

```
# tipo string
pre_nome = "Angela"
sobrenome = "Engelbrecht"

nome = pre_nome + " " + sobrenome

print("Nome = ", nome)

print("="*20)
print("Comp pre_nome = ", len(pre_nome))
print("Comp sobrenome = ", len(sobrenome))
print("Comp nome = ", len(nome))

print("="*20)
print("ge in pre_nome: ", "ge" in pre_nome)
print("LA in pre_nome: ", "LA" in pre_nome)

print("="*20)
print("X not in sobrenome: ", "x" not in sobrenome)
print("x in sobrenome: ", "x" in sobrenome)
```

concatenar

duplicar

comprimento

in e not in

```
==== RESTART: C:/Users/profa/O
Nome =  Angela Engelbrecht
=====
Comp pre_nome =  6
Comp sobrenome =  11
Comp nome =  18
=====
ge in pre_nome:  True
LA in pre_nome:  False
=====
X not in sobrenome:  True
x in sobrenome:  False
>>> |
```