Les fonctions en `python`

Comme dans tous langages, il est essentiel d'organiser un programme en petites tâches séparées. Une façon commode est d'utiliser des fonctions. Elles nous serviront évidemment lorsque l'on souhaite définir une fonction mathématique mais plus généralement lorsqu'une partie de code est utilisée plusieurs fois (sous-entendu, plutôt que de faire des *copier-coller*, construisez des fonctions...)

Fonctions natives et leur documentation

Une fonction est un bloc réutilisable d'instructions, qui dépend de paramètres (appelés arguments) et renvoie un résultat. Elle servent à découper un code complexe en briques élémentaires plus simples et à éviter de recopier plusieurs fois des instructions très proches. Elle permettent aussi à un développeur de fournir une fonctionnalité avancée à des utilisateurs « quelconques » sans qu'ils aient à savoir comment la fonction est codées. Ils doivent juste savoir comment utiliser la fonction, pour cela elle est fournit avec sa documentation.

Le language `python` contient nativement de nombreuses fonctions comme `print` , `input` . Ces fonctions sont reconnaissable à la synthaxe : il s'agit d'un mot suivit par des parenthèses dans lesquelles se trouvent zéro, une ou plusieurs variables utilisée par la fonction.

In [2]:
```
print(1)
print("Bonjour. ", "1+2 = ", 1+2, sep="|")
print(f"Bonjour. 1+2 = {1+2}", end='\n***')
```

```
1
Bonjour. |1+2 = |3
Bonjour. 1+2 = 3
***
```

Toutes ces fonctions sont fournies avec leur documentation qui est accessible à l'aide de la fonction `help` :

In [3]:
```
help(print)
```

```
Help on built-in function print in module builtins:

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

    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 sy
s.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.
```

In [4]:
```
import math
help(math)
```

```
Help on module math:

NAME
    math

MODULE REFERENCE
    https://docs.python.org/3.8/library/math

    The following documentation is automatically generated from th
e Python
    source files.  It may be incomplete, incorrect or include feat
ures that
    are considered implementation detail and may vary between Pyth
on
    implementations.  When in doubt, consult the module reference
at the
    location listed above.

DESCRIPTION
    This module provides access to the mathematical functions
    defined by the C standard.

FUNCTIONS
    acos(x, /)
        Return the arc cosine (measured in radians) of x.

    acosh(x, /)
        Return the inverse hyperbolic cosine of x.

    asin(x, /)
        Return the arc sine (measured in radians) of x.
```

```
    asinh(x, /)
        Return the inverse hyperbolic sine of x.

    atan(x, /)
        Return the arc tangent (measured in radians) of x.

    atan2(y, x, /)
        Return the arc tangent (measured in radians) of y/x.

        Unlike atan(y/x), the signs of both x and y are considered
.

    atanh(x, /)
        Return the inverse hyperbolic tangent of x.

    ceil(x, /)
        Return the ceiling of x as an Integral.

        This is the smallest integer >= x.

    comb(n, k, /)
        Number of ways to choose k items from n items without repe
tition and without order.

        Evaluates to n! / (k! * (n - k)!) when k <= n and evaluate
s
        to zero when k > n.

        Also called the binomial coefficient because it is equival
ent
        to the coefficient of k-th term in polynomial expansion of
the
        expression (1 + x)**n.

        Raises TypeError if either of the arguments are not intege
rs.
        Raises ValueError if either of the arguments are negative.

    copysign(x, y, /)
        Return a float with the magnitude (absolute value) of x bu
t the sign of y.

        On platforms that support signed zeros, copysign(1.0, -0.0
)
        returns -1.0.

    cos(x, /)
        Return the cosine of x (measured in radians).

    cosh(x, /)
        Return the hyperbolic cosine of x.

    degrees(x, /)
```

```
        Convert angle x from radians to degrees.

    dist(p, q, /)
        Return the Euclidean distance between two points p and q.

        The points should be specified as sequences (or iterables)
of
        coordinates.  Both inputs must have the same dimension.

        Roughly equivalent to:
            sqrt(sum((px - qx) ** 2.0 for px, qx in zip(p, q)))

    erf(x, /)
        Error function at x.

    erfc(x, /)
        Complementary error function at x.

    exp(x, /)
        Return e raised to the power of x.

    expm1(x, /)
        Return exp(x)-1.

        This function avoids the loss of precision involved in the
direct evaluation of exp(x)-1 for small x.

    fabs(x, /)
        Return the absolute value of the float x.

    factorial(x, /)
        Find x!.

        Raise a ValueError if x is negative or non-integral.

    floor(x, /)
        Return the floor of x as an Integral.

        This is the largest integer <= x.

    fmod(x, y, /)
        Return fmod(x, y), according to platform C.

        x % y may differ.

    frexp(x, /)
        Return the mantissa and exponent of x, as pair (m, e).

        m is a float and e is an int, such that x = m * 2.**e.
        If x is 0, m and e are both 0.  Else 0.5 <= abs(m) < 1.0.

    fsum(seq, /)
        Return an accurate floating point sum of values in the ite
```

```
rable seq.

        Assumes IEEE-754 floating point arithmetic.

    gamma(x, /)
        Gamma function at x.

    gcd(x, y, /)
        greatest common divisor of x and y

    hypot(...)
        hypot(*coordinates) -> value

        Multidimensional Euclidean distance from the origin to a p
oint.

        Roughly equivalent to:
            sqrt(sum(x**2 for x in coordinates))

        For a two dimensional point (x, y), gives the hypotenuse
        using the Pythagorean theorem:  sqrt(x*x + y*y).

        For example, the hypotenuse of a 3/4/5 right triangle is:

            >>> hypot(3.0, 4.0)
            5.0

    isclose(a, b, *, rel_tol=1e-09, abs_tol=0.0)
        Determine whether two floating point numbers are close in
value.

          rel_tol
            maximum difference for being considered "close", relat
ive to the
            magnitude of the input values
          abs_tol
            maximum difference for being considered "close", regar
dless of the
            magnitude of the input values

        Return True if a is close in value to b, and False otherwi
se.

        For the values to be considered close, the difference betw
een them
        must be smaller than at least one of the tolerances.

        -inf, inf and NaN behave similarly to the IEEE 754 Standar
d.  That
        is, NaN is not close to anything, even itself.  inf and -i
nf are
        only close to themselves.
```

```
    isfinite(x, /)
        Return True if x is neither an infinity nor a NaN, and Fal
    se otherwise.

    isinf(x, /)
        Return True if x is a positive or negative infinity, and F
    alse otherwise.

    isnan(x, /)
        Return True if x is a NaN (not a number), and False otherw
    ise.

    isqrt(n, /)
        Return the integer part of the square root of the input.

    ldexp(x, i, /)
        Return x * (2**i).

        This is essentially the inverse of frexp().

    lgamma(x, /)
        Natural logarithm of absolute value of Gamma function at x
    .

    log(...)
        log(x, [base=math.e])
        Return the logarithm of x to the given base.

        If the base not specified, returns the natural logarithm (
    base e) of x.

    log10(x, /)
        Return the base 10 logarithm of x.

    log1p(x, /)
        Return the natural logarithm of 1+x (base e).

        The result is computed in a way which is accurate for x ne
    ar zero.

    log2(x, /)
        Return the base 2 logarithm of x.

    modf(x, /)
        Return the fractional and integer parts of x.

        Both results carry the sign of x and are floats.

    perm(n, k=None, /)
        Number of ways to choose k items from n items without repe
    tition and with order.

        Evaluates to n! / (n - k)! when k <= n and evaluates
```

```
        to zero when k > n.

        If k is not specified or is None, then k defaults to n
        and the function returns n!.

        Raises TypeError if either of the arguments are not intege
rs.
        Raises ValueError if either of the arguments are negative.

    pow(x, y, /)
        Return x**y (x to the power of y).

    prod(iterable, /, *, start=1)
        Calculate the product of all the elements in the input ite
rable.

        The default start value for the product is 1.

        When the iterable is empty, return the start value.  This
function is
        intended specifically for use with numeric values and may
reject
        non-numeric types.

    radians(x, /)
        Convert angle x from degrees to radians.

    remainder(x, y, /)
        Difference between x and the closest integer multiple of y
.

        Return x - n*y where n*y is the closest integer multiple o
f y.
        In the case where x is exactly halfway between two multipl
es of
        y, the nearest even value of n is used. The result is alwa
ys exact.

    sin(x, /)
        Return the sine of x (measured in radians).

    sinh(x, /)
        Return the hyperbolic sine of x.

    sqrt(x, /)
        Return the square root of x.

    tan(x, /)
        Return the tangent of x (measured in radians).

    tanh(x, /)
        Return the hyperbolic tangent of x.
```

```
        trunc(x, /)
            Truncates the Real x to the nearest Integral toward 0.

            Uses the __trunc__ magic method.

    DATA
        e = 2.718281828459045
        inf = inf
        nan = nan
        pi = 3.141592653589793
        tau = 6.283185307179586

    FILE
        /Users/benjamin/anaconda3/envs/ens/lib/python3.8/lib-dynload/m
    ath.cpython-38-darwin.so
```

Définir ses propres fonctions

En `python`, il y a deux types de fonction que nous utiliserons régulièrement :

1. les fonctions avec le mot clé `def` sont utilisées pour les fonctions nécessitant plusieurs lignes de code. `python` va créer à la précompilation un bloc de code avec des entrées et des sorties ; ce code sera utilisé à chaque appel de la fonction. Syntaxe

```
def ma_fonction(argument_0, argument_1, ...):
    # un code qui fait ce que l'on veut
    return (retour_0, retour_1, ...)
```

2. Les fonctions avec le mot clé `lambda` sont utilisées pour les fonctions très courtes. Dans ce cas, `python` remplace le code de la fonction directement aux différents endroits où la fonction est appelée. Syntaxe :

```
f = lambda argument_0, argument_1, ...: (retour_0, retour_1, ...)
```

Notez l'utilisation des `:` dans les deux cas. Il s'agit de la même convention qui permet de définir des blocs avec l'indentation. Comme l'expression de la λ-fonction doit être court, il est possible de le mettre sur la même ligne.

Remarques :

- il est possible de définir une fonction n'importe où dans un programme (dans une fonction aussi)
- comme tout est objet en `python`, on peut très facilement passer une fonction en paramètre d'une autre fonction.

In [5]:
```
from plan import plan_cours
plan_cours(3)
```

Plan du cours

1. lambda_fonction (SC03_1_lambda_fonction.ipynb)
2. fonction_def (SC03_2_fonction_def.ipynb)
3. arguments (SC03_3_arguments.ipynb)
4. documentation (SC03_4_documentation.ipynb)

In [ ]:
```

```