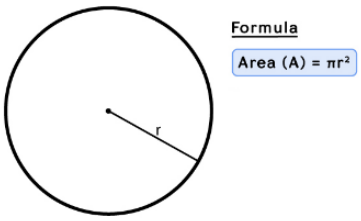
Extra Exercises (Basic/Intermediate Levels)

P001. Your job is to write a code that asks the user for the distance (in km) travelled and the amount of fuel (in litres) the car uses. The output should print the consumption in litres spent at 100 km.

P002. Consider a circle with a radius of **r**. The area of a circle is given as shown in the picture below.

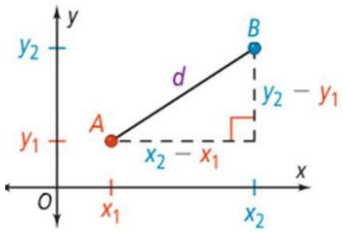


Your job is to implement a function **area_circ(r)** that calculates the area of a circle with radius **r**.

P003. Consider two integers, numbers **a** and **b**. Your job is to implement the function **summation(a, b)**, which takes these two integers as arguments and returns their sum.

P004. The conversion between temperature measurements in degrees Fahrenheit and Celsius can be done using the formula $C = \frac{5}{9} * (F - 32)$, where F is the temperature in degrees Fahrenheit, and C is the temperature in degrees Celsius. Considering this information, your job is to implement a function **celsius(F)** that reads a given temperature F (degrees Fahrenheit) and returns the equivalent temperature converted to C (degrees Celsius).

P005. The distance d between two points $A = (x_1, y_1)$ and $B = (x_2, y_2)$ on the coordinate plane is given by the formula $d = \sqrt{(x_2 - x_1)^2 + (y_2 - y)^2}$ as you can see in the picture below.



Your job here is to implement a function **dist(x1,y1,x2,y2)** that calculates the distance using the equation above.

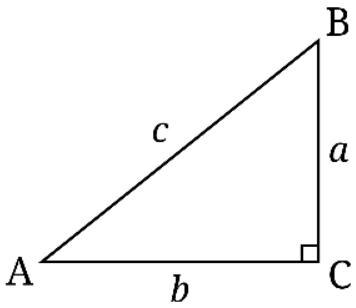
P006: A **degree** can be defined as the angle made by one part of 360 equally divided parts of a circle at the centre with a radius of **r**. The degree is calculated based on the rotation of the sun. Therefore, a degree equals the 1/360 angle of a complete sun rotation. One degree is equivalent to 60 minutes, and a further one minute is equivalent to 60 seconds. Also, considering that 360 degrees corresponds to 2πRadians, your job is to implement a function **radians(degrees, mins, secs)** that converts angle values in degrees, minutes, and seconds to radians.

P006.1. Similarly, you must implement a function **called seconds(hours,mins,secs), which, given a duration of hours, minutes,** and seconds, calculates and returns that same duration in seconds.

P007. Consider a grade obtained in a test valued from 0 to 100. Your job is to implement a function **classification(p)** that returns a status, which should be a string, based on the grade as follows.

< 0 or > 100 -> Grade not Valid
≥ 0, < 50 -> Not Enough to Pass
≥ 50, < 70 -> Enough to Pass
≥ 70, < 80 -> Good
≥ 80, < 90 -> Very Good
≥ 90, ≤ 100 -> Excellent

P008. Consider the right-angled triangle in the picture below.



Your job is to implement a function **hypotenuse(a,b),** which, given the dimensions **a** and **b** of the two sides of a right-angled triangle, calculates the length of the hypotenuse.

P009. Interest is usually calculated based on an integer number of periods (the number of years, for example). However, in some situations, it is useful to calculate interest as a continuous function of time. The formula for calculating the value of an investment over time **t** is as follows.

$$P(t) = P_0 * e^{r*t}$$

In the formula, $\mathbf{P_0}$ is the initial amount invested, **r** is the interest rate, and **t** is the time since the investment was made. Your job is to implement a function **P(P_0, r, t)** that returns the value of an investment $\mathbf{P_0}$ at time **t**, at an interest rate **r**.

P010. Your job is to write a program that reads an integer amount of USD and shows how to pay this amount in $100, $50, $20, $10, $2, and $1 banknotes.

Example: Amount in $? 258
$100 banknotes = 2
$50 banknotes = 1
$20 banknotes = 0
$10 banknotes = 0
$5 banknotes = 1
$2 banknotes = 1
$1 banknotes = 1

Hint: The quotient of dividing the total amount by 20 gives us the number of $20 banknotes. Be careful to use integer divisions and not float.

P011. Consider a list as follows:

$$\text{list} = [12, 25, 43, 34, 51, 116]$$

Your job is:

(a) write a **for** loop that prints each value in the **list** on a separate line.
(b) write another **for** loop that prints the number, its square, and its squared root on each line.
(c) Write a loop that adds up all the numbers in the list using an auxiliary variable total. Print each number in the list on a separate line, and the partial sum is added to that number.

P012. Your job is to implement a function **square(n)**, which, for the first n positive integers, prints the number and its square on each line, separated by a space. You can assume that n > 0.

P013. The formula for calculating the final value of a deposit with compound interest capitalised per **month** at an annual rate of **r** is:

$$C_F = C_I \times \left(1 + {}^{r}/_{12}\right)^{n}$$

In the formula, $\mathbf{C_I}$ is the initial capital, $\mathbf{C_F}$ is the final capital, **r** is the interest rate, and **n** is the months the deposit will last.

Example: for an initial capital of $6,000; an annual interest rate of 2% for 36 months, we get around $6,370.70.

Your job is to write a code that asks for the initial capital ($\mathbf{C_I}$), the interest rate (**r**), the number of months (**n**) and that prints the final capital after 0, 1,..., **n** months in each line.

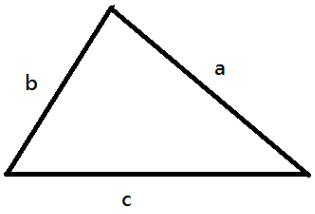
P014. The Sum of Squared Deviations (SQD) is a common measure for evaluating errors. Given a list of **n** deviations $(d_1, d_2, \cdots, d_n)$, the SQD can be calculated as follows.

$$\text{SQD} = \sum_{i=1}^{n} d_i^2$$

Your job is to create a list of deviations and implement a function **sqd(d)** that returns the SQD of your created list.

P015. Suppose the current price of the diesel is USD 1.75/litre. Your job is to create a list, in which each element represents the amount of diesel you filled up each time of your travel. Then, you must implement a function **value(v)** that calculates the total value spent on your travel with fuel.

P016. Consider any triangle with sides **a**, **b** and **c,** as in the picture.



The Heron's Formula (below) can calculate the area **A** of a triangle based on its sides.

$$A = \sqrt{s * (s - a) * (s - b) * (s - c)}$$

In the formula, $s = {(a + b + c)}/{2}$ is the semi-perimeter of the triangle. Your job is to implement a function **triangle_area(a,b,c)** that calculates the area of a triangle using Heron's Formula.

P017. Your job is to implement a function **triangle(a,b,c)** to check if three given numbers define a triangle with sides **a**, **b**, and **c**.

Hint: Remember that in any triangle, each side has a length **less** than the sum of the other two sides and **greater** than their absolute difference.

P018. Your job is to implement a function **classification(p)** which, given the score p obtained in an exam (from 0 to 100), returns a classification message according to the table below:

Grade	Status
< 0 or > 100	"Grade not Valid"
≥ 00, < 50	"Not enough to Pass"
≥ 50, < 70	"Enough to Pass"
≥ 70, < 80	"Good"
≥ 80, < 90	"Very Good"
≥ 90, < 100	"Excellent"

P019. The arithmetic mean of **n** numbers is given by

$$\frac{1}{n}\sum_{i=1}^{n} \alpha_i = \frac{\alpha_1 + \alpha_2 + \cdots + \alpha_{n-1} + \alpha_n}{n}$$

Your job is to write a code that asks the user the number **n** of elements to calculate the arithmetic mean. Then, the code should ask for each element. Finally, you should implement a function **arithmetic_mean(n)** that returns the arithmetic mean of the **n** numbers provided.

P020. The geometric mean of **n** numbers is given by

$$\left(\prod_{i=1}^{n} \alpha_i\right)^{\frac{1}{n}} = \sqrt[n]{\alpha_1 * \alpha_2 * \cdots * \alpha_{n-1} * \alpha_n}$$

Your job is to write a code that asks the user the number **n** of elements to calculate the geometric mean. Then, the code should ask for each element. Finally, you should implement a function **geometric_mean(n)** that returns the geometric mean of the **n** numbers provided.

P021. A year is a "leap year" if it is divisible by 4, except if it is a multiple of 100 and not divisible by 400. Your work is to write a code asking for a **year** and implement a function **leap(year),** which results in **True** if n is a "leap year" and **False** otherwise. Then you should complete the code using the function **leap(year),** to write the "leap years" within a range between two years.

P022. The factorial of a number is given by

$$n! = n \times (n-1) \times \cdots \times 1$$

Your job is to implement a function **factorial(n),** which returns the factorial of a number **n**.

P023. A number is a **"Perfect Square"** if, for any natural number **k**, it can be written as the sum of the **k** first odd numbers, which means **1 + 3 + 5 + ... + k**. The first five perfect square numbers are 1, 4, 9, 16 and 25. Your job is to implement a function **perfect_square(n)** that returns **"True"** when the number **n** is a perfect square or **"False"** otherwise.

P024. A number **n** is triangular if it can be written as a sum **1 + 2 + ... + k** for any natural **k**. The first five triangular numbers are 1, 3, 6, 10 and 15. Your job is to implement a function **triangular(n)** that returns **"True"** when the number **n** is triangular or **"False"** otherwise.

P025. Counting decimal digits in the representation of a number by doing integer divisions by ten is possible. For instance, the number 9733 has four digits because it can be successively divided by ten four times, retrieving 973, 97, 9 and 0 as quotients (the divisions should stop at 0). Considering this, your job is to implement a function **digits(n)**, which results in the number of decimal digits of **n**.

P026. A number **d** is a proper divisor of **n**, if, and only if the rest of the division of **n** by **d** is zero and **d < n**. The smallest proper divisor of an integer **n** is the smallest integer **d** such that **d > 1** and **d** divides **n** (i.e. the rest of the division of **n** by **d** is zero). Your job is to implement a function **mindiv(n)** to calculate the smallest proper divisor of **n**.

> **P026.1.** An integer **n** is a **prime number** if its smallest proper divisor is **n**. Now, your job is to use the function implemented before implementing another function **prime(n)**, which returns **True** if n is prime and **False** otherwise.
>
> **P026.2** Now, your job is to implement a function **divisors(n)**, which returns the list of proper divisors of **n** sorted in ascending order. **Example**: divisors(12) = [1, 2, 3, 4, 6]
>
> **P026.3** Finally, considering that an integer number **n** is perfect if it is equal to the sum of its proper divisors, your job is to implement a function **perfect(n)** that returns True, if **n** is perfect and False otherwise. **Example**: 6 is perfect, because 6 = 1 + 2 + 3.

P027. The number **π** is a mathematical constant that is the ratio of a circle's circumference to its diameter, approximately equal to 3.14159. This number can be approximated by using Leibniz's

formula, which results in the summation of the first **k** terms of the series. The Leibniz's formula is given by:

$$\pi = 4 \times \sum_{n=0}^{\infty} \frac{(-1)^n}{2n+1} = 4 \times \left(1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \cdots\right)$$

Your job is to implement a function **leibniz(k)**, which results in the sum of the **k** first terms of this series.

P028. In mathematics, the **binomial coefficients** are the positive integers that occur as coefficients in the binomial theorem. Commonly, a binomial coefficient is indexed by a pair of integers **n ≥ k ≥ 0** and is written as $\binom{n}{k}$. It is the coefficient of the $\mathbf{x^k}$ term in the polynomial expansion of the binomial power $\mathbf{(1 + x)^n}$. The following multiplicative formula can compute this coefficient:

$$\binom{n}{k} = \frac{n!}{k!\,(n-k)!} = \frac{n \times (n-1) \times \cdots \times (n-k+1)}{k \times (k-1) \times \cdots \times 1}$$

Your job is implementing a function **binomial(n,k)**, which calculates the binomial coefficients.

P029. Two integers are **coprime** if and only if their **greatest common divisor (gcd)** is 1. For example, 4 and 15 are coprime, but 4 and 14 are not (because 2 is a common divisor). Your job is to implement a function **coprime(n),** which returns the list of integers between 1 and **n** that are coprime with **n**.

P030. Newton's Method can compute square roots $x = \sqrt{a}, a > 0$. The algorithm starts with some guess $x_0 > 0$ and computes the sequence of improved guesses through

$$x_{n+1} = \frac{1}{2} \times \left(x_n + \frac{a}{x_n}\right)$$

A limit for the computation can be defined by setting an error $\in \rightarrow |x_{n+1} - x_n| < \in$. Your job is to implement a function **root(q, epsilon)** which approximates the root of **q** by iterating until the difference in absolute value between successive approximations is less than **ϵ.**

P031. Your job is to implement a function **standard_deviation(lst)**, which results in the sample standard deviation of a list containing **n** values, that is,

$$\sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

where $\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$ is the arithmetic median of the values. You can assume that **n > 1**.