1. Error-boodschappen

De python error-boodschappen kunnen gebruikt worden om de invoer te specifiëren.

Drie voorbeelden van error-boodschappen:

1. NameError: het gebruik van een niet gedefinieerde variabele, b.v. in een print()-statement

```
test1.py
print(x)

Python Shell
NameError: name 'x' isn't defined
```

2. ValueError: het uitvoeren van een functie op een argument van het verkeerde type

```
test2.py
int("Python")

Python Shell
ValueError: invalid syntax for integer with base 1 0: 'Python'
```

3. TypeError: het uitvoeren van een operatie op het verkeerde type

```
test3.py
w="Python"
w+=1

Python Shell
TypeError: can't convert 'int' object to str implicitly
```

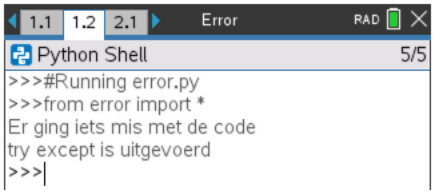
2. try & except

De statements try en except doen het volgende:

- try error-code testen
- except uitvoeren van code in geval van error

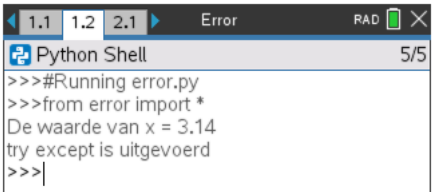
We verduidelijken even deze structuur. Bij het runnen van de onderstaande code krijg je geen error-melding.
try: probeert de code uit te voeren en in het geval van een error wordt de code in het except:-blok uitgevoerd.

```
try:
    print("De waarde van x =",x)
Except:
    print("Er ging iets mis met de code")
print("try except is uitgevoerd")
```



Indien er geen error optreedt, m.a.w. de variabele x is gedeclareerd, wordt de code in het try:-blok uitgevoerd

```
x=3.14
try:
    print("De waarde van x =",x)
Except:
    print("Er ging iets mis met de code")
print("try except is uitgevoerd")
```



Voor het except-statement kan het error-type gespecifieerd worden, b.v. except NameError: voor bovenstaande code.

3. Input op maat

Met de statements try en except kunnen we de input specifiëren en vermijden dat we een error-boodschap krijgen bij hier invoeren van een verkeerd type.

In het volgende programma willen we de input beperken tot een geheel getal n tussen 0 en 5, $0 \leq n \leq 5$, en blijven vragen naar input tot de ingegeven waarde correct is.

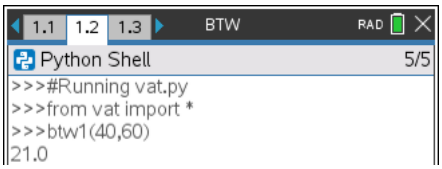
```
while True:
    getal=input("Getal n met 0 ≤ n ≤ 5: ")
    try:
        n=int(getal)
    except ValueError:
        print("VERKEERDE INPUT - Probeer opnieuw!")
        continue
    if n < 0 or n > 5:
        print("BUITEN DE GRENZEN - Probeer opnieuw!")
        continue
    break

print("De waarde van het getal n =",n)
```

Het error-type, ValueError, kan ook hier weggelaten worden.

4. *args

Veronderstel dat we de BTW willen berekenen op som van de netto-prijs van een aantal producten:

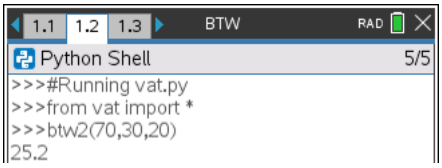
```
def btw1(a,b):
    return sum((a,b))*0.21
```



Wat als we de btw op meer dan twee producten willen berekenen?

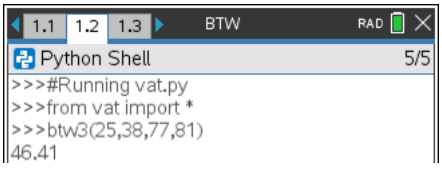
Een manier om dit te doen is het aantal argumenten te verhogen en een standaard waarde toe te kennen aan de argumenten.

```
def btw2(a=0,b=0,c=0,d=0,e=0):
    return sum((a,b,c,d,e))*0.21
```



Het starten van een parameter van een functie met een asterisk, *, maakt het mogelijk voor de functie om een willekeurig aantal argumenten te gebruiken. De functie beschouwt de argumenten als een tuple.

```
def btw3(*args):
    return sum(args)*0.21
```



5. Lambda

Lambda-uitdrukkingen zijn krachtige Python-tools die het toe laten om ad hoc anonieme functies te definiëren, zonder gebruik te maken van def. De structuur van lambda's is één enkele uitdrukking en niet een blok statements.

De onderstaande functie square() ziet er in lambda-vorm als volgt uit:

$f(x) = x^2$ $\qquad\qquad$ $x \longmapsto x^2$

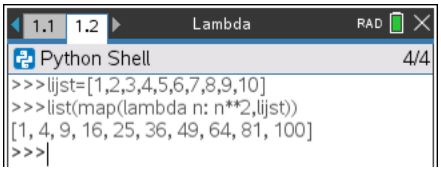
```
def square(n):
    kwad=n**2          <------------>   lambda num : num**2
    return kwad
```

Normaal geef je aan een lambda-uitdrukking geen naam, maar toch even om een lambda-uitdrukking te demonstreren:

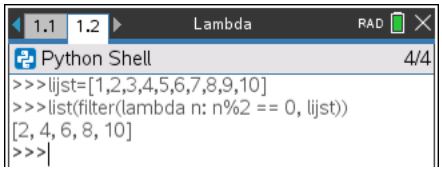
```
square = lambda num: num**2
```

Hoe gebruik je een lamda-uitdrukking dan wel? Soms moet je een functie maar één keer uitvoeren in een programma en graag zonder een formele definitie van de functie. Dan komt een lambda-uitdrukking goed van pas.

list(map(lambda n: n**2, lijst))



list(filter(lambda n: n%2 == 0, lijst))



6. Wat meer Palindroom-code ...

... gebruikmakend van string-functionaliteit

Code1

```
alfabet="abcdefghijklmnopqrstuvwxyz"

def strip_tekst(tekst):
    tekst = tekst.lower()
    str = ""
    for t in tekst:
        if t in alfabet:
            str = str+t
    return str

a=strip_tekst("Nelli plaatst op 'n parterretrap 'n pot staalpillen")
print(a)

def is_palindroom(tekst):
    l = len(tekst)
    for i in range(l/2):
        if tekst[i] != tekst[l-i-1]:
            return False
    return True

a=strip_tekst("Nelli plaatst op 'n parterretrap 'n pot staalpillen")
print(is_palindroom(a))
```

Code 2

```
def is_palindroom(tekst):
    a=list(tekst)
    b=a.copy()
    b.reverse()
    if b==a:
        return True
    else:
        return False

print(is_palindroom("meetsysteem"))
```

7. Opdrachten

7.1. Integer-input $n \geq 0$

Schrijf de code voor een input die enkel een geheel getal $n \geq 0$ aanvaardt en blijft vragen voor input totdat een aanvaardbare waarde is ingegeven:

- Indien de input geen geheel getal is, print "Verkeerde input – Probeer opnieuw",
- Indien de input een negatief geheel getal is, print "Negatief getal – Enter een positief getal".

Bereken m.b.v. van deze input-code en de onderstaande code $n!$ voor $n \geq 0$.

```
def fac(n):
    if n==0:
        return 1
    else:
        faculteit = 1
        for i in range(1,n+1):
            faculteit *= i
        return faculteit


# Input van enkel een geheel getal n ≥ 0.
......

print("De faculteit .....gem )
```

7.2. Dobbelen met Pascal


Blaise Pascal

De Franse Chevalier de Méré ontdekte bij het dobbelen dat het kansrijker was om in vier worpen met één dobbelsteen minstens een keer zes te gooien, dan in 24 worpen met twee dobbelstenen minstens een keer dubbel zes.

Daar hij dit helemaal niet verwachtte, vroeg hij uitleg aan Blaise Pascal (1623-1662).

Pascal vertelde hem dat de kans op winst respectievelijk 0,518 en 0,491 zijn.

De volgende codes simuleert beide dobbel-experimenten:

- worpen = [randint(1,6) for i in range(0,4)]
  6 in worpen
- worpen = [(randint(1,6),randint(1,6) for i in range(0,4)]
  (6,6) in worpen

Schrijf programma's die beide dobbel-experimenten uitvoert, met:

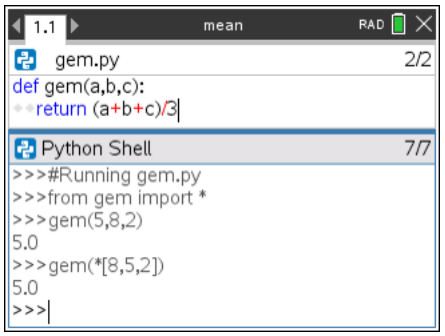
- na iedere uitvoering de optie om verder te stoppen (0) of verder te spelen (1),
- het aantal experimenten wordt bij gehouden, aantal,
- het aantal keren gewonnen wordt bijgehouden, winst.

Benader/simuleer de kans op winst voor beide experimenten.

7.3. Gemiddelde van getallen

Definieer een functie die van een willekeurig aantal getallen het gemiddelde berekent door b.v. gebruik te maken van *data als argument.

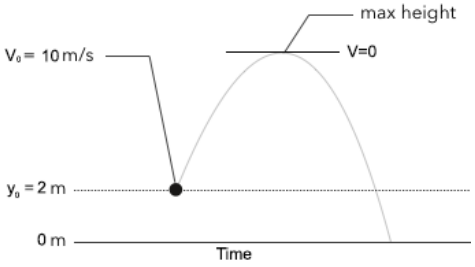
Omgekeerd kan een lijst (of tuple) voorafgegaan door een asteriks, *, gebruikt worden als de argumenten van een functie.



7.4. Gravitatie

Schrijf een programma om de maximale hoogte te vinden van een bal die recht omhoog wordt gegooid van af een hoogte van 2 meter met een beginsnelheid van 10 m/s.

$$y = \frac{1}{2}gt^2 + v_{y_0}y + y_0 \quad \text{met } g = -9{,}81\frac{m}{s^2}$$



- Stap 1
  Creëer d.m.v. lijstcomprehensie een lijst van tijdstippen, iedere 0,05 s.
- Stap 2
  Bereken voor de lijst uit Stap 1 de hoogte m.b.v. map() en lambda.
- Stap 3
  Bepaal het maximum van de berekende hoogtes met de max()-functie.
- Stap 4
  Bepaal het tijdstip behorende bij de maximale hoogte – lijst.index(element).
- Stap 5
  Exporteer de lijsten met de tijdstippen en hoogtes naar TI-Nspire-lijsten en teken hiermee een scatterplot in Graphs. Wanneer raakt de bal de grond?