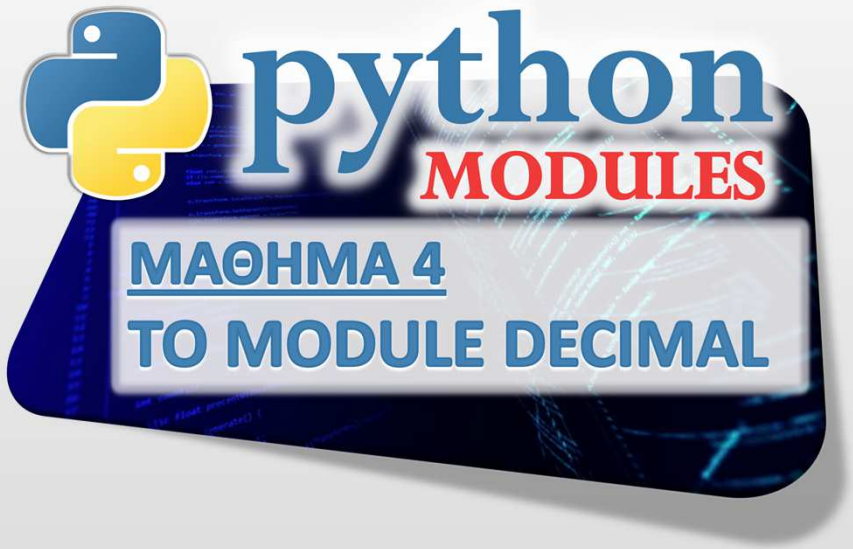

ΠΕΡΙΕΧΟΜΕΝΑ:

1. Αρχικοποίηση και αριθμητικές πράξεις
2. Ακρίβεια Πράξεων
3. Τελεστές και Μαθηματικές Συναρτήσεις
4. Ενδεικτικές Μέθοδοι του decimal
5. OPC:
    1. Format Money String

Γιώργος Μεταξάς

Σμαραγδένιος Χορηγός Μαθήματος

Ιωάννης Τ.

Χρυσός Χορηγός Μαθήματος

ΜΑΘΗΜΑ 4: Το module decimal — 1. Αρχικοποίηση και Αριθμητικές Πράξεις

modules psounis

- Το **module decimal** περιέχει την κλάση Decimal, η οποία διαχειρίζεται **δεκαδικούς αριθμούς**
  - **αλλά με απόλυτη ακρίβεια** (δεν υπάρχουν τα σφάλματα που παρατηρούνται στους float)

Αρχικοποίηση:

- Ο κατασκευαστής μπορεί να δουλέψει
  - με ακέραιο, πραγματικό, συμβολοσειρά
  - με tuple στη μορφή (πρόσημο(0-θετ ή 1-αρν), tuple ψηφίων, εκθέτης x όρου 10^x που πολλαπλασιάζεται με τον αριθμό)
  - με τις ειδικές τιμές NaN, infinity και -infinity

Παράδειγμα 1: constructor.py

```
from decimal import Decimal
D = Decimal
print(D(1))
print(D('3.14'))
print(D(3.14))
print(D('3.14000000000000012434497875801753252744674682
61718751123123'))
print(D((1,(1,4,1,5),-1)))
print(D("NaN"))
print(D("infinity"))
print(D("-infinity"))
```

Συνήθεις πράξεις:

- Υποστηρίζονται οι συνήθεις τελεστές: +, -, *, /, %, //

Παράδειγμα 2: operators.py

```
from decimal import Decimal
D=Decimal

x=D("2.2")
y=D("1.1")
print(x+y)
print(x-y)
print(x*y)
print(x/y)
print(x%y)
print(x//y)
```

Παρατήρηση:

- Επίσης υποστηρίζονται οι πράξεις μεταξύ Decimal και int, άλλά όχι μεταξύ Decimal και float (πρέπει πρώτα να μετατραπεί σε Decimal ο float)

Άσκηση 1:

Εκτελέστε πράξη (π.χ. πρόσθεση) μεταξύ Decimal και της μετατροπής float σε Decimal και διαπιστώστε προβλήματα που προκύπτουν στην ακρίβεια

- Μέσω της **getcontext()** μπορούμε να δούμε τις default παραμέτρους του module:

```
#context.py
from decimal import getcontext
print(getcontext())
```

- όπου βλέπουμε ότι:
  - prec=28: που καθορίζει ότι το πλήθος των σημαντικών ψηφίων, ως αποτέλεσμα πράξεων είναι ίσο με 28.
  - (προσοχή μόνο ως αποτέλεσμα πράξεων. Το πλήθος των ψηφίων που καθορίζεται από τον κατασκευαστή δεν επηρεάζεται από αυτήν την παράμετρο)
- Επηρεάζοντας αυτήν την παράμετρο, μπορούμε να κάνουμε πράξεις που να επιστρέφουν συγκεκριμένο πλήθος δεκαδικών ψηφίων.

Παράδειγμα 3: precision.py

```
from decimal import Decimal, getcontext
D=Decimal

getcontext().prec = 3
a=Decimal("0.4391")
b=Decimal("12.939")
print(a,b,b/a)

print(getcontext())
```

- Ο τρόπος που είδαμε στο παράδειγμα 3, καθόρισε το πλήθος των σημαντικών ψηφίων σε 3, για όλο το πρόγραμμα.
- Ωστόσο **μπορούμε να καθορίσουμε την ακρίβεια μόνο για ένα τμήμα κώδικα με τη with και τη localcontext**:

Παράδειγμα 4: with.py

```
from decimal import Decimal, localcontext
D=Decimal

with localcontext() as ctx:
   ctx.prec = 500
   res = D(10) / D(3)
   print(res)
```

Σημείωση:

- Σε πολλές εφαρμογές θέλουμε αντί για floating point ακρίβεια, ακρίβεια πλήθους δεκαδικών ψηφίων.
- Για το σκοπό αυτό, χρησιμοποιείται η μέθοδος quantize, μέσω της οποίας γίνεται η στρογγυλοποίηση σε πλήθος ψηφίων που μας ενδιαφέρει, π.χ.:

```
#quantize.py
from decimal import Decimal
D=Decimal
fpacc=Decimal(10) ** -2
```

```
a=Decimal("0.43")
b=Decimal("12.93")
print(a*b)
print((a*b).quantize(fpacc))
```

- Υποστηρίζονται όλες οι **πράξεις σχεσιακών τελεστών**:

<, <=, >, >=, ==, !=

Παράδειγμα 3: relational.operators.py

```
from decimal import Decimal
D=Decimal

print(D("12.0") == D("12"))
print(D("3.0") != D("3.000"))
print(D("1.1") < D("1.11"))
print(D("1.1") <= D("1.11"))
print(D("1.1") > D("1.11"))
print(D("1.1") >= D("1.11"))
```

- Καθώς και οι τελεστές επαυξημένης καταχώρησης:

+=, -=, *=, /=, //=, %=, **=

Παράδειγμα 5: augmented.assignent.py

```
# augmented.assigment.py
from decimal import Decimal
D=Decimal

x=D("0.4")
x**=D("0.1")
print(x)
```

- Οι decimals δεν συνεργάζονται αποδοτικά με το module math.
- Π.χ. στον ακόλουθο κώδικα, η τελική επιστρεφόμενη τιμή είναι float και όχι decimal.

```
# log.py
from decimal import Decimal
from math import ln
D=Decimal

print(type(ln(D("12.0"))))
```

- Για να υπερβεί αυτό το εμπόδιο, η κλάση Decimal χρησιμοποιεί έναν ιδιαίτερο τρόπο για να υπολογίσει συνήθεις συναρτήσεις, αφού τις ορίζει ως μεθόδους της. Έτσι, π.χ. αν θέλουμε να υπολογίσουμε το
  - ln(x)
- Γράφουμε:
  - x.ln()

Παράδειγμα 6: ln.syntax.py

```
from decimal import Decimal

D=Decimal
d=D("0.1")
print(d.ln(), type(d.ln()))
```

Μέθοδοι για συνήθεις μαθηματικές συναρτήσεις

συνάρτηση	Χρήση
exp()	x.exp() απεικονίζει το e^x
ln()	x.ln() απεικονίζει το ln(x)
log10()	x.log10() απεικονίζει το $\log_{10}(x)$
sqrt()	x.sqrt() απεικονίζει τη $\sqrt{x}$

Μέθοδοι για λογικές πράξεις

Εφαρμόζονται σε αριθμούς που αποτελούνται μόνο από 0 και 1:

συνάρτηση	Χρήση
logical_and()	x.logical_and(y) x and y
logical_or()	x.logical_or(y) x or y
logical_xor()	x.logical_xor(y) x xor y
logical_invert()	x.logical_invert() not x

Παράδειγμα 7: methods.py

```
from decimal import Decimal
D=Decimal

x=D(100)
print(x.exp())
print(x.ln())
print(x.log10())
print(x.sqrt())
```

Παράδειγμα 8: logical.py

```
from decimal import Decimal, getcontext
D=Decimal
x=D(1)
y=D(110)
print(x.logical_and(y))
print(x.logical_or(y))
print(x.logical_xor(y))
getcontext().prec=8
print(x.logical_invert())
```

Παρατήρηση: Υπάρχουν ακόμη αρκετές δεκάδες μέθοδοι:

- Περισσότερα: https://docs.python.org/3/library/decimal.html

```
# source(mod): https://docs.python.org/3/library/decimal.html

from decimal import Decimal
D=Decimal

def moneyfmt(value, places=2, curr='', sep=',', dp='.',
             pos='', neg='-', trailneg=''):
    """Convert Decimal to a money formatted string.

    places:  required number of places after the decimal point
    curr:    optional currency symbol before the sign (may be blank)
    sep:     optional grouping separator (comma, period, space, or
blank)
    dp:      decimal point indicator (comma or period)
             only specify as blank when places is zero
    pos:     optional sign for positive numbers: '+', space or blank
    neg:     optional sign for negative numbers: '-', '(', space or blank
    trailneg:optional trailing minus indicator:  '-', ')', space or blank
    """
    q = Decimal(10) ** -places      # 2 places --> '0.01'
    sign, digits, exp = value.quantize(q).as_tuple()
    result = []
    digits = list(map(str, digits))
    build, next = result.append, digits.pop
    if sign:
        build(trailneg)
    for i in range(places):
        build(next() if digits else '0')
    if places:
        build(dp)
    if not digits:
        build('0')
    i = 0
    while digits:
        build(next())
        i += 1
        if i == 3 and digits:
            i = 0
            build(sep)
    build(curr)
    build(neg if sign else pos)
    return ''.join(reversed(result))

d = D('-1234567.8901')
print(moneyfmt(d, curr='$'))
print(moneyfmt(d, places=0, sep='.', dp='', neg='', trailneg='-'))
print(moneyfmt(d, curr='$', neg='(', trailneg=')'))
print(moneyfmt(Decimal(123456789), sep=' '))
print(moneyfmt(Decimal('-0.02'), neg='<', trailneg='>'))
```