**Python**

**Curso 1 – Introdução a nova versão da linguagem:**

1. **Aula 1 – Introdução e instalação do python 3:**
   1. Para instalar o python na versão mais recente para o windows acesse esse site: <https://www.python.org/downloads/windows/>
   2. Nos Linux e macOS o python 2 e 3 geralmente já vem instalado.
      1. Caso não venha abra o terminal e use o comando *sudo apt-get install python3*.
      2. No mc entrar no site <https://brew.sh>, abrir o terminal e colar a linha de código que aparecerá. Essa linha irá instalar um gerenciador de pacotes (que no Linux já vem por padrão) e agora você poderá instalar o python3.
      3. Use o comando *brew update* para atualizar.
      4. Use o comando *brew install python3* para instalar o python3.
      5. Se quiser também pode entrar no site <https://www.python.org/> e baixar como no windows.
   3. Caso não possa instalar o python na sua máquina por qualquer motivo que seja, você pode utilizá-lo no navegador através do site: <https://replit.com>. Lá você também poderá escrever linhas de código.
      1. Esse site serve para diversas linguagens, não somente pytho.
      2. Python3 vem como primeira opção após clicar em “Start coding”.
      3. Este é um dos melhores sites para codar.
   4. Todas as funções em uma linguagem tem “()” depois de seu nome.
      1. Ex.: exit(), print(), help().
   5. A Função print() imprime o que estiver dentro de () na tela, ex.: print(“olá, mundo”). Essa frase será exibida quando o programa for executado.
   6. Utilizar o parâmetro*, sep=”caracter”* no print, faz com que todos os espaços que separam cada valor colocado no print seja substituído pelo que estiver dentro das “ ”.
   7. O parâmetro *end=”\n”* faz com que após o final da frase a linha seja quebrada e o que vier depois dele aparecerá na linha de baixo. Esse é o padrão
      1. Se deixarmos apenas *end=””* ele não quebra a linha e o que deveria ir para linha de baixo como o padrão, continua na mesma linha.
      2. Isso também funciona para o pointer de digitação, quando utilizamos esse end as >>> no terminal ficam logo no final da frase digitada.
   8. Podemos definir uma variável colocando o nome dela e o que ela receberá: *pais = “Italia”*.
      1. Sempre que a variável *pais* for chamada o que aparecerá será o nome *Italia*.
      2. Caso a gente *print(pais)* o resultado será exibido como *Itália* no terminal.
      3. Outro exemplo *print(pais, “ganhou”, 5, “títulos”, “mundiais”)*. Resultado: *Itália ganhou 5 títulos mundiais*.
   9. Podemos utilizar a função *type(variável)* para saber seu tipo. Ex.: *type(pais)*, o terminal exibirá: *<class ‘str’>*, ou seja, essa variável é uma string.
   10. Tipos de variáveis:
       1. String (str): texto. Sempre quando algo é escrito dentro de “ ” ela se torna uma string.
       2. Inteiro (int): Números inteiros. Quando uma variável recebe um número (sem vírgula), ela se torna do tipo inteiro.
       3. Float: Números flutuantes/reais. Quando uma variável recebe um real (com vírgula), ela se torna do tipo float.
       4. Boolean: Verdadeiro ou falso.
       5. Uma variável sempre tem um tipo.
       6. O python não precisa que você defina explicitamente qual é o tipo de uma string, ele já faz isso e à altera dinamicamente, diferentemente de outras linguagens como C++, por exemplo.
   11. Em python utilizamos o padrão Snake\_Case para nomear variáveis, ou seja, se o nome da variável for composto as palavras serão separadas por “\_”, ao invés de CamelCase, que o padrão de outras linguagens como JavaScript.
2. **Aula 2 – Lidando Com a Entrada do Usuário:**
   1. Para criar condições fazemos do mesmo modo que em outras linguagens, usando *if* e *else*.
      1. A diferença do python para as outras é que não precisa colocar {} para separar a condição em um bloco, o que conta é a organização/indentação do código.
      2. Depois de declarar qual a condição colocamos um “:” antes de começar a escrever o que acontece.
   2. Sempre que recebemos algo pelo input, essa informação vem em formato de *str*, caso queira que venha em um formato específico como *int* ou *float*, precisamos declarar isso com o *numero\_str =* *int(input(“Digite um número:”))*, ou colocando uma segunda variável e fazendo ela receber o input na conversão: *numero\_str = input(“Digite um número:”); numero = int(numero\_str)*.
   3. Quando utilizamos 1 “=” estamos atribuindo um valor a algo, quando utilizamos 2 “==” estamos fazendo uma comparação entre conteúdo de duas variáveis para ver se são iguais, e, quando usamos 3 “===”, estamos fazendo uma comparação de conteúdo e tipo de variável entre 2 variáveis. É um seletor estritamente rígido.
   4. O sinal de “+” quando colocado entre duas str irá concatenar ambas e imprimir uma palavra só com as duas palavras de cada variável, quando colocado entre números ocorrerá a soma e quando colocado entre número e str o código não funciona, a menos que convertamos o número de str para int/float.
   5. Syntax Sugar = nome dado à “técnica” de copiar várias vezes a mesma string como por exemplo *print(“20” \*10)* o que acontecerá é que será impresso 10 vezes o número “20”. Isso é muito útil para facilitar coisas que seria muito trabalhosas.
3. **Aula 3 – Testando Valores:**
   1. Podemos usar o *elif* *(condição):* quando temos mais de 2 opções de condições. Essa função fica depois do *if* e antes do *else*.
   2. Para melhorar a legibilidade do código podemos criar variáveis que recebam as condições impostas nos *if’*s e nos *elif’*s, e colocar essas variáveis no lugar das condições, facilitando sua leitura.
   3. Operadores de comparação:
      1. < - menor que;
      2. > - maior que;
      3. <= - menor ou igual a;
      4. >= - maior ou igual a;
      5. == - igual a;
      6. != - diferente de.
   4. No python é aceita as formas com e sem “()” nas condições, no entanto, utilizamos () para deixar mais claro qual é a condição, principalmente ao utilizar os operadores lógicos *AND* e *OR*.
4. **Aula 4 – A sequência do Jogo:**
   1. Para que um pedaço do código se repita diversas vezes podemos, ao invés de copiar e colar esse código diversas vezes, utilizar a função *While*, que faz com que o código se repita uma determinada quantidade de vezes.
   2. Podemos escrever uma única frase no print e depois formatar pontos específicos utilizando: *print(“frase {} completa {}”, format(item\_1, item\_2))*. Dessa forma os {} serão substituídos pelo valor das variáveis no *format*, o primeiro trocado pela primeira variável a assim subsequentemente.
      1. Isso é bom pois facilita na hora da escrita, uma vez que podemos escrever a frase inteira de uma vez e no final apenas pedir para que os {} sejam substituídos pelos valores respectivos da ordem.
      2. Isso evita com que tenhamos que ficar abrindo e fechando aspas, colocando vírgulas dentre outros.
5. **Aula 5 – Iterando de Maneira Diferente:**
   1. Existe outro sistema de laços chamado *for* que funciona para repetir algo uma determinada quantidade de vezes, mas sem definir condições.
      1. Para utilizar essa função precisamos logo após o *for* determinar uma variável contadora (geralmente utilizado o *i*) para armazenar a quantidade de repetições.
      2. Após a variável precisamos definir que o alcance/área de repetição desse código, ou seja, se ele vai repetir de qual número a qual número e qual é o step dele, portanto, de quantas em quantas casas ele irá pular (padrão = 1).
      3. Como o “alcance” é uma função ele será seguido de ().
      4. Para exemplificar fica assim: *for i in range (1, 10, 2): código*.
      5. Diferentemente do *While*, o for já define a variável quando colocada em sua sequência, enquanto no *While* temos a necessidade de defini-la antes.
      6. Como pudemos ver no exemplo, a única parte que possui () é na função range, também sendo essa uma diferença sintática entre ambas as funções.
      7. Podemos juntar o for com um *input* para definir a quantidade de vezes que algo será repetido, deixando com que o usuário possa definir da maneira que quiser.
      8. Quando colocamos um número máximo para repetir, precisamos lembrar sempre que será repetido até o número anterior ao colocado. No exemplo acima pedimos para ir até o 10, porém, como é essa lógica que se segue, o código se repetirá até o 9.
      9. Para corrigir esse problema podemos colocar o 11 no final ou adicionar um *+1*, para que se tenha o mesmo resultado.
   2. Caso a sua condições do While (em um jogo de adivinhação, por exemplo) seja satisfeita e você tenha acertado o número, o jogo não irá parar até que o número de tentativas escolhido tenha se completado.
      1. Porém isso não faz sentido, uma vez que o jogo tenha sido ganho, não tem porquê continuar. Portanto, para resolver esse problema, podemos utilizar a função *break* no final do *if* de acerto, desse modo o jogo para de rodar quando a condição for satisfeita.
      2. Existe também o *continue* que é o oposto do break. Ele vai terminar a interação (de um aviso, por exemplo), mas não vai quebrar o laço e terminar o programa.
      3. O programa continua seguindo a partir do aviso inicial, porém pulando aquela interação, ou seja, parte para o próximo loop.
   3. Sobre interpolação de str’s:
      1. Utilizando o *format(1,2)* depois do *print* fazemos com que as {} sejam substituídas pelos valores respectivos, como já vimos. Contudo, nós podemos determinar qual parâmetro do format será exibido em qual {} colocando números dentro deles.
      2. O primeiro é sempre representado pelo ***0*** e os próximos a partir disso.
      3. Exemplo: *print(“O {} tem um total de {} mundiais, já o {} tem {} mundiais”, format(palmeiras, 0, corinthias, 3)*. Ele irá imprimir os valores respectivos por padrão, mas e se quisermos mudar?
      4. *print(“O {2} tem um total de {3} mundiais, já o {0} tem {1} mundiais”, format(palmeiras, 0, corinthias, 3)*. Determinando as interpolações assim, os valores são trocados, apresentando primeiro o corinthians e depois o palmeiras, diferentemente do anterior.
   4. Podemos formatar números e quantidades de espaços utilizados por palavras frases formatadas colocando dentro dos {} os:
      1. **:** = indica formatação;
      2. **.número** = indica quantidade de casas depois da vírgula/ponto;
      3. **F** = Indica que é um número float;
      4. **D** = Indica número inteiro;
      5. **Número depois dos :** = Indica quantas casas o conteúdo do format ocupará;
      6. **Número antes do número depois dos :** = Indica qual é o número que irá ocupar todas as casas que em branco.
   5. Comparações do python 2 com o 3: <https://pyformat.info/>
   6. Podemos também interpolar diretamente dentro dos {}, sem utilizar o .format. O nome dessa funcionalidade é *f-strings* ou *formatted string literals*.
      1. Para usar isso basta colocar toda a string entre aspas simples ‘ ‘ e um f antes delas, ficando assim: *print(f’texto {qualquer} de {exemplo}’)*. Assim as informações recebidas pelas variáveis *qualquer* e *exemplo*, serão substituídas no texto printado.
      2. Isso também nos permite adicionar funções de formatação dentro, como *{qualquer.lower()}* para colocar todas as letras do valor da variável em minúsculas ou *.upper()* para em capslock e assim por diante.