

Preguntas

Encabezado

```
<tipo = encabezado[, cifras = _int_]>
```

```
<variables> ?
_v1[, vn]*_ = _expresion_ +
```

```
<pregunta>
_texto de la pregunta_ @exp
```

Selección única

También acepta cifras = _int_. Los índices para opcion son 0-indexados.

```
<tipo = seleccion unica[, opcion = (i[ & i]* | todos)]>
```

```
<variables> ?
_v1[, vn]*_ = _expresion_ +
```

```
<pregunta>
_texto de la pregunta_ @exp
```

```
<item> +
_texto de respuesta o distractor_ @exp
```

Respuesta corta

```
<tipo = respuesta corta[, cifras = _int_]>
```

```
<variables> ?
_v1[, vn]*_ = _expresion_ +
```

```
<pregunta>
_texto de la pregunta_ @exp
```

```
<item[, error = _error_][, factor = _entre_0_y_1_]> +
_respuesta_
```

Puede utilizar `error = math.inf` en caso de que deba aceptar todas las respuestas como correctas.

@-expresión

- `@<_expr_>`
- `@{ _expr_ }`
- `@(_expr_)`
- `@[_expr_]`
- Una `@` seguida de cualquier símbolo, que es el mismo que se utiliza para cerrar, por ejemplo podría ser `@|_expr_|`, siempre y cuando el símbolo no se encuentre en la expresión.

Operadores

■ `+` `-` `*` `/` `//` `%` `**`

Suma, resta, multiplicación, división, división entera, residuo y potencia.

Expresiones lambda

```
f = lambda x,y,z : <expresion que depende de x,y,z>
r = f(x,y,z)
```

Una forma de agregar funciones específicas. El problema es que debido a que Python utiliza evaluación perezosa (*lazy evaluation*), no se pueden utilizar variables definidas previamente en la expresión derecha de la definición de *f*. En caso de que se deban utilizar variables *m*, *n* definidas previamente, lo que se sugiere es que se agreguen como variables en la definición de la función:

```
m = ...
n = ...
f = lambda u,v,x,y,z : <expresion de u,v,x,y,z>
r = f(m,n,x,y,z)
```

Funciones aleatorias

Solamente se utilizan a la hora de definir variables.

- `randrange(start = 0, stop)`
- `randrange(start, stop, step = 1)`
Entero *n* = *start* + *k* * *step*, *k* entero no negativo tal que *start* <= *n* < *stop*.
- `randint(a,b)`
Entero *n* en el conjunto $\{a, a + 1, \dots, b\}$.
- `choice(seq)`
Un elemento del iterable no vacío *seq*.
- `sample(seq, k)`
Muestra no ordenada de tamaño *k* de un iterable. Si *k* es el mismo tamaño del iterable, devuelve una permutación del mismo.
- `random()`
Flotante con distribución uniforme en $[0, 1)$.
- `uniform(a, b)`
Flotante con distribución uniforme en $[a, b]$.
- `gauss(mu, sigma)`
Un elemento *x* que sigue una distribución gaussiana con media *mu* y desviación estándar *sigma*.

Sucesiones

Toda tupla, lista o conjunto, es 0-indexado.

- `range(<stop>)`
- `range(<start>, <stop>[, <step>])`
El inicio predeterminado es 0. El valor es siempre menor que *stop*.
- Lista al concatenar dos o más sucesiones:
`ls = [*range(...), *range(...)]`.
- Listas por comprensión:
`ls = [i**2 for i in range(1, 11) if i % 5]`
R/ [1, 4, 9, 16, 36, 49, 64, 81]

Funciones de Python

- `len(xs), sorted(xs)`
Longitud de *xs*; devuelve *xs* ordenada.
- `round(x[, ndigits=None])`
Redondea un número, y se puede especificar el número de dígitos a utilizar. Ninguno es el valor predeterminado.
- `sum(ls)`
Suma de los elementos de *ls*.

Texto

1. Si *num* es un entero, se puede imprimir el entero con signo como: `'%d' % num`
2. Si *x* es un número con decimales:
 - `'%f' % x` imprime el número con 6 decimales de manera predeterminada.
 - `'%.nf' % x`, donde *n* es un entero positivo, imprime el número con *n* decimales.
 - `'%e' % x` imprime el número en notación científica con 6 decimales de forma predeterminada.
 - `'%.ne' % x`, donde *n* es un entero positivo, imprime el número en notación científica con *n* decimales.
3. Si se necesita concatenar varios elementos:
`'(%d, %d)' % (m, n)` representa un par ordenado de enteros, o por ejemplo:
'Utilice el método %s para %s.' % (*var1*, *var2*) donde *var1* y *var2* son variables de texto.

Conjuntos

- Se construyen utilizando valores entre llaves, o con el comando `set` sobre una lista.
`s1 = set(sample(range(1, 16), 5))`
`s2 = {3, 5, 7, 11}`
- `ss.difference(s1, s2, ...)`
`ss.intersection(s1, s2, ...)`; `ss.union(s1, s2, ...)`
Se resta a *ss* cada uno de los conjuntos *s_i* y se devuelve el resultado; devuelve la intersección de los conjuntos *ss*, *s1*, ...; y lo mismo para la unión.
- `ss.symmetric_difference(s1)`
Diferencia simétrica de *ss* con *s1*.
- `ss.isdisjoint(s1)`; `ss.issubset(s1)`
True si *ss* y *s1* son disjuntos o si *ss* es subconjunto de *s1*.

Biblioteca math

- `math.degrees(x)`; `math.radians(x)`
Conversión de radianes a grados; y de grados a radianes.
- `math.acos(x)`; `math.acosh(x)`; `math.asin(x)`;
`math.asinh(x)`; `math.atan(x)`; `math.atan2(y, x)`;
`math.atanh(x)`; `math.cos(x)`; `math.cosh(x)`; `math.sin(x)`;
`math.sinh(x)`; `math.tan(x)`; `math.tanh(x)`
Funciones trigonométricas y sus inversas (en radianes).
- `math.isqrt(x)`; `math.sqrt(x)`; `math.exp(x)`; `math.log(x)`;
`math.log10(x)`
Parte entera de la raíz de *x*; funciones en punto flotante para la raíz, exponencial, logaritmo natural y logaritmo en base 10.

- `math.erf(x); math.erfc(x); math.gamma(x)`
Funciones de error y función gamma.
- `math.gcd(a_1, a_2, ..., a_n)`
Máximo común divisor del conjunto $\{a_1, a_2, \dots, a_n\}$.
- `math.fmod(x, y); math.modf(x) -> (float, float)`
Función residuo para variables de tipo flotante; parte decimal y parte entera de x .
- `math.ceil(x); math.floor(x); math.trunc(x)`
Funciones de manejo de decimales.
- `math.factorial(a); math.comb(n, r); math.perm(n, r)`
Factorial, combinaciones y permutaciones.
- `math.hypot(x, y); math.dist(xs, ys)`
 $\sqrt{x^2 + y^2}$ y distancia entre los vectores n -dimensionales `xs` y `ys`
- `math.prod(ls)`
Producto de los elementos de `ls`.

Biblioteca Fraction

Permite representar números como fracciones:

```
f = Fraction(a, b)
```

y realizar operaciones aritméticas con ellos, las cuales se simplifican de manera automática. Se puede extraer luego su `numerator` y `denominator` mediante `f.numerator` y `f.denominator`. Aunque es un poco engorroso, para imprimir se utiliza `fraccion(f.numerator, f.denominator)`.

Funciones CoronaTec

Funciones creadas en el proyecto. Se pueden ir agregando según se requiera.

Biblioteca mate

- `mate.descomponer(f)`
Dado un número flotante f , lo descompone en su mantisa y su exponente. Devuelve un par ordenado, donde el primer valor es un flotante en el intervalo $[1, 10)$, y el segundo es un entero.
- `mate.descUnid(f)`
Dado un número flotante f , lo descompone en un valor y unidades. Devuelve un par ordenado, donde el primer valor es un flotante en el intervalo $[1, 1000)$, y el segundo es una cadena de texto a utilizar.
- `mate.digSignif(x, d)`
Redondeo de un número de punto flotante x , según la cantidad de dígitos significativos d .
- `mate.divisores(n)`
Devuelve una lista de los divisores positivos del número entero n .
- `mate.factorios(a)`
Factorización de a . Por ejemplo `mate.factorios(1000)` devuelve la lista $[(2, 3), (5, 3)]$, que representa a $2^3 \cdot 5^3$.

Biblioteca txt

Para imprimir en la pregunta, o en las opciones de una pregunta de selección única. Texto en \LaTeX . Se asume que se está en modo matemático.

- `txt.coef(a: int, conSigno: bool = False)`
Coeficiente que precede a una variable entera. Si $a = 1$ entonces no se escribe (o se escribe solo un signo $+$ si `conSigno`). De manera similar ocurre si $a = -1$. En caso contrario imprime el valor.
- `txt.conSigno(n: int)`
Imprime el entero con signo.
- `txt.decimal(x: float, cifras: int, conSigno = False)`
Imprime un flotante según el número de cifras significativas indicado.
- `txt.expo(n: int)`
Si $n \neq 1$ no imprime nada. Si no imprime $\cdot 10^n$.
- `txt.fraccion(num: int, den: int, conSigno=False, signoNum=False, dfrac=True, coef=False)`
Simplifica e imprime utilizando `dfrac`. De manera opcional se especifica si se obliga el signo $+$; si el signo se imprime en el numerador o se imprime afuera; si se utiliza `tfrac` en lugar de `dfrac`, y si es un coeficiente, de manera que si da 1 o -1 no se imprima el valor.
- `txt.raiz(a: int, n: int = 2, conSigno = False)`
Simplifica y escribe la raíz respectiva. De manera opcional se puede especificar el índice n , y si se imprime un $+$ al inicio.
- `txt.sumar(a: int, conSigno = True, arg = '')`
Si $a = 0$, no escribe nada. Si `arg` es vacío, escribe el número. En caso contrario, escribe el coeficiente junto con el argumento.
- `txt.texto(n: int)`
Para $n > 0$, devuelve el texto que representa al valor de n .

Biblioteca vector

- `vector.ceros(n: int)`
Vector de ceros de tamaño n .
- `vector.aleatorio(n: int, vmin: int, vmax: int, factor: Fraction = Fraction(1,1))`
Vector de números aleatorios. Genera inicialmente enteros en `vmin..vmax`, y luego los multiplica por `factor`, que puede ser un número de punto flotante, o un número como fracción.
- `vector.latex(v: Vector, txtSep: str, ndigits: int = -2, ceros: int = 3)`
Imprime un vector. El encabezado en \LaTeX lo especifica el usuario. El texto separador da el formato entre elementos: por ejemplo `' , ' o ' & ' ' (para vectores fila)`, o `' \\\ ' (para vector columna)`. De manera opcional se puede especificar el número de dígitos a imprimir. Si es -1 , intenta imprimir la menor cantidad n , tal que `round(10**n * v[i], ceros)` sea un entero. Observe que -2 es el valor predeterminado, lo que asume que lo que va a imprimir son fracciones.
- `vector.kprod(k: float, v: Vector)`
 $k \cdot v$
- `vector.pprod(u: Vector, v: Vector)`
 $u \cdot v$ Producto punto.
- `vector.suma(v1: Vector, v2: Vector)`
 $v_1 + v_2$

Biblioteca matriz

- `matriz.aleatorio(nfilas: int, ncols: int, vmin: int, vmax: int, factor: Fraction = Fraction(1,1))`
Matriz de valores aleatorios. Ver `vector.aleatorio`

- `matriz.copia(A)`
Una copia de la matriz A . No debería ser necesario utilizar esta función.
 - `matriz.det(A)`
Determinante de la matriz A . Construye una matriz triangular superior utilizando pivoteo, y multiplica los valores de la diagonal para tener el valor del determinante.
 - `matriz.update(A, irow, icol, valor)`
Cambia el valor de una entrada de la matriz.
 - `matriz.dominante(n, vmin, vmax, factor = Fraction(1,1))`
Matriz cuadrada diagonalmente dominante. El rango de valores es solamente para los elementos fuera de la diagonal.
 - `matriz.gaussSeidel(A: Matriz, bb: Vector, x0: Vector, npasos: int)`
Vector que se obtiene al aplicar n pasos de Gauss-Seidel.
 - `matriz.intercambiar(mat, fila1: int, fila2: int)`
Matriz resultante al intercambiar dos filas de una matriz.
 - `matriz.jacobi(A: Matriz, bb: v.Vector, x0: v.Vector, npasos: int)`
Vector que se obtiene al aplicar n pasos de Jacobi.
 - `matriz.latex(mat: Matriz, decimal = False, dfrac = False, espacio = '[1ex]', cifras = -20, ceros = 3)`
Imprime una matriz. El encabezado \LaTeX lo especifica el usuario. `decimal` es si imprime punto decimal, y si no, lo imprime con `tfrac`. `dfrac` especifica si se utiliza `\dfrac` en lugar de `\tfrac`. `espacio` es el texto que pone al final de línea. Tiene como predeterminado `' [1ex] '`. Puede seleccionar también un string vacío: `' '`. `cifras` se refiere al número de cifras significativas. Solo si `cifras == -1` se utiliza el menor valor posible, y solo entonces se lee la variable `ceros`, que quiere decir cuántos ceros consecutivos se leen para descartar el resto. Es decir, con `ceros=3`, el valor 2.30005 se lee como 2.3.
 - `matriz.permutar(A: Matriz, perm: List[int])`
Coloca las filas de A , según la permutación dada.
 - `matriz.pivote(L: Matriz, U:Matriz, P:Matriz, ifila: int) -> (Matriz, Matriz, Matriz)`
Un paso de factorización LU con pivoteo.
 - `matriz.sistema(A, bb)`
Resuelve el sistema. Devuelve una lista de valores.
 - `matriz.vector(A: Matriz, v: Vector)`
Realiza el producto Av .
- ### Biblioteca metodos
- `metodos.cero(f, a, b)`
Determina un cero de la función f en el intervalo (a, b) . Asume que $f(a) \cdot f(b) < 0$.
 - `metodos.cuadratica(a, b, c)`
Determina, para un discriminante no-negativo, las soluciones reales.
 - `metodos.derivada(f, x0, n=1, delta=1e-6)`
Aproximación a la n -ésima derivada de la función f en el punto x_0 .
 - `metodos.integral(f, a, b, eps = 1e-12, prof = math.inf)`

Método adaptativo y recursivo que aproxima la integral de f entre a y b . **eps** es el máximo error relativo aceptado y **prof** la máxima profundidad en la recursividad.

```
■ metodos.newton(f, fp, x0, nmax = math.inf,
                 eps = 1e-16)
```

Método de Newton, donde se da la función, la derivada, el valor inicial, y opcionalmente el máximo número de iteraciones y tal que $|f(x)| < \varepsilon$, donde x es la respuesta.

```
■ metodos.fmin(f, a, b, tries = 3, eps = 1e-6,
               delta = 0.5)
```

Método que intenta encontrar el mínimo de f en el intervalo $[a, b]$. **tries** se refiere al número de intentos *sin* que haya mejorado la solución. En cada caso evalúa el doble de puntos que en el caso anterior. Utiliza el valor de **delta** para construir una cuadrática y determinar el vértice: $\delta = \min(\delta, (x_{i+1} - x_i)/2)$. **eps** se refiere a la diferencia en términos de **delta**, tal que un nuevo vértice se considere lo suficientemente cercano para detenerse. Devuelve $(x^*, f(x^*))$.

Biblioteca util

```
■ util.lista(f, inicio, fin, args)
```

Debido a las características de *lazy evaluation*, no es posible, por ejemplo, definir variables a, b, c, k , y luego construir la lista: `ls = [(a * i**2 + b * i + c) for i in range(ini, fin)]`. El problema son los valores de a, b y c que se utilizan (el valor de k no lo es). Así que se tuvo que implementar una función que haga lo mismo:

```
f = lambda(i, args): args[0] * i**2 + args[1] * i + args[2]
ls = lista(f, ini, fin, [a, b, c])
```

Luis Ernesto Carrera Retana
20 de enero de 2022